



SAPIENZA  
UNIVERSITÀ DI ROMA

---

# Algoritmi

## Esercizi 4 — Analisi degli algoritmi

---

Esercizi svolti sui capitoli 11–13: dimensione dell'input, tempo di esecuzione, correttezza. Ogni esercizio ha la soluzione completa.

**Fabio Furini**

[fabiofurini.github.io](https://fabiofurini.github.io)

Dipartimento di Ingegneria informatica, automatica e gestionale — Sapienza Università di Roma

Versione online e materiale scaricabile:

[fabiofurini.github.io/algoritmi/](https://fabiofurini.github.io/algoritmi/)  
[github.com/fabiofurini/algoritmi](https://github.com/fabiofurini/algoritmi)

Anno accademico 2026–2027

---

# Indice

---

|    |                       |    |
|----|-----------------------|----|
| 11 | Dimensione dell'input | 2  |
| 12 | Tempo di esecuzione   | 12 |
| 13 | Correttezza           | 30 |



## Dimensione dell'input

Negli esercizi di questo capitolo la dimensione dell'input si calcola con le formule del capitolo «Dimensione dell'input», riassunte qui sotto.

La dimensione dell'input  $|I|$  di un'istanza  $I$  è il numero totale di bit necessari per rappresentare  $I$  in binario:

- un numero naturale  $n \geq 1$ :

$$|I| = \lfloor \log_2 n \rfloor + 1 = \Theta(\log_2 n);$$

- due numeri naturali  $n, m \geq 1$ :

$$|I| = (\lfloor \log_2 n \rfloor + 1) + (\lfloor \log_2 m \rfloor + 1) = \Theta(\log_2 \max\{n, m\});$$

- un array  $a[1..n]$  di numeri naturali (una componente uguale a 0 usa 1 bit):

$$|I| = \sum_{i=1}^n (\lfloor \log_2(\max\{1, a[i]\}) \rfloor + 1), \quad n \leq |I| \leq n(1 + \log_2 M),$$

dove  $M = \max\{1, a[1], a[2], \dots, a[n]\}$ .

Un numero naturale si scrive con esattamente  $k$  bit se e solo se  $2^{k-1} \leq n \leq 2^k - 1$ .

### Esercizio 11.1 — Dimensione dell'input (E1)

Consideriamo le seguenti istanze di input:

- (a) i numeri naturali  $n = 1, n = 13, n = 255, n = 256, n = 1000, n = 10^6$ ;
- (b) le coppie di numeri naturali  $(n, m) = (48, 18)$  e  $(n, m) = (1000, 7)$ ;
- (c) gli array  $[5, 2, 4, 1, 3], [1, 1, 1, 1], [1024, 1, 1], [0, 7, 8]$ .

**Domande**

1. Calcolare la dimensione dell'input  $|I|$  di ogni istanza in (a), e scrivere la rappresentazione binaria di 13, 255 e 256.
2. Calcolare la dimensione dell'input delle due istanze in (b).
3. Calcolare la dimensione dell'input dei quattro array in (c). Per ogni array confrontare  $|I|$  con il numero  $n$  di componenti e con il limite superiore  $n(1 + \log_2 M)$ , dove  $M = \max\{1, a[1], a[2], \dots, a[n]\}$ .
4. Qual è il più grande numero naturale rappresentabile con  $|I| = k$  bit? E il più piccolo numero naturale  $n \geq 1$  la cui dimensione dell'input è  $k$ ?

## Soluzione

1. Usando  $|I| = \lfloor \log_2 n \rfloor + 1$ :

| $n$    | potenze di 2                | $\lfloor \log_2 n \rfloor$ | $ I $ |
|--------|-----------------------------|----------------------------|-------|
| 1      | $2^0 \leq 1 < 2^1$          | 0                          | 1     |
| 13     | $2^3 \leq 13 < 2^4$         | 3                          | 4     |
| 255    | $2^7 \leq 255 < 2^8$        | 7                          | 8     |
| 256    | $2^8 \leq 256 < 2^9$        | 8                          | 9     |
| 1000   | $2^9 \leq 1000 < 2^{10}$    | 9                          | 10    |
| $10^6$ | $2^{19} \leq 10^6 < 2^{20}$ | 19                         | 20    |

Rappresentazioni binarie:  $13 = (1101)_2$  (4 bit),  $255 = (11111111)_2$  (8 bit),  $256 = (100000000)_2$  (9 bit). Si noti che 255 e 256 differiscono di 1, ma le loro dimensioni dell'input differiscono di un bit, perché  $256 = 2^8$  è una potenza di 2.

2. Per (48, 18):  $2^5 \leq 48 < 2^6$  e  $2^4 \leq 18 < 2^5$ , quindi

$$|I| = (5 + 1) + (4 + 1) = 6 + 5 = 11.$$

Per (1000, 7):  $2^9 \leq 1000 < 2^{10}$  e  $2^2 \leq 7 < 2^3$ , quindi

$$|I| = (9 + 1) + (2 + 1) = 10 + 3 = 13.$$

3. Bit di ogni componente e totale:

| array           | bit per componente  | $ I $ | $n$ | $n(1 + \log_2 M)$               |
|-----------------|---------------------|-------|-----|---------------------------------|
| [5, 2, 4, 1, 3] | $3 + 2 + 3 + 1 + 2$ | 11    | 5   | $5(1 + \log_2 5) \approx 16.61$ |
| [1, 1, 1, 1]    | $1 + 1 + 1 + 1$     | 4     | 4   | $4(1 + 0) = 4$                  |
| [1024, 1, 1]    | $11 + 1 + 1$        | 13    | 3   | $3(1 + 10) = 33$                |
| [0, 7, 8]       | $1 + 3 + 4$         | 8     | 3   | $3(1 + 3) = 12$                 |

In tutti i casi  $n \leq |I| \leq n(1 + \log_2 M)$ , in accordo con  $|I| = \Omega(n)$  e  $|I| = O(n(1 + \log_2 M))$ . Il limite superiore è stretto per [1, 1, 1, 1] (componenti tutte uguali), mentre è lontano da  $|I|$  per [1024, 1, 1], dove solo una componente è grande.

4. Con  $k$  bit il più grande numero rappresentabile è

$$\underbrace{(11 \dots 1)}_k_2 = 2^k - 1,$$

e il più piccolo numero  $n \geq 1$  con  $\lfloor \log_2 n \rfloor + 1 = k$  è  $n = 2^{k-1} = (1 \underbrace{0 \dots 0}_{k-1})_2$ . Dunque i numeri con dimensione dell'input esattamente  $k$  sono quelli con

$$2^{k-1} \leq n \leq 2^k - 1.$$

Ad esempio, per  $k = 8$  sono i numeri da 128 a 255.

### Esercizio 11.2 — Dimensione dell'input (E2)

Consideriamo le seguenti famiglie di istanze di input, parametrizzate da un numero naturale  $n \geq 1$  (oppure  $k \geq 1$ ):

- (a) il singolo numero naturale  $2^k$ , e il singolo numero naturale  $2^k - 1$ ;
- (b) l'array  $a[1..n]$  con  $a[i] = i$  per ogni  $i \in \{1, 2, \dots, n\}$ ;
- (c) l'array  $a[1..n]$  con  $a[i] = 2^i$  per ogni  $i \in \{1, 2, \dots, n\}$ ;
- (d) l'array  $a[1..n]$  con  $a[i] = n$  per ogni  $i \in \{1, 2, \dots, n\}$ ;
- (e) un array  $a[1..n]$  le cui componenti sono numeri naturali minori di  $2^{32}$ , così che ciascuna di esse può essere scritta con al più 32 bit.

#### Domande

1. Calcolare esattamente la dimensione dell'input delle istanze in (a), in funzione di  $k$ .
2. Calcolare esattamente la dimensione dell'input degli array in (c), ed esprimerla asintoticamente in funzione di  $n$ .
3. Esprimere asintoticamente la dimensione dell'input degli array in (d) ed (e).
4. Per gli array in (b), calcolare  $|I|$  per  $n = 1, 2, 3, 4, 8, 16$ , e dimostrare che  $|I| = \Theta(n \log_2 n)$ .

#### Soluzione

1. Poiché  $\log_2 2^k = k$ , il numero  $2^k$  ha dimensione dell'input

$$|I| = \lfloor \log_2 2^k \rfloor + 1 = k + 1.$$

Poiché  $2^{k-1} \leq 2^k - 1 < 2^k$ , si ha  $\lfloor \log_2(2^k - 1) \rfloor = k - 1$  e

$$|I| = k.$$

In entrambi i casi  $|I| = \Theta(k)$ , mentre il valore numerico è  $\Theta(2^k)$ : il valore è esponenziale nella dimensione dell'input.

2. La componente  $a[i] = 2^i$  richiede  $\lfloor \log_2 2^i \rfloor + 1 = i + 1$  bit, quindi

$$|I| = \sum_{i=1}^n (i+1) = \frac{n(n+1)}{2} + n = \frac{n(n+3)}{2}.$$

Ad esempio, per  $n = 4$ :  $2 + 3 + 4 + 5 = 14 = \frac{4 \cdot 7}{2}$ . Poiché

$$\lim_{n \rightarrow \infty} \frac{n(n+3)/2}{n^2} = \lim_{n \rightarrow \infty} \frac{1}{2} \left( 1 + \frac{3}{n} \right) = \frac{1}{2} \in \mathbb{R}_{>0},$$

si ha  $|I| = \Theta(n^2)$  per il criterio del limite. In modo diretto, poiché  $n^2 \leq n(n+3) \leq 4n^2$  per ogni  $n \geq 1$ , le costanti  $c_1 = \frac{1}{2}$ ,  $c_2 = 2$  e  $n_1 = 1$  danno  $\frac{1}{2}n^2 \leq |I| \leq 2n^2$ . Qui le componenti **non** usano un numero costante di bit, e la dimensione dell'input è quadratica nel numero di componenti.

3. (d) Ogni componente è uguale a  $n$  e usa  $\lfloor \log_2 n \rfloor + 1$  bit, quindi

$$|I| = n(\lfloor \log_2 n \rfloor + 1) = \Theta(n \log_2 n),$$

perché  $\lfloor \log_2 n \rfloor + 1 = \Theta(\log_2 n)$  (Osservazione sulla dimensione dell'input di un numero naturale), moltiplicato per  $n$ .

