Approximation algorithms for bin packing

Outline

- Why bin packing algorithms?
- Notation
- The Next Fit
- The Any Fit class
  - The First Fit
  - The Best Fit
  - The Worst Fit
- Half-full property
- Performance guarantees
  - The worst case performance ratio of NF
  - The worst case performance ratio of WF
  - The worst case performance ratios of FF and BF
  - What the Any Fit condition does and does not give
- Typical mistakes
- C++ implementation
- In summary

Learning objectives

At the end of this chapter you will be able to:

- execute the Next Fit, First Fit, Best Fit and Worst Fit algorithms by hand, and write the item-bin assignment and the number of bins;
- decide whether an algorithm belongs to the Any Fit class, and whether a solution satisfies the half-full property;
- prove that Next Fit and every Any Fit algorithm have a relative performance guarantee of 2;
- describe the families of instances that make the guarantee 2 tight for Next Fit and Worst Fit, and state the worst-case ratio $\frac{17}{10}$ of First Fit and Best Fit;
- read and run the C++ implementations of the four algorithms.

Prerequisites

- From this module, the chapter «Principles of approximation algorithms»: performance ratio, relative performance guarantee, tight guarantee and worst-case performance ratio $r(\mathscr{A})$.
- From the module *Combinatorial Optimization Problems*, the chapter «Packing problems»: the BPP, its instances and the counting lower bound $\left\lceil \frac{\sum_{i \in \mathscr{I}} w_i}{c} \right\rceil$ on the optimal value.
- From the module *Heuristic Algorithms*, the chapter «Constructive heuristics»: heuristic algorithms that build a solution one item at a time.

The problem of this chapter

The problem of this chapter is the
bin packing problem (BPP):

its definition and its assumptions are on its page.

- We study four simple algorithms that pack the items one at a time: Next Fit, First Fit, Best Fit and Worst Fit
- For each of them we prove how far from the optimum it can be, in the worst case

Why bin packing algorithms?

Introduction: Why bin packing algorithms?

- We start from a small challenge: packing five items into as few bins as possible
- On a small instance we can find the optimum by hand, and prove it
- With many items we need simple rules, and a guarantee on how many bins they can waste

A challenge

Pack the items of weights $(6, 5, 4, 7, 3)$
into bins of capacity 10,
using as few bins as possible.

Can you prove that two bins are not enough?

The answer

- The total weight is $25 > 2 \cdot 10$, so at least

$$\left\lceil \frac{25}{10} \right\rceil = 3$$

bins are needed.

- Three bins are enough, for instance

$$\{6, 4\}, \qquad \{5, 3\}, \qquad \{7\}.$$

Simple rules with a guarantee

- With many items we cannot compare with the optimum, and we want simple rules that never use too many bins.
- The four algorithms of this chapter decide the bin of each item as it arrives.
- The chapter proves how far from the optimum they can be, in the worst case.

Summary

The challenge:

- the counting bound $\lceil \frac{25}{10} \rceil = 3$ shows that two bins are not enough
- three bins are enough: $\{6, 4\}$, $\{5, 3\}$, $\{7\}$

The goal of the chapter:

- simple rules that place each item as it arrives
- a proof of how far from the optimum they can be, in the worst case

Notation

Introduction: Notation

- In this section we fix the notation of the chapter
- We recall what a feasible solution of the BPP is
- We define the value $z(I)$ of an optimal solution and the value $\mathscr{A}(I)$ of the solution of an algorithm, which we compare in the whole chapter

Items, weights and bins

- As on the problem page, c is the bin capacity and $\mathscr{I} = \{1, 2, \ldots, n\}$ the set of items; in this chapter item $i \in \mathscr{I}$ has weight w_i.
- A **BPP feasible solution** is a partition of $\mathscr{I}$ into nonempty **packing patterns**, i.e., subsets of items each assigned to one bin, such that the total weight of every packing pattern does not exceed the bin capacity c.

Optimal value and value of an algorithm

- We denote by $z(I)$ the **optimal value** of the BPP on the instance I, i.e., the minimum number of bins.
- We denote by $\mathscr{A}(I)$ the number of bins used by the feasible solution returned by a heuristic algorithm $\mathscr{A}$.
- Since every heuristic algorithm of this chapter returns a feasible solution,

$$\mathscr{A}(I) \geq z(I) \qquad \text{and} \qquad \frac{\mathscr{A}(I)}{z(I)} \geq 1$$

for every BPP instance I.

Summary

Notation:

- capacity c, items $\mathscr{I} = \{1, 2, \ldots, n\}$, weights w_i
- a feasible solution is a partition of $\mathscr{I}$ into packing patterns of total weight at most c

Values:

- $z(I)$ is the optimal value, $\mathscr{A}(I)$ the number of bins used by algorithm $\mathscr{A}$
- $\mathscr{A}(I) \geq z(I)$, i.e., $\frac{\mathscr{A}(I)}{z(I)} \geq 1$

The Next Fit

Introduction: The Next Fit

- The Next Fit is the simplest algorithm of the chapter: it keeps only one bin open
- We describe the data structure of the solution, the pseudo-code and an execution by hand
- We also see how to read the packing pattern of each bin from the output of an algorithm: we use it in the whole chapter

The Next Fit algorithm

- The *Next Fit Algorithm* (denoted by NF) is a constructive heuristic for the BPP.
- It processes the items in the order in which they are given in the input instance, maintaining exactly one open bin at any time.
- For each item, NF packs it into the currently open bin if it fits; otherwise, it closes the bin, opens a new one, and places the item there.

Data structure for the solution

- We use a vector $bin = (bin_1, bin_2, \ldots, bin_n)$ of length n to store the solution.
- At position i, the vector contains the number of the bin to which item i is assigned. More formally:

$$bin_i = b \quad \Longleftrightarrow \quad \text{item } i \text{ is packed into bin } b.$$

  For example, if $bin_3 = 2$, then item 3 is assigned to bin 2.
- In addition, we maintain a variable nb that stores the total number of bins used, i.e., at the end of the algorithm $NF(I) = nb$.

Next: the pseudo-code of NF.

Algorithm: Next Fit (1/2)

Input : BPP instance I with n items, weights $w[1..n]$, capacity c

Output: Item-bin assignment $bin[1..n]$ and number of used bins nb

```
1 function NextFit(n, w, c)
2 |   bin := new array[1..n]    // item-bin assignment
3 |   nb := 1                          // current bin
4 |   rc := c                    // residual capacity
```

Algorithm: Next Fit (2/2)

function *NextFit*(n, w, c) (continued)

```
5   |   for i := 1 to n do
6   |   |   if w[i] > rc then
7   |   |   |   nb ← nb + 1                    // open new bin
8   |   |   |   rc ← c
9   |   |   end
10  |   |   bin[i] ← nb                  // assign item to bin
11  |   |   rc ← rc − w[i]
12  |   end
13  |   return bin, nb
14  end
```

Example: execution of NF (1/3)

We consider a bin of capacity $c = 10$ and a set $\mathscr{I} = \{1, 2, 3, 4, 5\}$ of 5 items ($n = 5$), with the following item weights:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
w_i	6	5	4	7	3

Initially $nb := 1$, $rc := c$. The NF algorithm produces the feasible solution of the next slides.

Example: execution of NF (2/3)

1. Item 1 (weight $w[1] = 6$): $w[1] = 6 \leq rc = 10$, so no new bin is opened. Item 1 is packed into bin $nb = 1$, and $rc \leftarrow 10 - 6 = 4$.

2. Item 2 (weight $w[2] = 5$): $w[2] = 5 > rc = 4$, so $nb \leftarrow 2$ and $rc \leftarrow c = 10$. Item 2 is packed into bin $nb = 2$, and $rc \leftarrow 10 - 5 = 5$.

3. Item 3 (weight $w[3] = 4$): $w[3] = 4 \leq rc = 5$, so no new bin is opened. Item 3 is packed into bin $nb = 2$, and $rc \leftarrow 5 - 4 = 1$.

Example: execution of NF (3/3)

4. Item 4 (weight $w[4] = 7$): $w[4] = 7 > rc = 1$, so $nb \leftarrow 3$ and $rc \leftarrow c = 10$. Item 4 is packed into bin $nb = 3$, and $rc \leftarrow 10 - 7 = 3$.

5. Item 5 (weight $w[5] = 3$): $w[5] = 3 \leq rc = 3$, so no new bin is opened. Item 5 is packed into bin $nb = 3$, and $rc \leftarrow 3 - 3 = 0$.

Hence NF returns:

$$bin = \begin{pmatrix} 1 & 2 & 2 & 3 & 3 \end{pmatrix}, \qquad nb = 3,$$

and therefore

$$NF(I) = nb = 3.$$

Packing patterns from the bin vector

- Given the output of an algorithm, namely the vector $bin = (bin_1, bin_2, \ldots, bin_n)$ and the number of bins nb, we can obtain the **packing pattern** P_b of each bin b, i.e., the set of items assigned to it, as follows:

$$P_b = \{\ i \in \{1, 2, \ldots, n\} : bin_i = b\ \},$$

  for $b \in \{1, 2, \ldots, nb\}$.

