Approximation algorithms part one

Constant factor approximations

Benjamin Sach
NP-completeness recap

**NP** is the class of *decision* problems we can check the answer to in polynomial time.

A problem \( A \) is **NP-complete** if

\[
A \text{ is in } \text{NP} \\
\text{Every } B \text{ in } \text{NP} \text{ has a polynomial time reduction to } A
\]

(*this second part is the definition of NP-hard*)
NP-completeness recap

NP is the class of decision problems we can check the answer to in polynomial time

A problem \( A \) is NP-complete if

\[ A \text{ is in } \text{NP} \]

Every \( B \) in \( \text{NP} \) has a polynomial time reduction to \( A \)

\( (this \ second \ part \ is \ the \ definition \ of \ NP-hard) \)
NP-completeness recap

**NP** is the class of *decision* problems we can check the answer to *in polynomial time*

A problem $A$ is **NP-complete** if

- $A$ is in **NP**
- Every $B$ in **NP** has a polynomial time reduction to $A$

*(this second part is the definition of **NP-hard*)
NP-completeness recap

\( \text{NP} \) is the class of \textit{decision} problems we can check the answer to \textit{in polynomial time}

A problem \( A \) is \textbf{NP-complete} if

\( A \) is in \textbf{NP}

Every \( B \) in \textbf{NP} has a polynomial time reduction to \( A \)

\textit{(this second part is the definition of \textbf{NP-hard})}
**NP-completeness recap**

**NP** is the class of *decision* problems we can check the answer to in polynomial time.

A problem $A$ is **NP-complete** if

- $A$ is in **NP**
- Every $B$ in **NP** has a polynomial time reduction to $A$

(*this second part is the definition of **NP-hard*)
NP-completeness recap

NP is the class of decision problems we can check the answer to in polynomial time

A problem \( A \) is \textbf{NP-complete} if

\( A \) is in NP

Every \( B \) in NP has a polynomial time reduction to \( A \)

*(this second part is the definition of NP-hard)*

If we could solve \( A \) quickly we could solve every problem in NP quickly
NP-completeness recap

**NP** is the class of *decision* problems we can check the answer to *in polynomial time*

A problem *A* is **NP-complete** if

- *A* is in **NP**
- Every *B* in **NP** has a polynomial time reduction to *A*

*(this second part is the definition of **NP-hard***

*If we could solve *A* quickly we could solve every problem in **NP** quickly*

*They are the ‘hardest’ problems in **NP***
NP-completeness recap

**NP** is the class of *decision* problems we can check the answer to in polynomial time

A problem $A$ is **NP-complete** if

- $A$ is in **NP**
- Every $B$ in **NP** has a polynomial time reduction to $A$

*This second part is the definition of **NP-hard***

If we could solve $A$ quickly we could solve every problem in **NP** quickly

They are the ‘hardest’ problems in **NP**

Most computer scientists *(I’ve met)* believe that you can’t solve them in polynomial time *(i.e. that $P \neq NP$)*
A polynomial time algorithm for an NP-complete problem is worth (a lot more than) a million dollars.
NP-completeness recap

**NP** is the class of *decision* problems we can check the answer to in polynomial time

A problem $A$ is **NP-complete** if

1. $A$ is in **NP**
2. Every $B$ in **NP** has a polynomial time reduction to $A$

*(this second part is the definition of **NP-hard]*)

*If we could solve $A$ quickly we could solve every problem in **NP** quickly*

*They are the ‘hardest’ problems in **NP***
NP-completeness recap

**NP** is the class of *decision* problems we can check the answer to in polynomial time

A problem **A** is NP-complete if

**A** is in **NP**

Every **B** in **NP** has a polynomial time reduction to **A**

(*this second part is the definition of NP-hard*)

If we could solve **A** quickly we could solve every problem in NP quickly

They are the ‘hardest’ problems in NP

So if a problem is NP-complete, we give up right?
Bin packing

1/8 2/8 4/8 7/8 2/8 3/8
Bin packing

Bins

4/8  2/8  4/8  7/8  2/8  3/8
Bin packing

Bins

Items

4/8  2/8  4/8  7/8  2/8  3/8
Bin packing

0 < |item| ≤ 1

Bins

Items

4/8  2/8  4/8  7/8  2/8  3/8
Bin packing

$0 < |\text{Item}| \leq 1$

$I$ is the sum of all item sizes
Bin packing

$|\text{Bin}| = 1$ and there is an unlimited number of bins...