- (e) Ogni componente usa almeno 1 e al più 32 bit, quindi

$$n \leq |I| \leq 32n,$$

e  $|I| = \Theta(n)$ . Questa è l'ipotesi "ogni componente usa un numero costante di bit" del corollario sulle componenti di dimensione costante. (Se ogni componente è memorizzata in una parola di memoria di esattamente 32 bit, il numero di bit memorizzati è  $32n$ , di nuovo  $\Theta(n)$ .)

4. La componente  $a[i] = i$  usa  $\lfloor \log_2 i \rfloor + 1$  bit:

| $n$ | $ I $                                |
|-----|--------------------------------------|
| 1   | 1                                    |
| 2   | $1 + 2 = 3$                          |
| 3   | $1 + 2 + 2 = 5$                      |
| 4   | $1 + 2 + 2 + 3 = 8$                  |
| 8   | $1 + 2 + 2 + 3 + 3 + 3 + 3 + 4 = 21$ |
| 16  | $21 + 7 \cdot 4 + 5 = 54$            |

**Limite superiore.** Poiché  $i \leq n$ , ogni componente usa al più  $\lfloor \log_2 n \rfloor + 1 \leq \log_2 n + 1 \leq 2 \log_2 n$  bit per  $n \geq 2$ , quindi

$$|I| \leq 2n \log_2 n, \quad \forall n \geq 2.$$

**Limite inferiore.** Ogni componente usa almeno  $\log_2 i$  bit (poiché  $\lfloor \log_2 i \rfloor + 1 \geq \log_2 i$ ). Tenendo solo gli indici  $i \geq \lceil n/2 \rceil$ , che sono almeno  $n/2$  e sono almeno  $n/2$  in numero,

$$|I| \geq \sum_{i=\lceil n/2 \rceil}^n \log_2 i \geq \frac{n}{2} \log_2 \frac{n}{2} = \frac{n}{2} (\log_2 n - 1) \geq \frac{n}{4} \log_2 n,$$

dove l'ultima disuguaglianza vale quando  $\log_2 n - 1 \geq \frac{1}{2} \log_2 n$ , cioè per  $n \geq 4$ .

Quindi, con  $c_1 = \frac{1}{4}$ ,  $c_2 = 2$  e  $n_1 = 4$ ,

$$0 \leq \frac{1}{4} n \log_2 n \leq |I| \leq 2 n \log_2 n, \quad \forall n \geq 4,$$

cioè  $|I| = \Theta(n \log_2 n)$ . (Verifica: per  $n = 16$ ,  $\frac{1}{4} \cdot 16 \cdot 4 = 16 \leq 54 \leq 2 \cdot 16 \cdot 4 = 128$ .)

### Esercizio 11.3 — Dimensione dell'input (E3)

Consideriamo il seguente algoritmo, e richiamiamo l'algoritmo **PrimalityTest** (divisione per tentativi per  $d = 2, 3, \dots, \lfloor \sqrt{n} \rfloor$ ), il cui numero di passi nel caso peggiore ( $n$  primo) è  $S(n) = 3\lfloor \sqrt{n} \rfloor + 1$ , con il corpo del suo ciclo while eseguito  $\lfloor \sqrt{n} \rfloor - 1$  volte.

---

**Algorithm:** SumUpTo

---

**Input :** Un numero naturale  $n \geq 1$

**Output:**  $1 + 2 + \dots + n$

---

```

1 function SumUpTo(n)
2   s := 0
3   i := 1
4   while i ≤ n do
5     s ← s + i
6     i ← i + 1
7   end
8   return s
9 end
```

---

#### Domande

1. Calcolare il numero di passi  $S(n)$  di **SumUpTo** e determinare il suo tempo di esecuzione nel caso peggiore  $T(|I|)$ . L'algoritmo è polinomiale?
2. Proporre un algoritmo che calcola lo stesso output con un numero costante di passi (si assuma che le operazioni aritmetiche richiedano un numero costante di passi). È polinomiale?
3. Dimostrare che **PrimalityTest** è esponenziale in  $|I|$ , anche se  $S(n) = \Theta(\sqrt{n})$  è "sublineare" nel valore  $n$ .
4. Dare un limite superiore al numero di iterazioni del ciclo while di **PrimalityTest** per un numero primo  $n$  di 32 bit e di 64 bit. Che cosa succede al numero di iterazioni quando la dimensione dell'input raddoppia?

## Soluzione

1. Il numero di volte in cui ogni riga di codice viene eseguita è:

- La riga 2 è eseguita 1 volta
- La riga 3 è eseguita 1 volta
- La riga 4 è eseguita  $n + 1$  volte
- La riga 5 è eseguita  $n$  volte
- La riga 6 è eseguita  $n$  volte
- La riga 8 è eseguita 1 volta

Quindi

$$S(n) = 1 + 1 + (n + 1) + n + n + 1 = 3n + 4 = \Theta(n).$$

L'input è un solo numero naturale, quindi  $|I| = \lfloor \log_2 n \rfloor + 1 = \Theta(\log_2 n)$ : esistono  $c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$c_1 \log_2 n \leq |I| \leq c_2 \log_2 n \iff 2^{\frac{1}{c_2}|I|} \leq n \leq 2^{\frac{1}{c_1}|I|}.$$

Poiché  $a_1 n \leq S(n) \leq a_2 n$  per certi  $a_1, a_2 > 0$ ,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{quindi} \quad T(|I|) = 2^{\Theta(|I|)},$$

poiché i fattori costanti  $a_1$  e  $a_2$  sono assorbiti nell'esponente (Osservazione «fattori costanti nell'esponente» del capitolo). Il caso peggiore si può anche calcolare esattamente. Gli input di dimensione  $|I|$  sono i numeri  $n$  con  $2^{|I|-1} \leq n \leq 2^{|I|} - 1$ , e  $S(n) = 3n + 4$  è massimo per  $n = 2^{|I|} - 1$ , quindi

$$T(|I|) = 3(2^{|I|} - 1) + 4 = 3 \cdot 2^{|I|} + 1.$$

SumUpTo è lineare nel valore  $n$  ma **esponenziale** nella dimensione dell'input: non è polinomiale.

2. Per la formula di Gauss,  $1 + 2 + \dots + n = \frac{n(n+1)}{2}$ . L'algoritmo “**return**  $n(n+1)/2$ ” esegue un numero costante di passi (una somma, un prodotto, una divisione, un return), quindi  $T(|I|) = \Theta(1) = O(|I|^0)$ : è polinomiale.

3. L'input è un solo numero naturale, quindi come sopra  $2^{\frac{1}{c_2}|I|} \leq n \leq 2^{\frac{1}{c_1}|I|}$ . Estraendo le radici quadrate (tutti i termini sono positivi),

$$2^{\frac{1}{2c_2}|I|} \leq \sqrt{n} \leq 2^{\frac{1}{2c_1}|I|}.$$

Poiché  $S(n) = \Theta(\sqrt{n})$ , esistono  $a_1, a_2 > 0$  con  $a_1 \sqrt{n} \leq S(n) \leq a_2 \sqrt{n}$ , quindi

$$a_1 2^{\frac{1}{2c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{2c_1}|I|}, \quad \text{quindi} \quad T(|I|) = 2^{\Theta(|I|)},$$

poiché i fattori costanti  $a_1$  e  $a_2$  sono assorbiti nell'esponente (Osservazione «fattori costanti nell'esponente» del capitolo). Essere sublineare in  $n$  non aiuta:  $\sqrt{n} = 2^{\frac{1}{2} \log_2 n}$  è ancora esponenziale nel numero di bit  $\log_2 n$ . L'algoritmo non è polinomiale: per ogni costante  $k$ ,  $\lim_{|I| \rightarrow \infty} 2^{c|I|}/|I|^k = +\infty$  per ogni  $c > 0$ .

4. Il ciclo while è eseguito  $\lfloor \sqrt{n} \rfloor - 1$  volte per un  $n$  primo.

- 32 bit:  $n < 2^{32}$ , quindi  $\sqrt{n} < 2^{16} = 65\,536$  e il ciclo esegue meno di 65 536 iterazioni.
- 64 bit:  $n < 2^{64}$ , quindi  $\sqrt{n} < 2^{32} = 4\,294\,967\,296$  e il ciclo esegue fino a circa  $4.3 \cdot 10^9$  iterazioni.

Quando la dimensione dell'input raddoppia (da 32 a 64 bit) il limite sul numero di iterazioni viene **elevato al quadrato** ( $2^{16} \rightarrow 2^{32}$ ), mentre per un algoritmo polinomiale con  $T(|I|) = \Theta(|I|^k)$  verrebbe solo moltiplicato per la costante  $2^k$ .

#### Esercizio 11.4 — Dimensione dell'input (E4)

Consideriamo i due algoritmi seguenti. Entrambi contengono due cicli annidati.

---

##### Algorithm: CountPairs

---

**Input** : Un numero naturale  $n \geq 1$

**Output**: Il numero di coppie  $(i, j)$  con  $i, j \in \{1, 2, \dots, n\}$

---

```

1 function CountPairs(n)
2   c := 0
3   i := 1
4   while i ≤ n do
5     j := 1
6     while j ≤ n do
7       c ← c + 1
8       j ← j + 1
9     end
10    i ← i + 1
11  end
12  return c
13 end
```

---

**Algorithm:** CountInversions**Input** : Un array  $a[1..n]$  di  $n \geq 1$  numeri naturali**Output**: Il numero di coppie  $i < j$  con  $a[i] > a[j]$ 

```

1 function CountInversions(a, n)
2   c := 0
3   i := 1
4   while i ≤ n - 1 do
5     j := i + 1
6     while j ≤ n do
7       if a[i] > a[j] then
8         c ← c + 1
9       end
10      j ← j + 1
11    end
12    i ← i + 1
13  end
14  return c
15 end

```

**Domande**

1. Calcolare il numero di passi  $S(n)$  di **CountPairs** e il suo tempo di esecuzione nel caso peggiore  $T(|I|)$ .
2. Calcolare il numero di passi  $S(n)$  di **CountInversions** nel caso peggiore e il suo tempo di esecuzione nel caso peggiore  $T(|I|)$ , in generale e sotto l'ipotesi che ogni componente usi un numero costante di bit.
3. Entrambi gli algoritmi hanno  $S(n) = \Theta(n^2)$ . Spiegare perché uno è polinomiale e l'altro è esponenziale nella dimensione dell'input.
4. Supponiamo che **CountInversions** riceva anche il numero  $n$  come parte dell'input. Calcolare  $|I|$  per  $n = 5$ ,  $a = [5, 2, 4, 1, 3]$ , e spiegare perché la conclusione della domanda 2 non cambia.