- In other words, P_b is the set of all items i such that $bin_i = b$; P_b is the packing pattern of bin b, not the bin itself.

Example: packing patterns from the bin vector (1/2)

Consider the instance of the previous example where NF returns:

$$bin = \begin{pmatrix} 1 & 2 & 2 & 3 & 3 \end{pmatrix}, \qquad nb = 3.$$

The packing patterns are:

$$P_1 = \{i \in \{1,2,3,4,5\} : bin_i = 1\} = \{1\},$$
$$P_2 = \{i \in \{1,2,3,4,5\} : bin_i = 2\} = \{2,3\},$$
$$P_3 = \{i \in \{1,2,3,4,5\} : bin_i = 3\} = \{4,5\}.$$

Example: packing patterns from the bin vector (2/2)

These are exactly the sets of items packed into the bins constructed by the algorithm:

- Bin 1 contains item 1 with weight $w_1 = 6$;
- Bin 2 contains items 2 and 3 with total weight $w_2 + w_3 = 5 + 4 = 9$;
- Bin 3 contains items 4 and 5 with total weight $w_4 + w_5 = 7 + 3 = 10$.

Summary

The Next Fit:

- processes the items in the given order, with exactly one open bin
- if the item does not fit, it closes the bin and opens a new one

The output:

- the vector bin, with $bin_i = b$ if item i is packed into bin b, and the number of bins nb
- the packing pattern of bin b is $P_b = \{i : bin_i = b\}$
- on the instance $w = (6, 5, 4, 7, 3)$, $c = 10$: $bin = (1, 2, 2, 3, 3)$ and $NF(I) = 3$

The Any Fit class

Introduction: The Any Fit class

- The Any Fit algorithms keep all the bins opened so far, and open a new bin only when the item fits nowhere
- We prove the key property of the class: the items of two bins never fit together
- Then we study three members of the class: First Fit, Best Fit and Worst Fit, with their pseudo-codes and an execution by hand

Any Fit algorithms

- A constructive heuristic for the BPP that keeps all the bins opened so far, and opens a new bin *only when* the current item fits into none of them, is called an *Any Fit Algorithm* (denoted by AF).
- Any Fit is therefore not a single algorithm but a *family* of algorithms.
- What the condition leaves free is the rule used to select one bin among those in which the item fits, and each rule gives a different member of the family.

Definition 1: Any Fit algorithm

A constructive heuristic $\mathscr{A}$ for the BPP is an ***Any Fit algorithm*** *if, denoting by $P_1, P_2, \ldots, P_{nb}$ the packing patterns of the bins opened so far, it opens a new bin for item i only when*

$$\sum_{h \in P_b} w_h + w_i > c \qquad \text{for every } b \in \{1, 2, \ldots, nb\},$$

whereas the choice of the bin, when more than one can accommodate the item, is left unspecified.

The items of two bins never fit together (1/2)

- Let b and b' be two bins opened by an Any Fit algorithm, with $b < b'$.
- Let f be the first item packed into bin b', and let W_b be the total weight of the items contained in bin b at the moment in which f was processed.
- Bin b' was opened because f did not fit into any of the bins opened so far, and in particular not into bin b, hence

$$W_b + w_f > c.$$

The items of two bins never fit together (2/2)

- The total weight of a bin never decreases during the execution, so $\sum_{i \in P_b} w_i \geq W_b$, and $f \in P_{b'}$ gives $\sum_{i \in P_{b'}} w_i \geq w_f$.
- Therefore

$$\sum_{i \in P_b} w_i + \sum_{i \in P_{b'}} w_i \;\geq\; W_b + w_f \;>\; c$$

  for any two distinct bins b and b': accordingly, no two bins of an Any Fit solution can be merged.
- Only the Any Fit condition is used here, the selection rule plays no role.

NF is not an Any Fit algorithm

- NF keeps a single open bin and closes it as soon as an item does not fit.
- It does so even when that item would fit into a bin closed earlier.

Next: an instance on which NF violates the Any Fit condition.

Example: NF is not an Any Fit algorithm (1/4)

We consider a bin of capacity $c = 10$ and a set $\mathscr{I} = \{1, 2, 3, 4\}$ of 4 items ($n = 4$), with the following item weights:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$
w_i	3	8	3	8

Initially $nb := 1$, $rc := c$. The NF algorithm produces the feasible solution of the next slides.

Example: NF is not an Any Fit algorithm (2/4)

1. Item 1 (weight $w[1] = 3$): $w[1] = 3 \leq rc = 10$, so no new bin is opened. Item 1 is packed into bin $nb = 1$, and $rc \leftarrow 10 - 3 = 7$.
2. Item 2 (weight $w[2] = 8$): $w[2] = 8 > rc = 7$, so $nb \leftarrow 2$ and $rc \leftarrow c = 10$. Item 2 is packed into bin $nb = 2$, and $rc \leftarrow 10 - 8 = 2$.
3. Item 3 (weight $w[3] = 3$): $w[3] = 3 > rc = 2$, so $nb \leftarrow 3$ and $rc \leftarrow c = 10$. Item 3 is packed into bin $nb = 3$, and $rc \leftarrow 10 - 3 = 7$.
4. Item 4 (weight $w[4] = 8$): $w[4] = 8 > rc = 7$, so $nb \leftarrow 4$ and $rc \leftarrow c = 10$. Item 4 is packed into bin $nb = 4$, and $rc \leftarrow 10 - 8 = 2$.

Example: NF is not an Any Fit algorithm (3/4)

Hence NF returns:

$$bin = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}, \qquad nb = 4.$$

At the third iteration item 3 would fit into bin 1, whose residual capacity is 7, and NF opens a new bin all the same: NF violates the Any Fit condition.

Example: NF is not an Any Fit algorithm (4/4)

- An Any Fit algorithm, on the contrary, cannot open a third bin for item 3, because the item fits into bin 1.
- It packs item 3 there and then opens a new bin for item 4, which fits in neither bin, using 3 bins instead of 4.
- This holds for every algorithm of the class, whatever the selection rule.

We now study three members of the Any Fit family.

They differ only in the rule used to select the bin:

the **first** bin, the **fullest** bin, the **emptiest** bin.

The First Fit algorithm

- The *First Fit Algorithm* (denoted by FF) is a constructive heuristic for the BPP.
- It processes the items in the order in which they are given in the input instance, maintaining all bins opened so far.
- For each item, FF scans the bins in the order they were opened and packs the item into the *first* bin in which it fits; if no current bin can accommodate the item, FF opens a new bin and places the item there.

First Fit and the Any Fit class

- FF is an Any Fit algorithm: it opens a new bin only when the item fits into none of the bins opened so far.
- The rule left unspecified by the definition of Any Fit algorithm is here resolved by taking the first bin in which the item fits.

Next: the pseudo-code of FF.

Algorithm: First Fit (1/3)

Input : BPP instance I with n items, weights $w[1..n]$, capacity c

Output: Item-bin assignment $bin[1..n]$ and number of used bins nb

```
1 function FirstFit(n, w, c)
2 |   bin := new array[1..n]     // item-bin assignment
3 |   rc := new array[1..n]      // residual capacities
4 |   nb := 1                          // number of bins
5 |   rc[1] ← c                 // initialize first bin
```

Algorithm: First Fit (2/3)

function *FirstFit*(n, w, c) (continued)

```
6     for i := 1 to n do
7         s := −1                                  // selected bin
8         for b := 1 to nb do
9             if w[i] ≤ rc[b] then
10                s ← b
11                break
12            end
13        end
```

Algorithm: First Fit (3/3)

function *FirstFit*(n, w, c) (continued)

```
    for i := 1 to n (continued) do
14      if s = −1 then
15          nb ← nb + 1                    // open new bin
16          rc[nb] ← c
17          s ← nb
18      end
19      bin[i] ← s                          // assign item
20      rc[s] ← rc[s] − w[i]
21  end
22  return bin, nb
23 end
```

Example: execution of FF (1/3)

We consider a bin of capacity $c = 10$ and a set $\mathscr{I} = \{1, 2, 3, 4, 5\}$ of 5 items ($n = 5$), with the following item weights:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
w_i	6	5	4	7	3

The FF algorithm produces the feasible solution of the next slides.

Example: execution of FF (2/3)

1. Item 1 (weight 6): bin 1 is open at the start ($nb = 1$, $rc[1] = c = 10$) and $w[1] \leq rc[1]$, so $s = 1$ and FF packs item 1 into bin 1, and $rc[1] \leftarrow 10 - 6 = 4$.