**Soluzione**

1. Il numero di volte in cui ogni riga di codice viene eseguita è:

- Le righe 2, 3 e 12 sono eseguite 1 volta ciascuna
- La riga 4 è eseguita  $n + 1$  volte
- Le righe 5 e 10 sono eseguite  $n$  volte ciascuna
- La riga 6 è eseguita  $n(n + 1)$  volte
- Le righe 7 e 8 sono eseguite  $n^2$  volte ciascuna

Quindi

$$S(n) = 3 + (n + 1) + 2n + n(n + 1) + 2n^2 = 3n^2 + 4n + 4 = \Theta(n^2).$$

L'input è un solo numero naturale, quindi  $|I| = \Theta(\log_2 n)$  e, come per ogni input costituito da un solo numero,  $2^{\frac{1}{c_2}|I|} \leq n \leq 2^{\frac{1}{c_1}|I|}$  per certi  $c_1, c_2 > 0$ . Elevando al quadrato,

$$2^{\frac{2}{c_2}|I|} \leq n^2 \leq 2^{\frac{2}{c_1}|I|},$$

e poiché  $a_1 n^2 \leq S(n) \leq a_2 n^2$  per certi  $a_1, a_2 > 0$ , assorbendo i fattori costanti nell'esponente (Osservazione «fattori costanti nell'esponente» del capitolo) otteniamo

$$T(|I|) = 2^{\Theta(|I|)} :$$

**CountPairs** è esponenziale nella dimensione dell'input.

2. Nel caso peggiore (array in ordine strettamente decrescente) la riga 8 è eseguita a ogni iterazione del ciclo interno. Per un  $i$  fissato, il ciclo interno esegue  $n - i$  iterazioni, per un totale di  $\sum_{i=1}^{n-1} (n - i) = \frac{n(n-1)}{2}$ . Il numero di volte in cui ogni riga viene eseguita è:

- Le righe 2, 3 e 14 sono eseguite 1 volta ciascuna
- La riga 4 è eseguita  $n$  volte
- Le righe 5 e 12 sono eseguite  $n - 1$  volte ciascuna
- La riga 6 è eseguita  $\frac{n(n-1)}{2} + (n - 1)$  volte
- Le righe 7, 8 e 10 sono eseguite  $\frac{n(n-1)}{2}$  volte ciascuna

Quindi

$$S(n) = 3 + n + 3(n - 1) + 4 \cdot \frac{n(n - 1)}{2} = 2n^2 + 2n = \Theta(n^2).$$

L'input è un array, quindi  $|I| = \Omega(n)$ , cioè  $n \leq \frac{1}{c}|I|$ , e

$$S(n) \leq a n^2 \leq \frac{a}{c^2} |I|^2 \implies T(|I|) = O(|I|^2).$$

Se ogni componente usa al più  $c$  bit, con  $c$  una costante, allora  $|I| = \Theta(n)$ , e più precisamente  $n \geq \frac{1}{c}|I|$ . Per un limite inferiore, si noti che solo la riga 8 dipende dai valori: anche se non viene mai eseguita, l'algoritmo esegue

$$2n^2 + 2n - \frac{n(n-1)}{2} = \frac{3n^2 + 5n}{2} \geq \frac{3}{2} n^2 \geq \frac{3}{2c^2} |I|^2$$

passi. Ogni input di dimensione  $|I|$  richiede almeno questo numero di passi, quindi  $T(|I|) = \Omega(|I|^2)$  e, insieme al limite superiore,

$$T(|I|) = \Theta(|I|^2).$$

In entrambi i casi **CountInversions** è polinomiale (quadratico).

3. Il numero di passi è la stessa funzione di  $n$ , ma  $n$  ha una relazione diversa con la dimensione dell'input:

- in **CountPairs**,  $n$  è il **valore** dell'input, che si scrive con soli  $\Theta(\log_2 n)$  bit, quindi  $n = 2^{\Theta(|I|)}$ ;

- in `CountInversions`,  $n$  è il **numero di componenti** dell'array, e ogni componente richiede almeno un bit, quindi  $n = O(|I|)$ .

Lo stesso  $S(n)$  può corrispondere a un tempo di esecuzione polinomiale o esponenziale: dipende da come il parametro  $n$  è legato alla dimensione dell'input  $|I|$ .

4. I bit di  $n = 5$  sono  $\lfloor \log_2 5 \rfloor + 1 = 3$ , e i bit dell'array sono  $3 + 2 + 3 + 1 + 2 = 11$ , quindi

$$|I| = 3 + 11 = 14.$$

In generale  $|I| = (\lfloor \log_2 n \rfloor + 1) + \sum_{i=1}^n (\lfloor \log_2 a[i] \rfloor + 1) \geq n$ , quindi ancora  $|I| = \Omega(n)$  e  $T(|I|) = O(|I|^2)$ . Con componenti di dimensione costante,  $|I| = \Theta(\log_2 n) + \Theta(n) = \Theta(n)$ , quindi  $T(|I|) = \Theta(|I|^2)$  come prima: aggiungere  $n$  all'input aggiunge solo  $\Theta(\log_2 n)$  bit, trascurabili rispetto agli  $\Omega(n)$  bit dell'array.

## Tempo di esecuzione

Negli esercizi di questo capitolo i passi si contano con le convenzioni del capitolo «Tempo di esecuzione»: ogni esecuzione di una riga numerata dello pseudocodice (comprese le righe **return**) è un passo, la condizione di un ciclo **while** il cui corpo viene eseguito  $k$  volte viene valutata  $k + 1$  volte, e le righe di un ramo che non viene percorso non vengono eseguite. Dopo aver ottenuto il numero di passi  $S(n)$ , il tempo di esecuzione si esprime in funzione della dimensione dell'input  $|I|$  con una delle due regole seguenti.

**Regola (A): l'input è un numero naturale  $n$ .** Allora  $|I| = \Theta(\log_2 n)$ : esistono costanti  $c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$c_1 \log_2 n \leq |I| \leq c_2 \log_2 n \quad \Longleftrightarrow \quad 2^{\frac{1}{c_2}|I|} \leq n \leq 2^{\frac{1}{c_1}|I|},$$

poiché  $x \mapsto 2^x$  è strettamente crescente e  $2^{\log_2 n} = n$ . Se  $S(n) = \Theta(n)$ , esistono costanti  $a_1, a_2 > 0$  tali che  $a_1 n \leq S(n) \leq a_2 n$  per ogni  $n$  sufficientemente grande, e quindi

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{cioè} \quad T(|I|) = 2^{\Theta(|I|)}.$$

**Regola (B): l'input è un array  $a[1..n]$  i cui elementi usano un numero costante fisso di bit.** Allora  $|I| = \Theta(n)$ : esistono costanti  $c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$c_1 n \leq |I| \leq c_2 n \quad \Longleftrightarrow \quad \frac{1}{c_2}|I| \leq n \leq \frac{1}{c_1}|I|.$$

Se  $S(n) = \Theta(n^k)$  per una costante  $k \geq 1$ , esistono costanti  $a_1, a_2 > 0$  tali che  $a_1 n^k \leq S(n) \leq a_2 n^k$  per ogni  $n$  sufficientemente grande, e quindi

$$\frac{a_1}{c_2^k} |I|^k \leq S(n) \leq \frac{a_2}{c_1^k} |I|^k, \quad \text{cioè} \quad T(|I|) = \Theta(|I|^k).$$

### Esercizio 12.1 — Tempo di esecuzione ( $E1$ )

Si consideri il seguente algoritmo, il cui input è un numero naturale  $n$ .

**Algorithm:** Algoritmo 1**Input** : Un numero naturale  $n$ **Output:** Un numero naturale  $m$ 

```

1 function Algorithm1( $n$ )
2    $m := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $m \leftarrow m + 2$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $m$ 
9 end

```

**Domande**

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

**Soluzione**

1. L'algoritmo somma 2 a  $m$  esattamente  $n$  volte, partendo da  $m = 0$ : determina il doppio di  $n$ , cioè  $2n$ .
2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm1**, dove  $n$  è il numero naturale dato in input. Per ogni valore di  $n$  c'è un solo input, quindi questo è anche il caso peggiore. Il corpo del ciclo viene eseguito per  $i = 1, 2, \dots, n$ , cioè  $n$  volte, e la condizione della riga 4 viene valutata  $n + 1$  volte:

| riga | numero di esecuzioni |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $n$                  |
| 8    | 1                    |

Quindi il numero totale di passi è

$$S(n) = 1 + 1 + (n + 1) + n + n + 1 = 3n + 4 = \Theta(n).$$

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. L'input è il numero naturale  $n$ , quindi  $|I| = \Theta(\log_2 n)$ , e  $S(n) = 3n + 4 = \Theta(n)$ . Per la regola

(A) dell'introduzione agli esercizi esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{quindi} \quad T(|I|) = 2^{\Theta(|I|)}.$$

Il tempo di esecuzione nel caso peggiore di **Algorithm1**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = 2^{\Theta(|I|)}$ : è **esponenziale** in  $|I|$ , anche se il numero di passi è lineare nel valore  $n$ .

### Esercizio 12.2 — Tempo di esecuzione ( $E2$ )

Si consideri il seguente algoritmo, il cui input è un numero naturale  $n$ .

**Algorithm:** Algoritmo 2

**Input** : Un numero naturale  $n$

**Output:** Un numero naturale  $m$

```

1 function Algorithm2( $n$ )
2   if  $n \bmod 2 = 0$  then
3      $m := 0$ 
4      $i := 1$ 
5     while  $i \leq n$  do
6        $m \leftarrow m + 2$ 
7        $i \leftarrow i + 1$ 
8     end
9   else
10     $m := n$ 
11  end
12  return  $m$ 
13 end
```

#### Domande

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

#### Soluzione

1. L'algoritmo restituisce il doppio  $2n$  di  $n$  se  $n$  è pari. Se  $n$  è dispari, restituisce  $n$  invariato.

2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm2**, dove  $n$  è il numero naturale dato in input. L'esecuzione delle righe 3–7 dipende dallo specifico valore di input  $n$ : prima di eseguire l'algoritmo non sappiamo se  $n$  sia pari o dispari. Definiamo una funzione

$$t : \mathbb{N} \rightarrow \{0, 1\},$$

dove  $t(n) = 1$  se  $n \bmod 2 = 0$ , e  $t(n) = 0$  altrimenti. In altre parole,  $t(n)$  è un oracolo che ci dice se il numero in input  $n$  è pari. Il numero di volte in cui ciascuna riga viene eseguita è:

| riga | numero di esecuzioni |
|------|----------------------|
| 2    | 1                    |
| 3    | $t(n)$               |
| 4    | $t(n)$               |
| 5    | $t(n)(n+1)$          |
| 6    | $t(n)n$              |
| 7    | $t(n)n$              |
| 10   | $1 - t(n)$           |
| 12   | 1                    |

Quindi il numero totale di passi è

$$1 + t(n) + t(n) + t(n)(n+1) + t(n)n + t(n)n + (1 - t(n)) + 1 = 3nt(n) + 2t(n) + 3.$$

Il caso peggiore si ha quando  $n$  è pari: allora  $t(n) = 1$  e l'algoritmo esegue  $3n + 5$  passi, mentre per  $n$  dispari ne esegue solo 3. Ogni dimensione dell'input di almeno due bit è la dimensione di qualche numero pari, quindi nel caso peggiore

$$S(n) = 3n + 5 = \Theta(n).$$

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. L'input è il numero naturale  $n$ , quindi  $|I| = \Theta(\log_2 n)$ , e  $S(n) = 3n + 5 = \Theta(n)$ . Per la regola (A) dell'introduzione agli esercizi esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{quindi} \quad T(|I|) = 2^{\Theta(|I|)}.$$

Il tempo di esecuzione nel caso peggiore di **Algorithm2**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = 2^{\Theta(|I|)}$ : è **esponenziale** in  $|I|$ , anche se il numero di passi è lineare nel valore  $n$ .

### Esercizio 12.3 — Tempo di esecuzione ( $E3$ )

Si consideri il seguente algoritmo, il cui input è un numero naturale  $n$ .

**Algorithm:** Algoritmo 3**Input** : Un numero naturale  $n$ **Output:** Un numero naturale  $s$ 

```

1 function Algorithm3( $n$ )
2    $s := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $s \leftarrow s + i$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $s$ 
9 end

```

**Domande**

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

**Soluzione**

1. L'algoritmo determina la somma dei primi  $n$  numeri naturali positivi:

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm3**, dove  $n$  è il numero naturale dato in input (per ogni valore di  $n$  c'è un solo input). Il corpo del ciclo viene eseguito  $n$  volte e la sua condizione  $n + 1$  volte:

| riga | numero di esecuzioni |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $n$                  |
| 8    | 1                    |

Quindi il numero totale di passi è

$$S(n) = 1 + 1 + (n + 1) + n + n + 1 = 3n + 4 = \Theta(n).$$

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. L'input è il numero naturale  $n$ , quindi  $|I| = \Theta(\log_2 n)$ , e  $S(n) = 3n + 4 = \Theta(n)$ . Per la regola (A) dell'introduzione agli esercizi esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{quindi} \quad T(|I|) = 2^{\Theta(|I|)}.$$

Il tempo di esecuzione nel caso peggiore di **Algorithm3**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = 2^{\Theta(|I|)}$ : è **esponenziale** in  $|I|$ , anche se il numero di passi è lineare nel valore  $n$ .

### Esercizio 12.4 — Tempo di esecuzione (E4)

Si consideri il seguente algoritmo, il cui input è un numero naturale  $n$ .

**Algorithm:** Algoritmo 4

**Input** : Un numero naturale  $n$

**Output:** Un numero naturale  $p$

```

1 function Algorithm4( $n$ )
2    $p := 1$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $p \leftarrow p \cdot i$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $p$ 
9 end
```

### Domande

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

### Soluzione

1. L'algoritmo determina il prodotto

$$1 \cdot 2 \cdot \dots \cdot n = n!$$

(con  $0! = 1$ , restituito per  $n = 0$ ).

2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm4**, dove  $n$  è il numero naturale dato in input (per ogni valore di  $n$  c'è un solo input). Il corpo del ciclo viene eseguito  $n$  volte e la sua condizione  $n + 1$  volte:

| riga | numero di esecuzioni |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $n$                  |
| 8    | 1                    |

Quindi il numero totale di passi è

$$S(n) = 1 + 1 + (n + 1) + n + n + 1 = 3n + 4 = \Theta(n).$$

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. L'input è il numero naturale  $n$ , quindi  $|I| = \Theta(\log_2 n)$ , e  $S(n) = 3n + 4 = \Theta(n)$ . Per la regola (A) dell'introduzione agli esercizi esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{quindi} \quad T(|I|) = 2^{\Theta(|I|)}.$$

Il tempo di esecuzione nel caso peggiore di **Algorithm4**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = 2^{\Theta(|I|)}$ : è **esponenziale** in  $|I|$ , anche se il numero di passi è lineare nel valore  $n$ .

### Esercizio 12.5 — Tempo di esecuzione (E5)

Si consideri il seguente algoritmo, il cui input è un numero naturale  $n$ .

**Algorithm:** Algoritmo 5**Input** : Un numero naturale  $n$ **Output:** Un numero naturale  $c$ 

```

1 function Algorithm5( $n$ )
2    $c := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5     if  $i \bmod 2 = 0$  then
6        $c \leftarrow c + 1$ 
7     end
8      $i \leftarrow i + 1$ 
9   end
10  return  $c$ 
11 end

```

**Domande**

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

**Soluzione**

1. L'algoritmo determina quanti numeri naturali tra 1 e  $n$  sono pari, cioè  $\lfloor n/2 \rfloor$ .
2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm5**, dove  $n$  è il numero naturale dato in input. La riga 6 viene eseguita solo quando  $i$  è pari, cioè  $\lfloor n/2 \rfloor$  volte. Il numero di volte in cui ciascuna riga viene eseguita è:

| riga | numero di esecuzioni  |
|------|-----------------------|
| 2    | 1                     |
| 3    | 1                     |
| 4    | $n + 1$               |
| 5    | $n$                   |
| 6    | $\lfloor n/2 \rfloor$ |
| 8    | $n$                   |
| 10   | 1                     |

Quindi il numero totale di passi è

$$S(n) = 1 + 1 + (n + 1) + n + \lfloor n/2 \rfloor + n + 1 = 3n + \lfloor n/2 \rfloor + 4.$$

Poiché  $\frac{n-1}{2} \leq \lfloor n/2 \rfloor \leq \frac{n}{2}$ , si ha  $\frac{7}{2}n + \frac{7}{2} \leq S(n) \leq \frac{7}{2}n + 4$ , quindi  $S(n) = \Theta(n)$ .

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. L'input è il numero naturale  $n$ , quindi  $|I| = \Theta(\log_2 n)$ , e  $S(n) = 3n + \lfloor n/2 \rfloor + 4 = \Theta(n)$ . Per la regola (A) dell'introduzione agli esercizi esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{quindi} \quad T(|I|) = 2^{\Theta(|I|)}.$$

Il tempo di esecuzione nel caso peggiore di **Algorithm5**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = 2^{\Theta(|I|)}$ : è **esponenziale** in  $|I|$ , anche se il numero di passi è lineare nel valore  $n$ .

### Esercizio 12.6 — Tempo di esecuzione (E6)

Si consideri il seguente algoritmo, il cui input è un numero naturale  $n$ .

**Algorithm:** Algoritmo 6

**Input :** Un numero naturale  $n$

**Output:** Un numero naturale  $q$

```

1 function Algorithm6( $n$ )
2    $q := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $q \leftarrow q + n$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $q$ 
9 end
```

### Domande

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

### Soluzione

1. L'algoritmo somma  $n$  a  $q$  esattamente  $n$  volte, partendo da  $q = 0$ : determina  $n \cdot n = n^2$ .
2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm6**, dove  $n$  è il numero naturale dato in input (per ogni valore di  $n$  c'è un solo input). Il corpo del ciclo viene eseguito  $n$  volte e la sua condizione  $n + 1$  volte:

| riga | numero di esecuzioni |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $n$                  |
| 8    | 1                    |

Quindi il numero totale di passi è

$$S(n) = 1 + 1 + (n + 1) + n + n + 1 = 3n + 4 = \Theta(n).$$

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. L'input è il numero naturale  $n$ , quindi  $|I| = \Theta(\log_2 n)$ , e  $S(n) = 3n + 4 = \Theta(n)$ . Per la regola (A) dell'introduzione agli esercizi esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{quindi} \quad T(|I|) = 2^{\Theta(|I|)}.$$

Il tempo di esecuzione nel caso peggiore di **Algorithm6**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = 2^{\Theta(|I|)}$ : è **esponenziale** in  $|I|$ , anche se il numero di passi è lineare nel valore  $n$ .

### Esercizio 12.7 — Tempo di esecuzione (*E7*)

Si consideri il seguente algoritmo, il cui input è un array  $a[1..n]$  di  $n$  numeri interi.