2. Item 2 (weight 5): $b = 1$: $w[2] = 5 \leq rc[1] = 4$? No. No bin fits ($s = -1$), so FF opens bin 2 ($rc[2] \leftarrow 10$). Item 2 is packed into bin 2, and $rc[2] \leftarrow 10 - 5 = 5$.

3. Item 3 (weight 4): $b = 1$: $w[3] = 4 \leq rc[1] = 4$? Yes, so $s \leftarrow 1$ and the scan stops. Item 3 is packed into bin 1, and $rc[1] \leftarrow 4 - 4 = 0$.

Example: execution of FF (3/3)

4. Item 4 (weight 7): $b = 1$: $w[4] = 7 \leq rc[1] = 0$? No. $b = 2$: $w[4] = 7 \leq rc[2] = 5$? No. No bin fits ($s = -1$), so FF opens bin 3 ($rc[3] \leftarrow 10$). Item 4 is packed into bin 3, and $rc[3] \leftarrow 10 - 7 = 3$.

5. Item 5 (weight 3): $b = 1$: $w[5] = 3 \leq rc[1] = 0$? No. $b = 2$: $w[5] = 3 \leq rc[2] = 5$? Yes, so $s \leftarrow 2$ and the scan stops. Item 5 is packed into bin 2, and $rc[2] \leftarrow 5 - 3 = 2$.

Hence FF returns:

$$bin = \begin{pmatrix} 1 & 2 & 1 & 3 & 2 \end{pmatrix}, \qquad nb = 3,$$

and therefore

$$FF(I) = nb = 3.$$

Example: packing patterns from the bin vector (1/2)

Consider the instance of the previous example where FF returns:

$$bin = \begin{pmatrix} 1 & 2 & 1 & 3 & 2 \end{pmatrix}, \qquad nb = 3.$$

The packing patterns are:

$$P_1 = \{i \in \{1,2,3,4,5\} : bin_i = 1\} = \{1,3\},$$
$$P_2 = \{i \in \{1,2,3,4,5\} : bin_i = 2\} = \{2,5\},$$
$$P_3 = \{i \in \{1,2,3,4,5\} : bin_i = 3\} = \{4\}.$$

Example: packing patterns from the bin vector (2/2)

These are exactly the sets of items packed into the bins constructed by the algorithm:

- Bin 1 contains items 1 and 3 with total weight $w_1 + w_3 = 6 + 4 = 10$;
- Bin 2 contains items 2 and 5 with total weight $w_2 + w_5 = 5 + 3 = 8$;
- Bin 3 contains item 4 with weight $w_4 = 7$.

The Best Fit algorithm

- The *Best Fit Algorithm* (denoted by BF) is a constructive heuristic for the BPP.
- It processes the items in the order in which they are given in the input instance, maintaining all bins opened so far.
- For each item, BF considers all the bins in which the item fits and packs it into the one with the *smallest residual capacity* (in case of ties, the bin opened first); if no bin can accommodate the item, BF opens a new bin and places the item there.

Why Best Fit

- The rationale is to fill the bins as tightly as possible: by using the bin where the item leaves the least space unused, BF keeps the bins with a larger residual capacity available for the larger items that may still arrive.

- BF is an Any Fit algorithm: it opens a new bin only when the item fits into none of the bins opened so far, and the rule left unspecified by the definition of Any Fit algorithm is here resolved by taking the bin with the smallest residual capacity.

Next: the pseudo-code of BF.

Algorithm: Best Fit (1/3)

Input : BPP instance I with n items, weights $w[1..n]$, capacity c

Output: Item-bin assignment $bin[1..n]$ and number of used bins nb

```
1 function BestFit(n, w, c)
2 |   bin := new array[1..n]     // item-bin assignment
3 |   rc := new array[1..n]      // residual capacities
4 |   nb := 1                          // number of bins
5 |   rc[1] ← c                 // initialize first bin
```

Algorithm: Best Fit (2/3)

function *BestFit*(n, w, c) (continued)

```
6   |   for i := 1 to n do
7   |   |   minrc := c + 1          // smallest residual
    |   |    capacity found so far
8   |   |   s := −1                 // selected bin
9   |   |   for b := 1 to nb do
10  |   |   |   if w[i] ≤ rc[b] and rc[b] < minrc then
11  |   |   |   |   s ← b
12  |   |   |   |   minrc ← rc[b]
13  |   |   |   end
14  |   |   end
```

Algorithm: Best Fit (3/3)

function *BestFit*(n, w, c) (continued)

```
    for i := 1 to n (continued) do
15      if s = −1 then
16          nb ← nb + 1                    // open new bin
17          rc[nb] ← c
18          s ← nb
19      end
20      bin[i] ← s                          // assign item
21      rc[s] ← rc[s] − w[i]
22  end
23  return bin, nb
24 end
```

Example: execution of BF (1/4)

We consider a bin of capacity $c = 10$ and a set $\mathscr{I} = \{1, 2, 3, 4, 5\}$ of 5 items ($n = 5$), with the following item weights:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
w_i	6	5	4	7	3

The BF algorithm produces the feasible solution of the next slides.

Example: execution of BF (2/4)

1. Item 1 (weight 6): bin 1 is open at the start ($nb = 1$, $rc[1] = c = 10$) and $w[1] \leq rc[1]$, so $s = 1$ and BF packs item 1 into bin 1, and $rc[1] \leftarrow 10 - 6 = 4$.

2. Item 2 (weight 5): $minrc := 11$, $s := -1$. $b = 1$: $w[2] = 5 \leq rc[1] = 4$? No. No bin fits ($s = -1$), so BF opens bin 2 ($rc[2] \leftarrow 10$). Item 2 is packed into bin 2, and $rc[2] \leftarrow 10 - 5 = 5$.

3. Item 3 (weight 4): $minrc := 11$, $s := -1$. $b = 1$: $w[3] = 4 \leq rc[1] = 4$ and $rc[1] = 4 < 11$, so $s \leftarrow 1$, $minrc \leftarrow 4$. $b = 2$: $w[3] = 4 \leq rc[2] = 5$ and $rc[2] = 5 < minrc = 4$? No. Hence $s = 1$: item 3 is packed into bin 1 (the one leaving the smallest residual capacity, 0, against 1 for bin 2), and $rc[1] \leftarrow 4 - 4 = 0$.

Example: execution of BF (3/4)

4. Item 4 (weight 7): $b = 1$: $w[4] = 7 \leq rc[1] = 0$? No. $b = 2$: $w[4] = 7 \leq rc[2] = 5$? No. No bin fits ($s = -1$), so BF opens bin 3 ($rc[3] \leftarrow 10$). Item 4 is packed into bin 3, and $rc[3] \leftarrow 10 - 7 = 3$.

5. Item 5 (weight 3): $minrc := 11$, $s := -1$. $b = 1$: $w[5] = 3 \leq rc[1] = 0$? No. $b = 2$: $w[5] = 3 \leq rc[2] = 5$ and $rc[2] = 5 < 11$, so $s \leftarrow 2$, $minrc \leftarrow 5$. $b = 3$: $w[5] = 3 \leq rc[3] = 3$ and $rc[3] = 3 < minrc = 5$, so $s \leftarrow 3$, $minrc \leftarrow 3$. Hence $s = 3$: item 5 is packed into bin 3 (the one leaving the smallest residual capacity, 0, against 2 for bin 2), and $rc[3] \leftarrow 3 - 3 = 0$.

Example: execution of BF (4/4)

Hence BF returns:

$$bin = \begin{pmatrix} 1 & 2 & 1 & 3 & 3 \end{pmatrix}, \qquad nb = 3,$$

and therefore

$$BF(I) = nb = 3.$$

Example: packing patterns from the bin vector (1/2)

Consider the instance of the previous example where BF returns:

$$bin = \begin{pmatrix} 1 & 2 & 1 & 3 & 3 \end{pmatrix}, \qquad nb = 3.$$

The packing patterns are:

$$P_1 = \{i \in \{1,2,3,4,5\} : bin_i = 1\} = \{1,3\},$$
$$P_2 = \{i \in \{1,2,3,4,5\} : bin_i = 2\} = \{2\},$$
$$P_3 = \{i \in \{1,2,3,4,5\} : bin_i = 3\} = \{4,5\}.$$

Example: packing patterns from the bin vector (2/2)

These are exactly the sets of items packed into the bins constructed by the algorithm:

- Bin 1 contains items 1 and 3 with total weight $w_1 + w_3 = 6 + 4 = 10$;
- Bin 2 contains item 2 with weight $w_2 = 5$;
- Bin 3 contains items 4 and 5 with total weight $w_4 + w_5 = 7 + 3 = 10$.

The Worst Fit algorithm

- The *Worst Fit Algorithm* (denoted by WF) is a constructive heuristic for the BPP.
- It processes the items in the order in which they are given in the input instance, maintaining all bins opened so far.
- For each item, WF considers all the bins in which the item fits and packs it into the one with the *largest residual capacity* (in case of ties, the bin opened first); if no bin can accommodate the item, WF opens a new bin and places the item there.