**Algorithm:** Algoritmo 7

**Input :** Un array  $a[1..n]$  di  $n$  numeri interi

**Output:** Un numero intero  $s$

```

1 function Algorithm7( $a, n$ )
2    $s := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5     if  $a[i] > 0$  then
6        $s \leftarrow s + a[i]$ 
7     end
8      $i \leftarrow i + 1$ 
9   end
10  return  $s$ 
11 end
```

**Domande**

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

**Soluzione**

1. L'algoritmo determina la somma dei valori positivi nell'array  $a[1..n]$ .
2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm7** nel caso peggiore, dove  $n$  è la dimensione dell'array. L'esecuzione della riga 6 dipende dall'array in input. Definiamo una funzione

$$t : \{1, 2, \dots, n\} \rightarrow \{0, 1\},$$

dove  $t(i) = 1$  se  $a[i] > 0$ , e  $t(i) = 0$  altrimenti. Il numero di volte in cui ciascuna riga viene eseguita è:

| riga | numero di esecuzioni |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $\sum_{i=1}^n t(i)$  |
| 8    | $n$                  |
| 10   | 1                    |

Quindi il numero totale di passi è

$$1 + 1 + (n + 1) + n + \sum_{i=1}^n t(i) + n + 1 = 3n + 4 + \sum_{i=1}^n t(i).$$

Il caso peggiore si ha quando tutti gli elementi sono positivi. Allora  $t(i) = 1$  per ogni  $i \in \{1, 2, \dots, n\}$ , e

$$S(n) = 3n + 4 + \sum_{i=1}^n 1 = 4n + 4 = \Theta(n).$$

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. Se tutti i valori in input usano un numero costante fisso di bit, allora  $|I| = \Theta(n)$ , e  $S(n) = \Theta(n)$ . Per la regola (B) dell'introduzione agli esercizi (con  $k = 1$ ) esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$\frac{a_1}{c_2} |I| \leq S(n) \leq \frac{a_2}{c_1} |I|, \quad \text{quindi} \quad T(|I|) = \Theta(|I|).$$

Il tempo di esecuzione nel caso peggiore di **Algorithm7**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = \Theta(|I|)$ : è **lineare** in  $|I|$ .

Esercizio 12.8 — Tempo di esecuzione ( $E8$ )

Si consideri il seguente algoritmo, il cui input è un array  $a[1..n]$  di  $n$  numeri interi.

**Algorithm:** Algoritmo 8

**Input :** Un array  $a[1..n]$  di  $n$  numeri interi

**Output:** Un numero naturale  $c$

```

1 function Algorithm8( $a, n$ )
2    $c := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5     if  $a[i] \bmod 2 = 0$  then
6        $c \leftarrow c + 1$ 
7     end
8      $i \leftarrow i + 1$ 
9   end
10  return  $c$ 
11 end

```

## Domande

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

## Soluzione

1. L'algoritmo determina il numero di valori pari nell'array  $a[1..n]$ .
2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm8** nel caso peggiore, dove  $n$  è la dimensione dell'array. L'esecuzione della riga 6 dipende dall'array in input. Definiamo una funzione

$$t : \{1, 2, \dots, n\} \rightarrow \{0, 1\},$$

dove  $t(i) = 1$  se  $a[i] \bmod 2 = 0$ , e  $t(i) = 0$  altrimenti. Il numero di volte in cui ciascuna riga viene eseguita è:

| riga | numero di esecuzioni |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $\sum_{i=1}^n t(i)$  |
| 8    | $n$                  |
| 10   | 1                    |

Quindi il numero totale di passi è

$$1 + 1 + (n + 1) + n + \sum_{i=1}^n t(i) + n + 1 = 3n + 4 + \sum_{i=1}^n t(i).$$

Il caso peggiore si ha quando tutti gli elementi sono pari. Allora  $t(i) = 1$  per ogni  $i \in \{1, 2, \dots, n\}$ , e

$$S(n) = 3n + 4 + \sum_{i=1}^n 1 = 4n + 4 = \Theta(n).$$

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. Se tutti i valori in input usano un numero costante fisso di bit, allora  $|I| = \Theta(n)$ , e  $S(n) = \Theta(n)$ . Per la regola (B) dell'introduzione agli esercizi (con  $k = 1$ ) esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$\frac{a_1}{c_2} |I| \leq S(n) \leq \frac{a_2}{c_1} |I|, \quad \text{quindi} \quad T(|I|) = \Theta(|I|).$$

Il tempo di esecuzione nel caso peggiore di **Algorithm8**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = \Theta(|I|)$ : è **lineare** in  $|I|$ .

### Esercizio 12.9 — Tempo di esecuzione (E9)

Si consideri il seguente algoritmo, il cui input è un array  $a[1..n]$  di  $n \geq 1$  numeri interi.

**Algorithm:** Algoritmo 9**Input** : Un array  $a[1..n]$  di  $n \geq 1$  numeri interi**Output:** Un numero intero  $m$ 

```

1 function Algorithm9( $a, n$ )
2    $m := a[1]$ 
3    $i := 2$ 
4   while  $i \leq n$  do
5     if  $a[i] < m$  then
6        $m \leftarrow a[i]$ 
7     end
8      $i \leftarrow i + 1$ 
9   end
10  return  $m$ 
11 end

```

**Domande**

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

**Soluzione**

1. L'algoritmo determina il valore minimo nell'array  $a[1..n]$ .
2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm9** nel caso peggiore, dove  $n$  è la dimensione dell'array. Il corpo del ciclo viene eseguito per  $i = 2, 3, \dots, n$ , cioè  $n - 1$  volte. Definiamo

$$t : \{2, 3, \dots, n\} \rightarrow \{0, 1\},$$

dove  $t(i) = 1$  se la riga 6 viene eseguita all'iterazione  $i$ , e  $t(i) = 0$  altrimenti. Il numero di volte in cui ciascuna riga viene eseguita è:

| riga | numero di esecuzioni |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n$                  |
| 5    | $n - 1$              |
| 6    | $\sum_{i=2}^n t(i)$  |
| 8    | $n - 1$              |
| 10   | 1                    |

Quindi il numero totale di passi è

$$1 + 1 + n + (n - 1) + \sum_{i=2}^n t(i) + (n - 1) + 1 = 3n + 1 + \sum_{i=2}^n t(i).$$

Il caso peggiore si ha quando

$$a[1] > a[2] > \dots > a[n].$$

Allora  $t(i) = 1$  per ogni  $i \in \{2, 3, \dots, n\}$ , e

$$S(n) = 3n + 1 + \sum_{i=2}^n 1 = 3n + 1 + (n - 1) = 4n = \Theta(n).$$

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. Se tutti i valori in input usano un numero costante fisso di bit, allora  $|I| = \Theta(n)$ , e  $S(n) = \Theta(n)$ . Per la regola (B) dell'introduzione agli esercizi (con  $k = 1$ ) esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$\frac{a_1}{c_2} |I| \leq S(n) \leq \frac{a_2}{c_1} |I|, \quad \text{quindi} \quad T(|I|) = \Theta(|I|).$$

Il tempo di esecuzione nel caso peggiore di **Algorithm9**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = \Theta(|I|)$ : è **lineare** in  $|I|$ .

### Esercizio 12.10 — Tempo di esecuzione (E10)

Si consideri il seguente algoritmo, il cui input è un array  $a[1..n]$  di  $n \geq 1$  numeri interi.

**Algorithm:** Algoritmo 10

**Input :** Un array  $a[1..n]$  di  $n \geq 1$  numeri interi

**Output:** true or false

```

1 function Algorithm10(a, n)
2   i := 1
3   while i < n do
4     if a[i] = a[i + 1] then
5       return true
6     end
7     i ← i + 1
8   end
9   return false
10 end
```

Domande

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

### Soluzione

1. L'algoritmo determina se l'array contiene due valori consecutivi uguali: restituisce **true** se  $a[i] = a[i + 1]$  per qualche  $i \in \{1, 2, \dots, n - 1\}$ , e **false** altrimenti.
2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm10** nel caso peggiore, dove  $n$  è la dimensione dell'array. Se la prima coppia di valori consecutivi uguali viene trovata all'iterazione  $k$ , l'algoritmo si ferma alla riga 5 dopo  $1 + k + k + (k - 1) + 1 = 3k + 1 \leq 3(n - 1) + 1$  passi. Il caso peggiore è quindi quello in cui non ci sono due valori consecutivi uguali: la riga 5 non viene mai eseguita e l'algoritmo controlla tutte le  $n - 1$  coppie consecutive. In questo caso il numero di volte in cui ciascuna riga viene eseguita è:

| riga | numero di esecuzioni |
|------|----------------------|
| 2    | 1                    |
| 3    | $n$                  |
| 4    | $n - 1$              |
| 5    | 0                    |
| 7    | $n - 1$              |
| 9    | 1                    |

Quindi il numero totale di passi è

$$S(n) = 1 + n + (n - 1) + (n - 1) + 1 = 3n = \Theta(n).$$

(e  $3n > 3(n - 1) + 1$ , quindi questo è effettivamente il caso peggiore).

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. Se tutti i valori in input usano un numero costante fisso di bit, allora  $|I| = \Theta(n)$ , e  $S(n) = \Theta(n)$ . Per la regola (B) dell'introduzione agli esercizi (con  $k = 1$ ) esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$\frac{a_1}{c_2} |I| \leq S(n) \leq \frac{a_2}{c_1} |I|, \quad \text{quindi} \quad T(|I|) = \Theta(|I|).$$

Il tempo di esecuzione nel caso peggiore di **Algorithm10**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = \Theta(|I|)$ : è **lineare** in  $|I|$ .

## Esercizio 12.11 — Tempo di esecuzione (E11)

Si consideri il seguente algoritmo, il cui input è un array  $a[1..n]$  di  $n \geq 1$  numeri interi.