Why Worst Fit

- The rationale is the opposite of the one of BF: by packing each item into the emptiest bin, WF avoids creating bins whose residual capacity is so small that no future item can use it, and it keeps the free space spread over the open bins.
- WF is an Any Fit algorithm: it opens a new bin only when the item fits into none of the bins opened so far, and the rule left unspecified by the definition of Any Fit algorithm is here resolved by taking the bin with the largest residual capacity.

Next: the pseudo-code of WF.

Algorithm: Worst Fit (1/3)

Input : BPP instance I with n items, weights $w[1..n]$, capacity c

Output: Item-bin assignment $bin[1..n]$ and number of used bins nb

```
1 function WorstFit(n, w, c)
2 |   bin := new array[1..n]     // item-bin assignment
3 |   rc := new array[1..n]      // residual capacities
4 |   nb := 1                          // number of bins
5 |   rc[1] ← c                 // initialize first bin
```

Algorithm: Worst Fit (2/3)

function *WorstFit*(n, w, c) (continued)

```
6   |   for i := 1 to n do
7   |   |   maxrc := −1  // largest residual capacity
    |   |    found so far
8   |   |   s := −1                          // selected bin
9   |   |   for b := 1 to nb do
10  |   |   |   if w[i] ≤ rc[b] and rc[b] > maxrc then
11  |   |   |   |   s ← b
12  |   |   |   |   maxrc ← rc[b]
13  |   |   |   end
14  |   |   end
```

Algorithm: Worst Fit (3/3)

```
function WorstFit(n, w, c) (continued)
    for i := 1 to n (continued) do
15      if s = −1 then
16          nb ← nb + 1                    // open new bin
17          rc[nb] ← c
18          s ← nb
19      end
20      bin[i] ← s                          // assign item
21      rc[s] ← rc[s] − w[i]
22  end
23  return bin, nb
24 end
```

Example: execution of WF (1/4)

We consider a bin of capacity $c = 10$ and a set $\mathscr{I} = \{1, 2, 3, 4, 5\}$ of 5 items ($n = 5$), with the following item weights:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
w_i	6	5	4	7	3

The WF algorithm produces the feasible solution of the next slides.

Example: execution of WF (2/4)

1. Item 1 (weight 6): bin 1 is open at the start ($nb = 1$, $rc[1] = c = 10$) and $w[1] \leq rc[1]$, so $s = 1$ and WF packs item 1 into bin 1, and $rc[1] \leftarrow 10 - 6 = 4$.

2. Item 2 (weight 5): $maxrc := -1$, $s := -1$. $b = 1$: $w[2] = 5 \leq rc[1] = 4$? No. No bin fits ($s = -1$), so WF opens bin 2 ($rc[2] \leftarrow 10$). Item 2 is packed into bin 2, and $rc[2] \leftarrow 10 - 5 = 5$.

3. Item 3 (weight 4): $maxrc := -1$, $s := -1$. $b = 1$: $w[3] = 4 \leq rc[1] = 4$ and $rc[1] = 4 > -1$, so $s \leftarrow 1$, $maxrc \leftarrow 4$. $b = 2$: $w[3] = 4 \leq rc[2] = 5$ and $rc[2] = 5 > maxrc = 4$, so $s \leftarrow 2$, $maxrc \leftarrow 5$. Hence $s = 2$: item 3 is packed into bin 2 (the one with the largest residual capacity), and $rc[2] \leftarrow 5 - 4 = 1$.

Example: execution of WF (3/4)

4. Item 4 (weight 7): $b = 1$: $w[4] = 7 \leq rc[1] = 4$? No. $b = 2$: $w[4] = 7 \leq rc[2] = 1$? No. No bin fits ($s = -1$), so WF opens bin 3 ($rc[3] \leftarrow 10$). Item 4 is packed into bin 3, and $rc[3] \leftarrow 10 - 7 = 3$.

5. Item 5 (weight 3): $maxrc := -1$, $s := -1$. $b = 1$: $w[5] = 3 \leq rc[1] = 4$ and $rc[1] = 4 > -1$, so $s \leftarrow 1$, $maxrc \leftarrow 4$. $b = 2$: $w[5] = 3 \leq rc[2] = 1$? No. $b = 3$: $w[5] = 3 \leq rc[3] = 3$ and $rc[3] = 3 > maxrc = 4$? No. Hence $s = 1$: item 5 is packed into bin 1 (the one with the largest residual capacity), and $rc[1] \leftarrow 4 - 3 = 1$.

Example: execution of WF (4/4)

Hence WF returns:

$$bin = \begin{pmatrix} 1 & 2 & 2 & 3 & 1 \end{pmatrix}, \qquad nb = 3,$$

and therefore

$$WF(I) = nb = 3.$$

Example: packing patterns from the bin vector (1/2)

Consider the instance of the previous example where WF returns:

$$bin = \begin{pmatrix} 1 & 2 & 2 & 3 & 1 \end{pmatrix}, \qquad nb = 3.$$

The packing patterns are:

$$P_1 = \{i \in \{1,2,3,4,5\} : bin_i = 1\} = \{1,5\},$$
$$P_2 = \{i \in \{1,2,3,4,5\} : bin_i = 2\} = \{2,3\},$$
$$P_3 = \{i \in \{1,2,3,4,5\} : bin_i = 3\} = \{4\}.$$

Example: packing patterns from the bin vector (2/2)

These are exactly the sets of items packed into the bins constructed by the algorithm:

- Bin 1 contains items 1 and 5 with total weight $w_1 + w_5 = 6 + 3 = 9$;
- Bin 2 contains items 2 and 3 with total weight $w_2 + w_3 = 5 + 4 = 9$;
- Bin 3 contains item 4 with weight $w_4 = 7$.

Summary (1/2)

The Any Fit class:

- keeps all the bins opened so far, and opens a new bin only when the item fits nowhere
- the selection rule is left free: each rule gives a member of the family
- the items of two bins of an Any Fit solution never fit together

Next Fit:

- is not an Any Fit algorithm: on $w = (3, 8, 3, 8)$ it uses 4 bins, every Any Fit algorithm 3

Summary (2/2)

Three members of the class:

- First Fit: the first bin in which the item fits
- Best Fit: the bin with the smallest residual capacity
- Worst Fit: the bin with the largest residual capacity
- ties: the bin opened first

On the instance $w = (6, 5, 4, 7, 3)$, $c = 10$:

- 3 bins each, with three different assignments: $(1, 2, 1, 3, 2)$, $(1, 2, 1, 3, 3)$, $(1, 2, 2, 3, 1)$

Half-full property

Introduction: Half-full property

- In this section we define a structural property of a BPP solution: all the bins but at most one are more than half full
- We prove that every Any Fit algorithm produces solutions with this property
- This property is exactly what we need, in the next section, to prove the guarantee 2

The half-full property (1/2)

- The results of this section do not depend on the rule used to select the bin.
- They are proved using *only* the Any Fit condition, and therefore hold for every algorithm of the family, including FF, BF and WF.

The half-full property (2/2)

- The **half-full property** of a feasible BPP solution states that, with the exception of at most one bin, all bins are filled to more than half of their capacity, or equivalently, that for any pair of distinct bins at least one of them is filled to more than half capacity.
- The half-full property alone does *not* guarantee that two bins cannot be merged: for instance, with $c = 10$ two bins of total weight 6 and 1 satisfy the property but can be merged into a single bin.
- Its role is different: it is exactly what is needed to prove a relative performance guarantee of 2.

Definition 2: Half-full property (1/2)

Given a feasible BPP solution with k bins and packing patterns $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ (where P_b is the set of items packed into bin b, for $b \in \{1, 2, \ldots, k\}$), we say that the solution satisfies the ***half-full property*** *if one of the following equivalent conditions holds:*

- *there exists at most one bin $b \in \{1, 2, \ldots, k\}$ such that*

$$\sum_{i \in P_b} w_i \leq \frac{c}{2}.$$

Definition: Half-full property (2/2)

- *for any two distinct bins* $b, b' \in \{1, 2, \ldots, k\}$*, at least one of them is filled to more than half of its capacity, i.e.,*

$$\sum_{i \in P_b} w_i > \frac{c}{2} \qquad \text{or} \qquad \sum_{i \in P_{b'}} w_i > \frac{c}{2}.$$

Example: half-full property (1/3)

We consider a bin of capacity $c = 10$ and a set $\mathscr{I} = \{1, 2, 3, 4\}$ of 4 items ($n = 4$), with the following item weights:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$
w_i	5	5	5	5

Initially $nb := 1$, $rc := c$. The NF algorithm produces the feasible solution of the next slide.