**Algorithm:** Algoritmo 11

**Input :** Un array  $a[1..n]$  di  $n \geq 1$  numeri interi

**Output:** Un numero naturale  $c$

```

1 function Algorithm11( $a, n$ )
2    $c := 0$ 
3    $i := 1$ 
4   while  $i \leq n - 1$  do
5      $j := i + 1$ 
6     while  $j \leq n$  do
7       if  $a[i] < a[j]$  then
8          $c \leftarrow c + 1$ 
9       end
10       $j \leftarrow j + 1$ 
11    end
12     $i \leftarrow i + 1$ 
13  end
14  return  $c$ 
15 end

```

## Domande

1. Che cosa determina l'algoritmo?
2. Determinare il tempo di esecuzione nel caso peggiore dell'algoritmo.

## Soluzione

1. L'algoritmo determina il numero di coppie di posizioni  $(i, j)$ , con  $1 \leq i < j \leq n$ , tali che

$$a[i] < a[j].$$

2. Sia  $S(n)$  il numero di passi eseguiti da **Algorithm11** nel caso peggiore, dove  $n$  è la dimensione dell'array. Il ciclo esterno viene eseguito per  $i = 1, 2, \dots, n - 1$  (la sua condizione viene valutata  $n$  volte) e, per ogni  $i$ , il ciclo interno viene eseguito per  $j = i + 1, \dots, n$ , cioè  $n - i$  volte (la sua condizione viene valutata  $n - i + 1$  volte). Quindi il corpo del ciclo interno viene eseguito una volta per ogni coppia  $(i, j)$  con

$1 \leq i < j \leq n$ , e il numero di tali coppie è

$$P = (n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}.$$

Definiamo  $t(i, j) = 1$  se la riga 8 viene eseguita per la coppia  $(i, j)$ , e  $t(i, j) = 0$  altrimenti. Il numero di volte in cui ciascuna riga viene eseguita è:

| riga | numero di esecuzioni                   |
|------|----------------------------------------|
| 2    | 1                                      |
| 3    | 1                                      |
| 4    | $n$                                    |
| 5    | $n-1$                                  |
| 6    | $\sum_{i=1}^{n-1} (n-i+1) = P + (n-1)$ |
| 7    | $P$                                    |
| 8    | $\sum_{1 \leq i < j \leq n} t(i, j)$   |
| 10   | $P$                                    |
| 12   | $n-1$                                  |
| 14   | 1                                      |

Quindi il numero totale di passi è

$$\begin{aligned} & 1 + 1 + n + (n-1) + (P + (n-1)) + P + \sum_{1 \leq i < j \leq n} t(i, j) + P + (n-1) + 1 \\ &= 4n + 3P + \sum_{1 \leq i < j \leq n} t(i, j). \end{aligned}$$

Il caso peggiore per la riga 8 si ha quando

$$a[1] < a[2] < \cdots < a[n] :$$

allora  $t(i, j) = 1$  per ogni coppia, e

$$S(n) = 4n + 4P = 4n + 2n(n-1) = 2n^2 + 2n = \Theta(n^2).$$

Ad esempio, per  $n = 4$  e  $a[1..4] = (1, 2, 3, 4)$  l'algoritmo restituisce  $c = 6$  dopo  $S(4) = 2 \cdot 16 + 8 = 40$  passi.

Esprimiamo ora il tempo di esecuzione in funzione della dimensione dell'input. Se tutti i valori in input usano un numero costante fisso di bit, allora  $|I| = \Theta(n)$ , e  $S(n) = \Theta(n^2)$ . Per la regola (B) dell'introduzione agli esercizi (con  $k = 2$ ) esistono costanti  $a_1, a_2, c_1, c_2 > 0$  tali che, per ogni  $n$  sufficientemente grande,

$$\frac{a_1}{c_2^2} |I|^2 \leq S(n) \leq \frac{a_2}{c_1^2} |I|^2, \quad \text{quindi} \quad T(|I|) = \Theta(|I|^2).$$

Il tempo di esecuzione nel caso peggiore di **Algorithm11**, in funzione della dimensione dell'input  $|I|$ , è  $T(|I|) = \Theta(|I|^2)$ : è **quadratico** in  $|I|$ .

---

## Correttezza

---

Negli esercizi di questo capitolo la correttezza di un ciclo si dimostra con un invariante di ciclo, come nel capitolo «Correttezza», seguendo lo schema seguente.

### Attenzione

Un invariante di ciclo è una proprietà delle variabili che è vera ogni volta che viene valutata la condizione del ciclo. Per dimostrare la correttezza del ciclo:

1. **Inizializzazione:** l'invariante è vero prima della prima iterazione;
2. **Conservazione:** se l'invariante è vero prima di un'iterazione, è vero prima della successiva;
3. **Terminazione:** il ciclo termina dopo un numero finito di iterazioni (un numero naturale diminuisce strettamente a ogni iterazione), e l'invariante, insieme alla negazione della condizione del ciclo, fornisce l'output corretto.

Un sotto-array  $a[l..r]$  con  $l > r$  è vuoto, e una somma vuota è uguale a 0. Per mostrare che un algoritmo è **sbagliato** basta un controesempio.

### Esercizio 13.1 — Correttezza (E1)

Consideriamo il seguente algoritmo, che calcola il valore minimo di un array  $a[1..n]$  di  $n \geq 1$  numeri naturali.

**Algorithm:** FindMinimum**Input** : Un array  $a[1..n]$  di  $n$  numeri naturali**Output**: Il valore minimo dell'array  $a$ 

```

1 function FindMinimum( $a, n$ )
2    $min := a[1]$ 
3    $i := 2$ 
4   while  $i \leq n$  do
5     if  $a[i] < min$  then
6        $min \leftarrow a[i]$ 
7     end
8      $i \leftarrow i + 1$ 
9   end
10  return  $min$ 
11 end

```

**Domande**

1. Enunciare un invariante di ciclo per il ciclo while di FindMinimum.
2. Verificare l'invariante di ciclo sull'array  $a = [7, 3, 9, 2, 5, 2]$ , mostrando i valori di  $i$  e  $min$  ogni volta che viene valutata la condizione alla riga 4.
3. Dimostrare la correttezza di FindMinimum (inizializzazione, conservazione, terminazione).
4. Uno studente sostituisce la riga 2 con  $min := 0$ . Mostrare, con un controesempio, che il nuovo algoritmo è sbagliato, e indicare quale proprietà dell'invariante di ciclo non vale.

**Soluzione**

1. Invariante di ciclo:

Ogni volta che viene valutata la condizione alla riga 4,

$$min = \min\{a[1], a[2], \dots, a[i-1]\}.$$

2. For  $a = [7, 3, 9, 2, 5, 2]$  ( $n = 6$ ):

| $i$ | $min$ | $a[1..i-1]$        | $\min\{a[1], \dots, a[i-1]\}$ |
|-----|-------|--------------------|-------------------------------|
| 2   | 7     | [7]                | 7                             |
| 3   | 3     | [7, 3]             | 3                             |
| 4   | 3     | [7, 3, 9]          | 3                             |
| 5   | 2     | [7, 3, 9, 2]       | 2                             |
| 6   | 2     | [7, 3, 9, 2, 5]    | 2                             |
| 7   | 2     | [7, 3, 9, 2, 5, 2] | 2 (il ciclo termina)          |

L'invariante vale a ogni valutazione, e l'algoritmo restituisce 2.

### 3. Dimostrazione di correttezza.

- **Inizializzazione.** Prima della prima iterazione  $min = a[1]$  e  $i = 2$ . Il sotto-array  $a[1..1]$  ha il solo elemento  $a[1]$ , il cui minimo è  $a[1] = min$ .
- **Conservazione.** Supponiamo  $min = \min\{a[1], \dots, a[i-1]\}$  e  $i \leq n$ . Se  $a[i] < min$ , la riga 6 pone  $min \leftarrow a[i]$ , che è minore di tutti gli elementi di  $a[1..i-1]$ ; altrimenti  $min \leq a[i]$  e  $min$  non cambia. In entrambi i casi  $min = \min\{a[1], \dots, a[i]\}$ . La riga 8 pone  $i \leftarrow i + 1$ , quindi l'invariante vale con il nuovo valore di  $i$ .
- **Terminazione.** La variabile  $i$  parte da  $2 \leq n+1$  e aumenta di 1 a ogni iterazione, quindi il ciclo termina dopo  $n - 1$  iterazioni, quando  $i = n + 1$ . Sostituendo  $i = n + 1$  nell'invariante:

$$min = \min\{a[1], \dots, a[n]\},$$

che è restituito alla riga 10. Quindi l'algoritmo è corretto.

4. Con  $min := 0$ , consideriamo  $a = [3, 5]$ . Poiché nessun elemento è minore di 0, la riga 6 non viene mai eseguita e l'algoritmo restituisce 0, mentre l'output corretto è 3. Quindi l'algoritmo è sbagliato.

La proprietà che non vale è l'**inizializzazione**: prima della prima iterazione  $i = 2$  e  $min = 0$ , mentre  $\min\{a[1]\} = a[1] = 3 \neq 0$ . La conservazione è ancora vera (se l'invariante valesse, continuerebbe a valere), ma poiché è falso all'inizio, l'argomento di terminazione non può essere applicato. In effetti, poiché le componenti sono numeri naturali, l'algoritmo restituisce sempre  $\min\{0, a[2], \dots, a[n]\} = 0$ , che è corretto solo se almeno una componente dell'array è uguale a 0.

### Esercizio 13.2 — Correttezza (E2)

Consideriamo il seguente algoritmo.

**Algorithm:** ArraySum**Input** : Un array  $a[1..n]$  di  $n \geq 1$  numeri naturali**Output**: Un numero naturale  $s$ 

```

1 function ArraySum( $a, n$ )
2    $s := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $s \leftarrow s + a[i]$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $s$ 
9 end

```

**Domande**

1. Che cosa calcola l'algoritmo? Enunciare un invariante di ciclo per il suo ciclo while.
2. Verificare l'invariante di ciclo sull'array  $a = [4, 1, 7, 3]$ , mostrando i valori di  $i$  e  $s$  ogni volta che viene valutata la condizione alla riga 4.
3. Dimostrare la correttezza di ArraySum.
4. Uno studente modifica la riga 3 in  $i := 0$  e la riga 4 in "**while**  $i < n$ ", mantenendo  $a[i]$  alla riga 5. Il nuovo algoritmo è corretto?

**Soluzione**

1. L'algoritmo calcola la somma degli elementi dell'array:

$$s = \sum_{k=1}^n a[k].$$

Invariante di ciclo:

Ogni volta che viene valutata la condizione alla riga 4,

$$s = \sum_{k=1}^{i-1} a[k],$$

dove una somma vuota ( $i = 1$ ) è uguale a 0.