Example: half-full property (2/3)

1. Item 1 (weight $w[1] = 5$): $w[1] = 5 \leq rc = 10$, so no new bin is opened. Item 1 is packed into bin $nb = 1$, and $rc \leftarrow 10 - 5 = 5$.
2. Item 2 (weight $w[2] = 5$): $w[2] = 5 \leq rc = 5$, so no new bin is opened. Item 2 is packed into bin $nb = 1$, and $rc \leftarrow 5 - 5 = 0$.
3. Item 3 (weight $w[3] = 5$): $w[3] = 5 > rc = 0$, so $nb \leftarrow 2$ and $rc \leftarrow c = 10$. Item 3 is packed into bin $nb = 2$, and $rc \leftarrow 10 - 5 = 5$.
4. Item 4 (weight $w[4] = 5$): $w[4] = 5 \leq rc = 5$, so no new bin is opened. Item 4 is packed into bin $nb = 2$, and $rc \leftarrow 5 - 5 = 0$.

Example: half-full property (3/3)

Hence NF returns:

$$bin = \begin{pmatrix} 1 & 1 & 2 & 2 \end{pmatrix}, \qquad nb = 2.$$

The packing patterns are $P_1 = \{1, 2\}$, $P_2 = \{3, 4\}$, with total weights:

$$\sum_{i \in P_1} w_i = 5 + 5 = 10 > \frac{c}{2} = 5, \qquad \sum_{i \in P_2} w_i = 5 + 5 = 10 > \frac{c}{2} = 5.$$

Since there is *no* bin with total weight at most $\frac{c}{2}$, the solution **satisfies** the half-full property.

Proposition 1

For every BPP instance, the heuristic BPP solutions produced by any Any Fit algorithm satisfy the half-full property. In particular this holds for FF, BF and WF.

- The proof is by contradiction: two bins at most half full would contain items that fit together into a single bin.
- This is excluded by the property of the Any Fit class proved in the previous section.

Proof (1/2)

Let $\mathscr{A}$ be an Any Fit algorithm, let $k := \mathscr{A}(I)$, and let $P_1, P_2, \ldots, P_k$ be the packing patterns produced by $\mathscr{A}$, indexed in the order the bins are opened. If $k = 1$ there is nothing to prove, so assume $k \geq 2$.

Suppose, by contradiction, that there exist two distinct bins b and b' such that

$$\sum_{i \in P_b} w_i \leq \frac{c}{2} \qquad \text{and} \qquad \sum_{i \in P_{b'}} w_i \leq \frac{c}{2}.$$

Proof (2/2)

Summing the two inequalities gives

$$\sum_{i \in P_b} w_i + \sum_{i \in P_{b'}} w_i \leq \frac{c}{2} + \frac{c}{2} = c,$$

which contradicts the property of the Any Fit class proved in the previous section, namely that the items of two distinct bins never fit together into a single bin.

Therefore there can be at most one bin with total weight at most $\frac{c}{2}$.

Only the Any Fit condition is used, so the statement holds for every algorithm of the class. □

Example: half-full property for FF (1/3)

As an illustration of the proposition above, we consider a bin of capacity $c = 10$ and a set $\mathscr{I} = \{1, 2, 3, 4\}$ of 4 items ($n = 4$), with the following item weights:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$
w_i	7	7	3	4

Initially $nb := 1$, $rc[1] := c = 10$. The FF algorithm produces the feasible solution of the next slide.

Example: half-full property for FF (2/3)

1. Item 1 (weight 7): $b = 1$: $w[1] = 7 \leq rc[1] = 10$? Yes, so $s \leftarrow 1$. Item 1 is packed into bin 1, and $rc[1] \leftarrow 10 - 7 = 3$.

2. Item 2 (weight 7): $b = 1$: $w[2] = 7 \leq rc[1] = 3$? No. No bin fits, so FF opens bin 2 ($rc[2] \leftarrow 10$). Item 2 is packed into bin 2, and $rc[2] \leftarrow 10 - 7 = 3$.

3. Item 3 (weight 3): $b = 1$: $w[3] = 3 \leq rc[1] = 3$? Yes, so $s \leftarrow 1$. Item 3 is packed into bin 1, and $rc[1] \leftarrow 3 - 3 = 0$.

4. Item 4 (weight 4): $b = 1$: $w[4] = 4 \leq rc[1] = 0$? No. $b = 2$: $w[4] = 4 \leq rc[2] = 3$? No. No bin fits, so FF opens bin 3 ($rc[3] \leftarrow 10$). Item 4 is packed into bin 3, and $rc[3] \leftarrow 10 - 4 = 6$.

Example: half-full property for FF (3/3)

Hence FF returns:

$$bin = \begin{pmatrix} 1 & 2 & 1 & 3 \end{pmatrix}, \qquad nb = 3.$$

The packing patterns are $P_1 = \{1, 3\}$, $P_2 = \{2\}$, $P_3 = \{4\}$, with total weights:

$$\sum_{i \in P_1} w_i = 7 + 3 = 10 > \frac{c}{2} = 5,$$

$$\sum_{i \in P_2} w_i = 7 > \frac{c}{2} = 5,$$

$$\sum_{i \in P_3} w_i = 4 \leq \frac{c}{2} = 5.$$

Since there is *exactly one* bin (bin 3) with total weight at most $\frac{c}{2}$, the solution **satisfies** the half-full property, as guaranteed by the proposition above since FF is an Any Fit algorithm.

Next Fit and the half-full property

The Next Fit may produce solutions with many bins filled to less than half capacity,

and therefore does not satisfy the half-full property.

Example: half-full property for NF (1/2)

We consider again the instance of the example on NF and the Any Fit class, with bin capacity $c = 10$ and item weights $w = (3, 8, 3, 8)$, on which NF opens a new bin for every item and returns:

$$bin = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}, \qquad nb = 4.$$

The packing patterns are:

$$P_1 = \{i \in \{1,2,3,4\} : bin_i = 1\} = \{1\},$$
$$P_2 = \{i \in \{1,2,3,4\} : bin_i = 2\} = \{2\},$$
$$P_3 = \{i \in \{1,2,3,4\} : bin_i = 3\} = \{3\},$$
$$P_4 = \{i \in \{1,2,3,4\} : bin_i = 4\} = \{4\}.$$

Example: half-full property for NF (2/2)

The total weights in the bins are:

$$\sum_{i \in P_1} w_i = 3 \leq \frac{c}{2} = 5, \qquad \sum_{i \in P_2} w_i = 8 > \frac{c}{2} = 5,$$

$$\sum_{i \in P_3} w_i = 3 \leq \frac{c}{2} = 5, \qquad \sum_{i \in P_4} w_i = 8 > \frac{c}{2} = 5.$$

Since there are *two* bins (bins 1 and 3) with total weight at most $\frac{c}{2}$, the solution does **not** satisfy the half-full property.

Summary

The half-full property:

- all the bins but at most one have total weight larger than $\frac{c}{2}$
- equivalently: of any two distinct bins, at least one is more than half full
- it does not say that no two bins can be merged

Who satisfies it:

- every Any Fit algorithm (FF, BF, WF), whatever the selection rule
- not NF: on $w = (3, 8, 3, 8)$ two bins have total weight at most $\frac{c}{2}$

Performance guarantees

Introduction: Performance guarantees

- In this section we determine the worst case performance ratio of the four algorithms
- We prove the guarantee 2 and we show families of instances on which it is tight
- We state the better ratio $\frac{17}{10}$ of First Fit and Best Fit, and see what the Any Fit condition alone can and cannot give

The plan of the section

- Next Fit is not an Any Fit algorithm, so it needs an argument of its own.
- For the three Any Fit algorithms the bound $\mathscr{A}(I) \leq 2\,z(I)$ comes at once from the half-full property, without any assumption on the rule used to select the bin.
- The question is how tight that bound is: for WF the factor 2 cannot be improved, whereas FF and BF obey a strictly better bound.

Proposition 2

Let $\mathscr{A}$ be a heuristic algorithm for the BPP that produces solutions satisfying the half-full property. Then, for every BPP instance I, we have

$$\frac{\mathscr{A}(I)}{z(I)} \leq 2.$$

- The proof compares two bounds on the total weight of the items: a lower bound from the half-full property, an upper bound from the optimal solution.

Proof (1/3)

Let $k := \mathscr{A}(I)$, and let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be the packing patterns produced by $\mathscr{A}$, indexed in the order the bins are opened.

Since the solution satisfies the half-full property, there exists at most one bin $b \in \{1, 2, \ldots, k\}$ such that

$$\sum_{i \in P_b} w_i \leq \frac{c}{2}.$$

Therefore, at least $k - 1$ bins have total weight strictly greater than $\frac{c}{2}$. Summing over these bins, we obtain

$$\sum_{i \in \mathscr{I}} w_i \; > \; (k-1)\,\frac{c}{2}.$$

Proof (2/3)

On the other hand, any feasible packing using $z(I)$ bins satisfies

$$\sum_{i \in \mathscr{I}} w_i \;\leq\; z(I)\, c.$$

Combining the two inequalities yields

$$(k-1)\,\frac{c}{2} \;<\; \sum_{i \in \mathscr{I}} w_i \;\leq\; z(I)\, c$$

$$\Longrightarrow \quad k - 1 \;<\; 2\, z(I)$$

$$\Longrightarrow \quad k \;<\; 2\, z(I) + 1.$$

Proof (3/3)

Since both k and $z(I)$ are positive integers, we have

$$k \leq 2\, z(I).$$

Dividing by $z(I) \geq 1$ gives

$$\frac{k}{z(I)} = \frac{\mathscr{A}(I)}{z(I)} \leq 2.$$

□

The guarantee of the Any Fit algorithms

- Since every Any Fit algorithm produces solutions satisfying the half-full property, it follows immediately that

$$\frac{\mathscr{A}(I)}{z(I)} \leq 2 \qquad \text{for every Any Fit algorithm } \mathscr{A}$$

and for every BPP instance I.

- In particular

$$\frac{FF(I)}{z(I)} \leq 2, \qquad \frac{BF(I)}{z(I)} \leq 2 \qquad \text{and} \qquad \frac{WF(I)}{z(I)} \leq 2,$$

and therefore FF, BF and WF are 2-approximation algorithms.

Proposition 3

For every BPP instance I, we have:

$$\frac{NF(I)}{z(I)} \leq 2.$$

- NF does not satisfy the half-full property, so the previous proposition does not apply.
- The proof uses instead two *consecutive* bins: their items never fit together.

Proof (1/4)

Let $k := NF(I)$, and let $P_1, P_2, \ldots, P_k$ be the packing patterns produced by NF, indexed in the order the bins are opened.

Fix any $b \in \{1, 2, \ldots, k-1\}$. The packing pattern P_b corresponds to a bin that is closed when the next item (call it f) does not fit into it, hence

$$\sum_{i \in P_b} w_i + w_f > c.$$

That same item f is the first item placed into the next bin, whose packing pattern is P_{b+1}, so

$$w_f \leq \sum_{i \in P_{b+1}} w_i.$$

Proof (2/4)

Therefore,

$$\sum_{i \in P_b} w_i + \sum_{i \in P_{b+1}} w_i \; > \; c.$$

Now consider the disjoint consecutive pairs $(P_1, P_2), (P_3, P_4), \ldots$.
Summing the inequality above over the $\lfloor k/2 \rfloor$ pairs yields

$$\underbrace{\sum_{b=1}^{k} \sum_{i \in P_b} w_i}_{= \sum_{i \in \mathscr{I}} w_i \; \leq \; z(I)\, c} \; > \; \left\lfloor \frac{k}{2} \right\rfloor c \quad \Longrightarrow \quad z(I) \; > \; \left\lfloor \frac{k}{2} \right\rfloor.$$

Proof (3/4)

Since $z(I)$ is an integer, it follows that

$$z(I) \geq \left\lfloor \frac{k}{2} \right\rfloor + 1.$$

Then

$$\begin{cases} \text{if } k \text{ is even, then } \left\lfloor \frac{k}{2} \right\rfloor = \frac{k}{2} \implies z(I) \geq \frac{k}{2} + 1 \\ \qquad\qquad\qquad\qquad\qquad\qquad \implies k \leq 2z(I) - 2 < 2z(I), \\ \text{if } k \text{ is odd, then } \left\lfloor \frac{k}{2} \right\rfloor = \frac{k-1}{2} \implies z(I) \geq \frac{k+1}{2} \\ \qquad\qquad\qquad\qquad\qquad\qquad \implies k \leq 2z(I) - 1 < 2z(I). \end{cases}$$

Proof (4/4)

Therefore $k < 2z(I)$, and dividing by $z(I) \geq 1$ yields

$$\frac{k}{z(I)} = \frac{NF(I)}{z(I)} \leq 2.$$

□

Proposition 4

There are BPP instances I_n such that:

$$\frac{NF(I_n)}{z(I_n)} \rightarrow 2 \qquad as \qquad n \rightarrow +\infty.$$

- The proof builds a family of instances that alternate a large item and a small item.
- NF puts each large item in its own bin, while the optimum puts two large items per bin.

Proof (1/4)

Fix an integer $n \geq 8$ divisible by 4 and consider the BPP instance I_n with bin capacity $c = n$ and item set $\mathscr{I} = \{1, 2, \ldots, n\}$, where the item weights are

$$w_i = \begin{cases} \frac{n}{2}, & \text{if } i \text{ is odd,} \\ 2, & \text{if } i \text{ is even.} \end{cases}$$

Proof (2/4)

Items are processed by NF in the natural order $1, 2, \ldots, n$. For every odd i,

$$w_i + w_{i+1} = \frac{n}{2} + 2 < n,$$

so NF packs items i and $i+1$ into the same bin. The remaining space is

$$n - \left(\frac{n}{2} + 2\right) = \frac{n}{2} - 2 < \frac{n}{2},$$

hence the next item of weight $\frac{n}{2}$ does not fit and NF opens a new bin. Therefore,

$$NF(I_n) = \frac{n}{2}.$$

Proof (3/4)

Pack the $\frac{n}{2}$ items of weight $\frac{n}{2}$ two per bin, obtaining $\frac{n}{4}$ full bins, and pack all $\frac{n}{2}$ small items together (since $\frac{n}{2} \cdot 2 = n$) in one additional bin; thus $z(I_n) \leq \frac{n}{4} + 1$.

On the other hand,

$$z(I_n) \geq \left\lceil \frac{\sum_{i \in \mathscr{I}} w_i}{c} \right\rceil = \left\lceil \frac{\frac{n}{2} \frac{n}{2} + \frac{n}{2} 2}{n} \right\rceil = \left\lceil \frac{n}{4} + 1 \right\rceil = \frac{n}{4} + 1.$$

Hence

$$z(I_n) = \frac{n}{4} + 1.$$

Proof (4/4)

Consequently,

$$\lim_{n\to\infty} \frac{NF(I_n)}{z(I_n)} = \lim_{n\to\infty} \frac{\frac{n}{2}}{\frac{n}{4}+1} = \lim_{n\to\infty} \frac{\frac{n}{2}}{\frac{n+4}{4}}$$

$$= \lim_{n\to\infty} \frac{2n}{n+4} = \lim_{n\to\infty} \frac{2}{1+\frac{4}{n}} = 2.$$

□

The worst case performance ratio of NF

From the last two propositions it follows that
the worst case performance ratio of NF is two, i.e., we have

$$r(NF) = 2$$

Proposition 5

There are BPP instances I_n such that:

$$\frac{WF(I_n)}{z(I_n)} \rightarrow 2 \qquad as \qquad n \rightarrow +\infty.$$

- We already know that $\frac{WF(I)}{z(I)} \leq 2$ for every instance, since WF is an Any Fit algorithm.
- The proof uses the same family of instances as for NF, and an induction on the number of processed items.

Proof (1/5)

Fix an integer $n \geq 8$ divisible by 4 and consider the BPP instance I_n with bin capacity $c = n$ and item set $\mathscr{I} = \{1, 2, \ldots, n\}$, where the item weights are

$$w_i = \begin{cases} \frac{n}{2}, & \text{if } i \text{ is odd,} \\ 2, & \text{if } i \text{ is even,} \end{cases}$$

that is, the same instance used above for NF. Items are processed by WF in the natural order $1, 2, \ldots, n$.

We show by induction that, after the first $2k$ items have been processed, WF has opened exactly k bins, each with residual capacity $\frac{n}{2} - 2$.

Proof (2/5)

For $k = 1$, item 1 (of weight $\frac{n}{2}$) is packed into bin 1, whose residual capacity becomes $\frac{n}{2}$; then item 2 (of weight 2) is packed into bin 1, the only open bin, whose residual capacity becomes $\frac{n}{2} - 2$.

Assume the claim holds for some k, and consider items $2k + 1$ and $2k + 2$.

Item $2k + 1$ has weight $\frac{n}{2}$, while every open bin has residual capacity $\frac{n}{2} - 2 < \frac{n}{2}$: the item fits nowhere, so WF opens bin $k + 1$, whose residual capacity becomes $\frac{n}{2}$.

Proof (3/5)

Item $2k+2$ has weight 2 and fits into every open bin; since bin $k+1$ is the one with the *largest* residual capacity, WF packs the item there, and the residual capacity of bin $k+1$ becomes $\frac{n}{2}-2$. This proves the claim for $k+1$.

It is precisely the Worst Fit rule that keeps every bin at residual capacity $\frac{n}{2}-2$: the small items are always placed in the emptiest bin, so no bin ever receives two items of weight $\frac{n}{2}$.