2. For  $a = [4, 1, 7, 3]$  ( $n = 4$ ):

| $i$ | $s$ | $\sum_{k=1}^{i-1} a[k]$                 |
|-----|-----|-----------------------------------------|
| 1   | 0   | 0 (somma vuota)                         |
| 2   | 4   | 4                                       |
| 3   | 5   | $4 + 1 = 5$                             |
| 4   | 12  | $4 + 1 + 7 = 12$                        |
| 5   | 15  | $4 + 1 + 7 + 3 = 15$ (il ciclo termina) |

L'algoritmo restituisce 15.

3. Dimostrazione di correttezza.

- **Inizializzazione.** Prima della prima iterazione  $s = 0$  e  $i = 1$ ; la somma  $\sum_{k=1}^0 a[k]$  è vuota, quindi uguale a  $0 = s$ .
- **Conservazione.** Supponiamo  $s = \sum_{k=1}^{i-1} a[k]$  e  $i \leq n$ . La riga 5 dà

$$s + a[i] = \sum_{k=1}^{i-1} a[k] + a[i] = \sum_{k=1}^i a[k],$$

e la riga 6 incrementa  $i$  di 1, quindi l'invariante vale con il nuovo valore di  $i$ .

- **Terminazione.** La variabile  $i$  parte da 1 e aumenta di 1 a ogni iterazione, quindi il ciclo termina dopo  $n$  iterazioni, quando  $i = n + 1$ . Sostituendo  $i = n + 1$  nell'invariante si ottiene  $s = \sum_{k=1}^n a[k]$ , che è restituito alla riga 8. Quindi l'algoritmo è corretto.
4. No. Con  $i := 0$  la prima iterazione accede ad  $a[0]$ , che non esiste in un array  $a[1..n]$ ; inoltre, la condizione  $i < n$  ferma il ciclo quando  $i = n$ , quindi  $a[n]$  non viene mai sommato. In termini di invariante, l'invariante naturale del nuovo ciclo è  $s = \sum_{k=0}^{i-1} a[k]$ , e alla terminazione ( $i = n$ ) dà  $s = \sum_{k=0}^{n-1} a[k] \neq \sum_{k=1}^n a[k]$ : il passo di terminazione non fornisce la proprietà richiesta. Per esempio, su  $a = [4, 1, 7, 3]$  l'elemento  $a[4] = 3$  non viene mai sommato, quindi anche ignorando l'accesso non valido il risultato non può essere 15. Questo è un errore tipico che si ottiene mescolando codice con indici da 0 (come in C++) e pseudocodice con indici da 1.

### Esercizio 13.3 — Correttezza (E3)

Consideriamo l'algoritmo `SelectionSort`, che ordina in modo non decrescente un array  $a[1..n]$  di  $n \geq 1$  numeri naturali.

**Algorithm:** SelectionSort**Input** : Un array  $a[1..n]$  di  $n$  numeri naturali**Output**: L'array  $a$  ordinato in modo non decrescente

```

1 function SelectionSort( $a, n$ )
2    $i := 1$ 
3   while  $i \leq n - 1$  do
4      $minIdx := i$  // indice del minimo in  $a[i..n]$ 
5      $j := i + 1$ 
6     while  $j \leq n$  do
7       if  $a[j] < a[minIdx]$  then
8          $minIdx \leftarrow j$  // trovato un nuovo minimo
9       end
10       $j \leftarrow j + 1$ 
11    end
12    if  $minIdx \neq i$  then
13       $tmp \leftarrow a[i]$  // scambia  $a[i]$  e  $a[minIdx]$ 
14       $a[i] \leftarrow a[minIdx]$ 
15       $a[minIdx] \leftarrow tmp$ 
16    end
17     $i \leftarrow i + 1$ 
18  end
19 end

```

**Domande**

1. Enunciare un invariante di ciclo per il ciclo while interno (riga 6) e dimostrarlo.
2. Enunciare un invariante di ciclo per il ciclo while esterno (riga 3).
3. Verificare l'invariante del ciclo esterno sull'array  $a = [29, 10, 14, 37, 13]$ .
4. Dimostrare la correttezza di SelectionSort.

**Soluzione**

1. Consideriamo un'iterazione del ciclo esterno con un dato valore di  $i$ . Il ciclo interno non modifica l'array. Il suo invariante è:

Ogni volta che viene valutata la condizione alla riga 6,  $i \leq minIdx \leq j - 1$  e

$$a[minIdx] = \min\{a[i], a[i+1], \dots, a[j-1]\}.$$

- **Inizializzazione.** Le righe 4–5 pongono  $minIdx = i$  e  $j = i + 1$ : il sotto-array  $a[i..j-1] = a[i..i]$  ha il solo elemento  $a[i] = a[minIdx]$ .
- **Conservazione.** Supponiamo l'invariante e  $j \leq n$ . Se  $a[j] < a[minIdx]$ , allora  $a[j]$  è minore di tutti gli elementi di  $a[i..j-1]$  e la riga 8 pone  $minIdx \leftarrow j$ ; altrimenti  $a[minIdx] \leq a[j]$ . In entrambi i casi  $a[minIdx] = \min\{a[i], \dots, a[j]\}$  e

$i \leq \text{minIdx} \leq j$ . La riga 10 pone  $j \leftarrow j + 1$ , quindi l'invariante vale con il nuovo  $j$ .

- **Terminazione.**  $j$  aumenta di 1 a partire da  $i + 1 \leq n$ , quindi il ciclo si ferma quando  $j = n + 1$ . Allora  $a[\text{minIdx}] = \min\{a[i], \dots, a[n]\}$ :  $\text{minIdx}$  è l'indice di un elemento minimo di  $a[i..n]$ .

2. Invariante del ciclo esterno:

Ogni volta che viene valutata la condizione alla riga 3:

- (P1) l'array  $a[1..n]$  contiene gli elementi originali (è un riordinamento dell'input);
- (P2) il sotto-array  $a[1..i - 1]$  è ordinato in modo non decrescente;
- (P3) ogni elemento di  $a[1..i - 1]$  è  $\leq$  ogni elemento di  $a[i..n]$ .

3. Per  $a = [29, 10, 14, 37, 13]$  ( $n = 5$ ); il prefisso ordinato  $a[1..i - 1]$  è evidenziato in **blu**:

| $i$ | $a[1]$ | $a[2]$ | $a[3]$ | $a[4]$ | $a[5]$ | minimo di $a[i..n]$                   |
|-----|--------|--------|--------|--------|--------|---------------------------------------|
| 1   | 29     | 10     | 14     | 37     | 13     | $a[2] = 10$ : scambio di $a[1], a[2]$ |
| 2   | 10     | 29     | 14     | 37     | 13     | $a[5] = 13$ : scambio di $a[2], a[5]$ |
| 3   | 10     | 13     | 14     | 37     | 29     | $a[3] = 14$ : nessuno scambio         |
| 4   | 10     | 13     | 14     | 37     | 29     | $a[5] = 29$ : scambio di $a[4], a[5]$ |
| 5   | 10     | 13     | 14     | 29     | 37     | il ciclo termina ( $i = n$ )          |

A ogni riga il prefisso blu è ordinato, e ogni elemento blu è  $\leq$  ogni elemento non blu: per esempio, per  $i = 4$ ,  $\max\{10, 13, 14\} = 14 \leq \min\{37, 29\} = 29$ .

4. Dimostrazione di correttezza.

- **Inizializzazione.** Prima della prima iterazione  $i = 1$  e nulla è stato modificato: (P1) vale, e  $a[1..0]$  è vuoto, quindi (P2) e (P3) valgono banalmente.
- **Conservazione.** Supponiamo (P1)–(P3) e  $i \leq n - 1$ . Per il punto 1, dopo il ciclo interno

$$a[\text{minIdx}] = \min\{a[i], \dots, a[n]\}.$$

Le righe 12–15 scambiano  $a[i]$  e  $a[\text{minIdx}]$  (se sono posizioni diverse). Uno scambio si limita a scambiare due elementi, quindi (P1) vale ancora. Ora  $a[i]$  è il minimo di  $a[i..n]$ , quindi:

- per (P3),  $a[i] \geq$  ogni elemento di  $a[1..i - 1]$ , e poiché  $a[1..i - 1]$  è ordinato (P2),  $a[1..i]$  è ordinato;
- $a[i] \leq$  ogni elemento di  $a[i + 1..n]$ , perché  $a[i]$  è il minimo di  $a[i..n]$  e lo scambio riordina solo i valori nelle posizioni  $i..n$ ; per lo stesso motivo, per (P3) ogni elemento di  $a[1..i - 1]$  è  $\leq$  ogni elemento di  $a[i + 1..n]$ . Quindi ogni elemento di  $a[1..i]$  è  $\leq$  ogni elemento di  $a[i + 1..n]$ .

La riga 17 pone  $i \leftarrow i + 1$ , quindi (P1)–(P3) valgono con il nuovo valore di  $i$ .

- **Terminazione.** Il ciclo interno termina sempre (punto 1), e  $i$  aumenta di 1 a partire da 1; il ciclo esterno si ferma quando  $i = n$  per la prima volta (dopo  $n - 1$  iterazioni; se  $n = 1$  non viene mai eseguito e  $i = 1 = n$ ). Sostituendo  $i = n$ :  $a[1..n - 1]$  è ordinato (P2) e ogni elemento di  $a[1..n - 1]$  è  $\leq a[n]$  (P3); quindi l'intero array  $a[1..n]$  è ordinato e, per (P1), contiene gli elementi originali. Quindi l'algoritmo è corretto.

Notiamo che il ciclo esterno richiede solo  $n - 1$  iterazioni: quando le prime  $n - 1$  posizioni sono corrette, l'ultimo elemento è automaticamente nella sua posizione corretta.

### Esercizio 13.4 — Correttezza (E4)

Consideriamo l'algoritmo **LinearSearch**, che determina se un valore  $m$  è uno degli elementi di un array  $a[1..n]$  di  $n$  numeri naturali (non necessariamente ordinato).

---

#### Algorithm: LinearSearch

---

**Input** : Un array  $a[1..n]$  di  $n$  numeri naturali e un valore  $m$  da cercare

**Output**: Indice di  $m$  in  $a$ , oppure  $-1$  se non trovato

---

```

1 function LinearSearch( $a, n, m$ )
2    $i := 1$ 
3   while  $i \leq n$  do
4     if  $a[i] = m$  then
5       return  $i$                                 // Elemento trovato all'indice  $i$ 
6     end
7      $i \leftarrow i + 1$ 
8   end
9   return  $-1$                                     // Elemento non trovato
10 end
```

---

#### Domande

1. Enunciare un invariante di ciclo per il ciclo while di **LinearSearch**.
2. Verificare l'invariante di ciclo sull'array  $a = [8, 3, 6, 3, 9]$  per  $m = 6$  e per  $m = 5$ .
3. Dimostrare che **LinearSearch** è corretto. Dimostrare anche che, quando  $m$  è nell'array, l'indice restituito è il **più piccolo** indice  $k$  tale che  $a[k] = m$ .
4. Uno studente scambia le righe 4–5 e la riga 7, cioè il corpo del ciclo prima incrementa  $i$  e poi controlla se  $a[i] = m$ . Trovare un controesempio che mostri che il nuovo algoritmo è sbagliato, e indicare quale proprietà dell'invariante di ciclo non vale.

## Soluzione

1. Invariante di ciclo:

Ogni volta che viene valutata la condizione alla riga 3,  $1 \leq i \leq n + 1$  e  $m$  **non** è nel sotto-array  $a[1..i - 1]$ :

$$a[k] \neq m \quad \text{per ogni } k \in \{1, 2, \dots, i - 1\}.$$

2. Per  $a = [8, 3, 6, 3, 9]$ :

| $m$ | $i$ | $a[1..i - 1]$   | azione                                   |
|-----|-----|-----------------|------------------------------------------|
| 6   | 1   | [ ]             | $a[1] = 8 \neq 6$                        |
|     | 2   | [8]             | $a[2] = 3 \neq 6$                        |
|     | 3   | [8, 3]          | $a[3] = 6 = m \Rightarrow$ restituisce 3 |
| 5   | 1   | [ ]             | $a[1] = 8 \neq 5$                        |
|     | 2   | [8]             | $a[2] = 3 \neq 5$                        |
|     | 3   | [8, 3]          | $a[3] = 6 \neq 5$                        |
|     | 4   | [8, 3, 6]       | $a[4] = 3 \neq 5$                        |
|     | 5   | [8, 3, 6, 3]    | $a[5] = 9 \neq 5$                        |
|     | 6   | [8, 3, 6, 3, 9] | $i > n \Rightarrow$ restituisce $-1$     |

A ogni valutazione,  $m$  non compare in  $a[1..i - 1]$ .

3. Dimostrazione di correttezza.

- **Inizializzazione.** Prima della prima iterazione  $i = 1$  e  $a[1..0]$  è vuoto, quindi non contiene  $m$ .
- **Conservazione.** Supponiamo l'invariante e  $i \leq n$ . Se  $a[i] = m$ , l'algoritmo restituisce  $i$  (riga 5), che è un output corretto. Altrimenti  $a[i] \neq m$ , quindi  $m$  non è in  $a[1..i]$ ; la riga 7 pone  $i \leftarrow i + 1$  e l'invariante vale con il nuovo valore di  $i$ .
- **Terminazione.**  $i$  aumenta di 1 a partire da 1, quindi il ciclo esegue al più  $n$  iterazioni. L'algoritmo finisce in uno di due modi:
  - restituisce  $i$  con  $a[i] = m$ : corretto. Inoltre, per l'invariante  $a[k] \neq m$  per ogni  $k < i$ , quindi  $i$  è il più piccolo indice con  $a[i] = m$ ;
  - la condizione alla riga 3 è falsa, cioè  $i = n + 1$ : per l'invariante  $m$  non è in  $a[1..n]$ , e l'algoritmo restituisce correttamente  $-1$ .

Quindi l'algoritmo è corretto.

4. Il nuovo corpo è:  $i \leftarrow i + 1$ ; **if**  $a[i] = m$  **then return**  $i$ . Consideriamo  $a = [6, 3]$  e  $m = 6$ . Nella prima iterazione  $i$  diventa 2 e  $a[2] = 3 \neq 6$ ; nella seconda iterazione  $i$  diventa 3 e l'algoritmo accede ad  $a[3]$ , che è fuori dall'array; in ogni caso  $a[1] = 6$  non viene **mai** controllato, quindi l'algoritmo non può restituire 1.

La proprietà che non vale è la **conservazione**: se  $m$  non è in  $a[1..i - 1]$  all'inizio dell'iterazione, dopo il corpo sappiamo solo che  $m$  non è in  $a[1..i - 1]$  e che  $a[i + 1] \neq m$ ,

ma non sappiamo nulla su  $a[i]$ . Con il nuovo valore  $i + 1$ , l'invariante richiederebbe che  $m$  non sia in  $a[1..i]$ , cosa che non è garantita: nell'esempio, prima della seconda valutazione della condizione  $i = 2$  e  $a[1] = 6 = m$ .

### Esercizio 13.5 — Correttezza (E5)

Consideriamo i due algoritmi seguenti, che dovrebbero calcolare  $x^n$  per due numeri naturali  $x \geq 1$  e  $n \geq 0$  (ricordiamo che  $x^0 = 1$ ).

---

#### Algorithm: Power

---

**Input** : Due numeri naturali  $x \geq 1$  e  $n \geq 0$

**Output**:  $x^n$

```

1 function Power(x, n)
2   p := 1
3   i := 1
4   while i ≤ n do
5     p ← p · x
6     i ← i + 1
7   end
8   return p
9 end

```

---



---

#### Algorithm: WrongPower

---

**Input** : Due numeri naturali  $x \geq 1$  e  $n \geq 0$

**Output**:  $x^n$  (?)

```

1 function WrongPower(x, n)
2   p := x
3   i := 1
4   while i ≤ n do
5     p ← p · x
6     i ← i + 1
7   end
8   return p
9 end

```

---

#### Domande

1. Enunciare un invariante di ciclo per il ciclo while di **Power** e verificarlo per  $x = 3$ ,  $n = 4$ .
2. Dimostrare la correttezza di **Power**, incluso il caso  $n = 0$ .
3. Mostrare che lo stesso invariante **non** è un invariante di ciclo di **WrongPower**: quale delle tre proprietà non vale? Dare un controesempio che mostri che **WrongPower** è sbagliato, con una traccia per  $x = 2$ ,  $n = 3$ .

4. Trovare l'invariante di ciclo corretto di **WrongPower** e usarlo per determinare esattamente che cosa calcola **WrongPower**. Per quali input il suo output è corretto?

### Soluzione

1. Invariante di ciclo di **Power**:

Ogni volta che viene valutata la condizione alla riga 4,  $1 \leq i \leq n + 1$  e

$$p = x^{i-1}.$$

Per  $x = 3$ ,  $n = 4$ :

| $i$ | $p$ | $x^{i-1}$                     |
|-----|-----|-------------------------------|
| 1   | 1   | $3^0 = 1$                     |
| 2   | 3   | $3^1 = 3$                     |
| 3   | 9   | $3^2 = 9$                     |
| 4   | 27  | $3^3 = 27$                    |
| 5   | 81  | $3^4 = 81$ (il ciclo termina) |

L'algoritmo restituisce  $81 = 3^4$ .

2. Dimostrazione di correttezza di **Power**.

- **Inizializzazione.** Prima della prima iterazione  $p = 1$  e  $i = 1$ , quindi  $p = x^0 = x^{i-1}$ ; inoltre  $1 \leq i \leq n + 1$  poiché  $n \geq 0$ .
- **Conservazione.** Supponiamo  $p = x^{i-1}$  e  $i \leq n$ . La riga 5 dà  $p \cdot x = x^{i-1} \cdot x = x^i$ , e la riga 6 pone  $i \leftarrow i + 1 \leq n + 1$ . Con il nuovo valore di  $i$ ,  $p = x^{i-1}$ .
- **Terminazione.**  $i$  aumenta di 1 a partire da  $1 \leq n + 1$ , quindi il ciclo termina dopo esattamente  $n$  iterazioni, quando  $i = n + 1$ . Sostituendo  $i = n + 1$ :  $p = x^n$ , che è restituito alla riga 8.

Se  $n = 0$ , la condizione  $1 \leq 0$  è falsa alla prima valutazione, il ciclo non viene mai eseguito e l'algoritmo restituisce  $p = 1 = x^0$ , in accordo con la dimostrazione (inizializzazione e terminazione coincidono, con  $i = 1 = n + 1$ ).

3. Per **WrongPower**, l'**inizializzazione** non vale: prima della prima iterazione  $i = 1$  e  $p = x$ , mentre  $x^{i-1} = x^0 = 1$ . Quindi  $p \neq x^{i-1}$  ogni volta che  $x \geq 2$ . (La conservazione è ancora vera: se valesse  $p = x^{i-1}$ , il corpo la preserverebbe; ma l'"induzione" non ha un caso base valido.)

Controesempio:  $x = 2$ ,  $n = 3$ , per cui  $x^n = 8$ .

| $i$ | $p$ | $x^{i-1}$ | $x^i$                 |
|-----|-----|-----------|-----------------------|
| 1   | 2   | 1         | 2                     |
| 2   | 4   | 2         | 4                     |
| 3   | 8   | 4         | 8                     |
| 4   | 16  | 8         | 16 (il ciclo termina) |

WrongPower restituisce  $16 \neq 8 = 2^3$ , quindi è sbagliato.

4. La tabella suggerisce l'invariante  $p = x^i$ , che è un invariante di ciclo corretto di WrongPower:

- **Inizializzazione:**  $p = x = x^1 = x^i$  con  $i = 1$ ;
- **Conservazione:** se  $p = x^i$  e  $i \leq n$ , dopo il corpo  $p = x^{i+1}$  e  $i$  è incrementato di 1;
- **Terminazione:** il ciclo termina con  $i = n + 1$ , quindi  $p = x^{n+1}$ .

Quindi WrongPower calcola  $x^{n+1}$ . Il suo output è corretto, cioè  $x^{n+1} = x^n$ , se e solo se  $x^n(x-1) = 0$ , ovvero (poiché  $x \geq 1$ ) se e solo se  $x = 1$ . Per ogni  $x \geq 2$  e ogni  $n \geq 0$  è sbagliato: basta un controesempio per mostrare che un algoritmo non è corretto.