Taking $k=\frac{n}{2}$ we obtain

$$WF(I_n)=\frac{n}{2}.$$

Proof (4/5)

Pack the $\frac{n}{2}$ items of weight $\frac{n}{2}$ two per bin, obtaining $\frac{n}{4}$ full bins, and pack all $\frac{n}{2}$ small items together (since $\frac{n}{2} \cdot 2 = n$) in one additional bin; thus $z(I_n) \leq \frac{n}{4} + 1$.

On the other hand,

$$z(I_n) \geq \left\lceil \frac{\sum_{i \in \mathscr{I}} w_i}{c} \right\rceil = \left\lceil \frac{\frac{n}{2}\ \frac{n}{2} + \frac{n}{2}\ 2}{n} \right\rceil = \left\lceil \frac{n}{4} + 1 \right\rceil = \frac{n}{4} + 1,$$

hence

$$z(I_n) = \frac{n}{4} + 1.$$

Proof (5/5)

Consequently,

$$\lim_{n\to\infty} \frac{WF(I_n)}{z(I_n)} = \lim_{n\to\infty} \frac{\frac{n}{2}}{\frac{n}{4}+1}$$

$$= \lim_{n\to\infty} \frac{2n}{n+4} = \lim_{n\to\infty} \frac{2}{1+\frac{4}{n}} = 2.$$

□

The worst case performance ratio of WF

From the guarantee of the half-full property
and the last proposition it follows that
the worst case performance ratio of WF is two, i.e., we have

$$r(WF) = 2$$

For WF the factor 2 therefore cannot be improved.

WF and NF

- We have $r(WF) = r(NF) = 2$: in the worst case WF is no better than NF.
- The two algorithms are however far from equivalent. WF never opens a new bin while the item still fits into one of the bins opened so far, whereas NF closes a bin as soon as an item does not fit and never reconsiders it.
- As a consequence NF may leave arbitrarily many bins filled to less than half of their capacity, whereas WF, as shown in the section on the half-full property, can leave at most one.
- The value $r(WF) = 2$ is due to

  D.S. Johnson, *Fast algorithms for bin packing*, Journal of Computer and System Sciences, 8(3):272–314, 1974.

Proposition 6

For every BPP instance I, we have:

$$FF(I) \leq \frac{17}{10} z(I) \quad \textit{and} \quad BF(I) \leq \frac{17}{10} z(I).$$

- The selection rules of FF and BF give a bound strictly better than the factor 2 of the Any Fit class.
- The proof is long, and it is omitted: we give the papers where it can be found.

Proof (1/2)

Proof omitted. The two bounds were first proved in the weaker form

$$FF(I) \leq \frac{17}{10} z(I) + 2 \qquad \text{and} \qquad BF(I) \leq \frac{17}{10} z(I) + 2$$

in

D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey and R.L. Graham, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM Journal on Computing, 3(4):299–325, 1974.

Proof (2/2)

The additive constant has later been removed, in the form stated above, respectively in

G. Dósa and J. Sgall, *First Fit bin packing: a tight analysis*, STACS 2013, pp. 538–549,

G. Dósa and J. Sgall, *Optimal analysis of Best Fit bin packing*, ICALP 2014, pp. 429–441. □

Proposition 7

There are BPP instances I_n such that:

$$\frac{FF(I_n)}{z(I_n)} \to \frac{17}{10} \quad \textit{and} \quad \frac{BF(I_n)}{z(I_n)} \to \frac{17}{10} \qquad \textit{as} \qquad n \to +\infty.$$

- Together with the previous proposition, it shows that the factor $\frac{17}{10}$ of FF and BF cannot be improved.
- Also this proof is omitted: we give the works where the families of instances can be found.

Proof

Proof omitted. Families of instances on which the ratio of FF and BF approaches $\frac{17}{10}$ are given in the paper by Johnson, Demers, Ullman, Garey and Graham cited above, and in

D.S. Johnson, *Near-optimal bin packing algorithms*, PhD thesis, Massachusetts Institute of Technology, 1973. □

The worst case performance ratios of FF and BF

From the last two propositions it follows that the worst case performance ratio of FF and BF is seventeen tenths, i.e., we have

$$r(FF) = r(BF) = \frac{17}{10}$$

Unlike WF, FF and BF stay strictly below the factor 2 that the Any Fit condition alone guarantees.

What the Any Fit condition gives

- The Any Fit condition alone guarantees the half-full property, and therefore the bound

$$\mathscr{A}(I) \leq 2\, z(I)$$

for every algorithm $\mathscr{A}$ of the class.

- This bound cannot be improved for the class as a whole: WF is an Any Fit algorithm and, as shown above, $r(WF) = 2$.

What the Any Fit condition leaves free (1/2)

- The rule used to select one of the bins in which the item fits is what distinguishes the members of the family.
- On the instances of the tightness proposition for FF and BF the ratio of FF and BF tends to $\frac{17}{10}$, while on those of the tightness proposition for WF the ratio of WF tends to 2: the selection rule alone accounts for the whole gap between $\frac{17}{10}$ and 2.

What the Any Fit condition leaves free (2/2)

- It is known that no algorithm of the class can go below $\frac{17}{10}$, so $\frac{17}{10}$ and 2 are exactly the best and the worst behaviour that the Any Fit condition allows, and both are attained, the first by FF and BF, the second by WF.

- The Any Fit class and the bounds above are due to the paper of Johnson already cited for $r(WF) = 2$: in particular, the lower bound $\frac{17}{10}$ for every Any Fit algorithm is due to that paper (D.S. Johnson, 1974).

Summary (1/2)

The guarantee 2:

- the half-full property gives $\mathscr{A}(I) \leq 2\, z(I)$, for every Any Fit algorithm
- NF: two consecutive bins never fit together, so $NF(I) \leq 2\, z(I)$ as well

Tight families:

- $c = n$, weights $\frac{n}{2}$ and 2 alternated: $NF(I_n) = WF(I_n) = \frac{n}{2}$ and $z(I_n) = \frac{n}{4} + 1$
- hence $r(NF) = r(WF) = 2$

Summary (2/2)

First Fit and Best Fit:

- $FF(I) \leq \frac{17}{10}\, z(I)$ and $BF(I) \leq \frac{17}{10}\, z(I)$ (proofs omitted)
- $r(FF) = r(BF) = \frac{17}{10}$

The Any Fit class:

- $\frac{17}{10}$ and 2 are the best and the worst behaviour that the Any Fit condition allows
- the selection rule alone accounts for the whole gap

Typical mistakes

Introduction: Typical mistakes

- We collect the mistakes that are made most often with the algorithms of this chapter
- Two of them concern the execution of the algorithms, three the meaning of the results
- Check them before running an algorithm by hand or reading a guarantee

Typical mistakes (1/3)

Observation 1: Believing that Next Fit is an Any Fit algorithm

NF never reconsiders a closed bin, so it may open a new bin for an item that would fit into an older one (as in the example with weights $(3, 8, 3, 8)$*).*

Observation 2: Choosing the wrong bin in BF and WF

BF packs the item into the bin with the smallest residual capacity among the bins in which it fits, WF into the one with the largest residual capacity; in case of ties, both take the bin of smallest index.

Typical mistakes (2/3)

Observation 3: Reading the half-full property as "no two bins can be merged"

Two bins of total weight 6 and 1 with $c = 10$ satisfy the property and can be merged; the property is what proves the guarantee 2.

Observation 4: Treating $r = 2$ as a value that is attained

The worst-case ratios of NF and WF are suprema: the families of instances approach 2 as n grows, and no instance reaches it.

Typical mistakes (3/3)

Observation 5: Changing the order of the items

The four algorithms process the items in the order in which they are given; sorting them first is a different algorithm.

Summary

Executing the algorithms:

- NF never reconsiders a closed bin: it is not an Any Fit algorithm
- BF: smallest residual capacity, WF: largest; ties to the smallest index
- keep the given order of the items

Reading the results:

- the half-full property does not forbid merging two bins
- $r = 2$ is approached by NF and WF, never reached

C++ implementation

Introduction: C++ implementation

- The four algorithms are implemented in C++, one function each
- We look at short excerpts: the Next Fit, and the scan of the bins that distinguishes First Fit, Best Fit and Worst Fit
- We follow the execution of each function step by step on the instance of the chapter

Indexing and instance (1/2)

- In this C++ implementation section we use 0-based indexing (items and bins are numbered starting from 0). The algorithmic logic is unchanged.

- The step-by-step executions all refer to the same BPP instance, namely a bin of capacity $c = 10$ and a set $\mathscr{I} = \{1, 2, 3, 4, 5\}$ of 5 items ($n = 5$), with the following item weights:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
w_i	6	5	4	7	3

Indexing and instance (2/2)

- With 0-based indexing the items are numbered $0, 1, 2, 3, 4$ and their weights are $w[0] = 6$, $w[1] = 5$, $w[2] = 4$, $w[3] = 7$ and $w[4] = 3$. This is the instance used in the examples of the previous sections, so the solutions found here are the ones already computed there.

The Next Fit in C++

```
1  int NextFit(int n, int c, int *w, int *bin)
2  {
3      int nb = 1;
4      int rc  = c;
5      for (int i = 0; i < n; i++)
6      {
7          if (w[i] > rc)
8          {
9              nb++;
10             rc = c;
11         }
12         bin[i] = nb - 1;
13         rc -= w[i];
14     }
15     return nb;
16 }
```

- `nb` is the current bin (number of open bins), `rc` its residual capacity; item `i` goes into bin `nb - 1`.

Example: C++ execution of NF (1/2)

Initialization

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	-	-	-	-	-

$nb = 1 \quad rc = 10$

Iteration 1 – Item 0 is packed into bin 0

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	-	-	-	-

$nb = 1 \quad rc = 4$

Iteration 2 – Item 1 is packed into bin 1

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	-	-	-

$nb = 2 \quad rc = 5$

Example: C++ execution of NF (2/2)

Iteration 3 – Item 2 is packed into bin 1

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	-	-

$nb = 2 \quad rc = 1$

Iteration 4 – Item 3 is packed into bin 2

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	2	-

$nb = 3 \quad rc = 3$

Iteration 5 – Item 4 is packed into bin 2

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	2	2

$nb = 3 \quad rc = 0$

The First Fit in C++: the scan of the bins

```
1 int s = -1;
2 for (int b = 0; b < nb; b++)
3 {
4     if (w[i] <= rc[b])
5     {
6         s = b;
7         break;
8     }
9 }
```

- `s` is the selected bin for item `i` (`-1` means: not found yet).
- The bins are scanned in the order they were opened, and the first that fits is picked.
- The array `rc` stores the residual capacities of the bins (at most n bins), with `rc[0] = c` at the start.

The First Fit in C++: opening a new bin

```
1 if (s == -1)
2 {
3     rc[nb] = c;
4     s = nb;
5     nb++;
6 }
7 bin[i] = s;
8 rc[s] -= w[i];
```

- If no open bin can accommodate item `i`, a new bin is opened, with residual capacity `c`, and selected.
- Item `i` is packed into the selected bin, and its residual capacity is updated.
- These lines are the same in First Fit, Best Fit and Worst Fit: only the scan of the bins changes.

Example: C++ execution of FF (1/3)

Initialization $nb = 1$

$bin[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	-	-	-	-	-

rc$[b]$	$b = 0$	$b = 1$	$b = 2$	$b = 3$	$b = 4$
	10	-	-	-	-

Iteration 1 – Item 0 is packed into bin 0 $nb = 1$

$bin[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	0	-	-	-	-

rc$[b]$	$b = 0$	$b = 1$	$b = 2$	$b = 3$	$b = 4$
	4	-	-	-	-

Example: C++ execution of FF (2/3)

Iteration 2 – Item 1 is packed into bin 1 $nb = 2$

$bin[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	0	1	-	-	-

rc$[b]$	$b = 0$	$b = 1$	$b = 2$	$b = 3$	$b = 4$
	4	5	-	-	-

Iteration 3 – Item 2 is packed into bin 0 $nb = 2$

$bin[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	0	1	0	-	-

rc$[b]$	$b = 0$	$b = 1$	$b = 2$	$b = 3$	$b = 4$
	0	5	-	-	-

Example: C++ execution of FF (3/3)

Iteration 4 – Item 3 is packed into bin 2 $nb = 3$

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	2	-

$rc[b]$	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	5	3	-	-

Iteration 5 – Item 4 is packed into bin 1 $nb = 3$

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	2	1

$rc[b]$	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	2	3	-	-

The Best Fit in C++: the scan of the bins

```
1  int s = -1;
2  int minrc = c + 1;
3  for (int b = 0; b < nb; b++)
4  {
5      if (w[i] <= rc[b] && rc[b] < minrc)
6      {
7          minrc = rc[b];
8          s = b;
9      }
10 }
```

- `minrc` is the smallest residual capacity found so far.
- Among the bins where item `i` fits, the one with the smallest residual capacity is picked.

Example: C++ execution of BF (1/3)

Initialization $nb = 1$

$bin[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	-	-	-	-	-

rc$[b]$	$b = 0$	$b = 1$	$b = 2$	$b = 3$	$b = 4$
	10	-	-	-	-

Iteration 1 – Item 0 is packed into bin 0 $nb = 1$

$bin[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	0	-	-	-	-

rc$[b]$	$b = 0$	$b = 1$	$b = 2$	$b = 3$	$b = 4$
	4	-	-	-	-

Example: C++ execution of BF (2/3)

Iteration 2 – Item 1 is packed into bin 1 $nb = 2$

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	-	-	-

rc$[b]$	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	4	5	-	-	-

Iteration 3 – Item 2 is packed into bin 0 $nb = 2$

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	-	-

rc$[b]$	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	5	-	-	-

Example: C++ execution of BF (3/3)

Iteration 4 – Item 3 is packed into bin 2 $nb = 3$

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	2	-

$rc[b]$	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	5	3	-	-

Iteration 5 – Item 4 is packed into bin 2 $nb = 3$

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	2	2

$rc[b]$	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	5	0	-	-

The Worst Fit in C++: the scan of the bins

```
1  int s = -1;
2  int maxrc = -1;
3  for (int b = 0; b < nb; b++)
4  {
5      if (w[i] <= rc[b] && rc[b] > maxrc)
6      {
7          maxrc = rc[b];
8          s = b;
9      }
10 }
```

- `maxrc` is the largest residual capacity found so far.
- Among the bins where item `i` fits, the one with the largest residual capacity is picked.

Example: C++ execution of WF (1/3)

Initialization $nb = 1$

$bin[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	-	-	-	-	-

$rc[b]$	$b = 0$	$b = 1$	$b = 2$	$b = 3$	$b = 4$
	10	-	-	-	-

Iteration 1 – Item 0 is packed into bin 0 $nb = 1$

$bin[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	0	-	-	-	-

$rc[b]$	$b = 0$	$b = 1$	$b = 2$	$b = 3$	$b = 4$
	4	-	-	-	-

Example: C++ execution of WF (2/3)

Iteration 2 – Item 1 is packed into bin 1 $nb = 2$

$bin[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	0	1	-	-	-

rc$[b]$	$b = 0$	$b = 1$	$b = 2$	$b = 3$	$b = 4$
	4	5	-	-	-

Iteration 3 – Item 2 is packed into bin 1 $nb = 2$

$bin[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	0	1	1	-	-

rc$[b]$	$b = 0$	$b = 1$	$b = 2$	$b = 3$	$b = 4$
	4	1	-	-	-

Example: C++ execution of WF (3/3)

Iteration 4 – Item 3 is packed into bin 2 $nb = 3$

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	2	-

$\mathrm{rc}[b]$	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	4	1	3	-	-

Iteration 5 – Item 4 is packed into bin 0 $nb = 3$

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	2	0

$\mathrm{rc}[b]$	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	1	1	3	-	-

The codes of the chapter

- All the functions of this section are in the folder `BPP_HEUR_CODE/src`.
- The command `make test` compiles them and runs every example of this chapter and every instance of the exercises, checking the number of bins and the item-bin assignment against the expected ones.

Summary

The code:

- one C++ function per algorithm, with 0-based indexing
- FF, BF and WF differ only in the scan of the bins; opening a new bin is the same

The executions:

- on $w = (6, 5, 4, 7, 3)$, $c = 10$ they give the solutions of the previous sections, shifted by one
- `make test` checks every example and every exercise

In summary

Introduction: In summary

- We collect the key ideas of the chapter on the approximation algorithms for the BPP
- The dashboard gathers the bounds on the example of the chapter

Summary (1/2)

- NF, FF, BF and WF place each item as it arrives; NF keeps one open bin, the other three keep all the bins opened so far and open a new bin only when the item fits nowhere (Any Fit class).
- In an Any Fit solution the items of two bins never fit together, so all the bins but at most one are more than half full (half-full property), and $\mathscr{A}(I) \leq 2\,z(I)$.
- $r(NF) = r(WF) = 2$ and $r(FF) = r(BF) = \frac{17}{10}$: the selection rule alone separates the best and the worst behaviour of the Any Fit class.

Summary (2/2)

Dashboard of the BPP instance $c = 10$, $w = (6, 5, 4, 7, 3)$. The bounds on $z(I)$ found so far:

Source	Lower bound	Upper bound	From
Counting bound	3	–	$\lceil \frac{25}{10} \rceil$
This chapter, NF	–	3	$bin = (1, 2, 2, 3, 3)$
This chapter, FF, BF, WF	–	3	3 bins each, with three different assignments
Optimal value	3	3	lower and upper bound coincide