$I$ is the sum of all item sizes
**Bin packing**

**Problem** pack all items into the fewest possible bins

```
1
```

```
4/8  2/8  4/8  7/8  2/8  3/8
```

1
Bin packing

**Problem** pack all items into the fewest possible bins

This is an example of an optimisation problem
Bin packing

**Problem** pack all items into the fewest possible bins

This is an example of an optimisation problem
Bin packing

Problem pack all items into the fewest possible bins

This is an example of an optimisation problem
Bin packing

**Problem** pack all items into the fewest possible bins

This is an example of an optimisation problem
Bin packing

**Problem** pack all items into the fewest possible bins
**Bin packing**

**Problem** pack all items into the fewest possible bins

The **BINPACKING** problem is known to be **NP**-hard.
Bin packing

**Problem** pack all items into the fewest possible bins

The **BINPACKING** problem is known to be **NP-hard**

*and the decision version... “Can you pack the items into at most \( k \) bins?”* is **NP-complete**
The **Bin Packing** problem is known to be **NP-hard** and the decision version... “Can you pack the items into at most $k$ bins?” is **NP-complete**.

**Problem** pack all items into the fewest possible bins
Bin packing

**Problem** pack all items into the fewest possible bins

The **BINPACKING** problem is known to be **NP**-hard
Bin packing

**Problem** pack all items into the fewest possible bins

The **BINPACKING** problem is known to be **NP**-hard

*but fortunately we can approximate*
Next fit

If item $i$ fits into bin $j$: pack it, $i++;$ else $j++;$
Next fit

If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
Next fit

If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
Next fit

If item \( i \) fits into bin \( j \): pack it, \( i++ \); else \( j++ \);
If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
Next fit

If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
Next fit

If item \( i \) fits into bin \( j \): pack it, \( i++ \); else \( j++ \);
Next fit

If item \( i \) fits into bin \( j \): pack it, \( i++ \); else \( j++ \);

1

\[
\begin{array}{c}
\frac{2}{8} \\
\frac{4}{8} \\
\frac{4}{8} \\
\frac{7}{8} \\
\end{array}
\]

1

\[
\begin{array}{c}
\frac{2}{8} \\
\frac{3}{8} \\
\end{array}
\]
Next fit

If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
Next fit

If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
Next fit
Next fit runs in $O(n)$ time but how good is it?
Next fit runs in $O(n)$ time but how good is it?

where $n$ is the number of items
Next fit runs in $O(n)$ time but how good is it?
Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$

and $s$ be the number of non-empty bins (using Next fit)
Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$

and $s$ be the number of non-empty bins (using Next fit)

Observe that $\text{fill}(2i - 1) + \text{fill}(2i) > 1$ (for $1 \leq 2i \leq s$)
Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$

and $s$ be the number of non-empty bins (using Next fit)

Observe that $\text{fill}(2i - 1) + \text{fill}(2i) > 1$ (for $1 \leq 2i \leq s$)

so $\left\lfloor \frac{s}{2} \right\rfloor < \sum_{1 \leq 2i \leq s} \text{fill}(2i - 1) + \text{fill}(2i)$
Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$

and $s$ be the number of non-empty bins (using Next fit)

Observe that $\text{fill}(2i - 1) + \text{fill}(2i) > 1$ (for $1 \leq 2i \leq s$)

so $\left\lfloor \frac{s}{2} \right\rfloor < \sum_{1 \leq 2i \leq s} \text{fill}(2i - 1) + \text{fill}(2i) \leq I$
Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$

and $s$ be the number of non-empty bins (using Next fit)

Observe that $\text{fill}(2i - 1) + \text{fill}(2i) > 1$ (for $1 \leq 2i \leq s$)

so $\left\lfloor \frac{s}{2} \right\rfloor < \sum_{1 \leq 2i \leq s} \text{fill}(2i - 1) + \text{fill}(2i) \leq I$  

the sum of the item weights
Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$

and $s$ be the number of non-empty bins (using Next fit)

Observe that $\text{fill}(2i - 1) + \text{fill}(2i) > 1$ (for $1 \leq 2i \leq s$)

so $\left\lfloor s/2 \right\rfloor < \sum_{1 \leq 2i \leq s} \text{fill}(2i - 1) + \text{fill}(2i) \leq I \leq \text{Opt}$
Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$

and $s$ be the number of non-empty bins (using Next fit)

Observe that $\text{fill}(2i - 1) + \text{fill}(2i) > 1$ (for $1 \leq 2i \leq s$)

so \[
\left\lfloor \frac{s}{2} \right\rfloor < \sum_{1 \leq 2i \leq s} \text{fill}(2i - 1) + \text{fill}(2i) \leq I \leq \text{Opt}
\]
Next fit runs in \(O(n)\) time but how good is it?

Let \(\text{fill}(i)\) be the sum of item sizes in bin \(i\)

and \(s\) be the number of non-empty bins (using Next fit)

Observe that \(\text{fill}(2i - 1) + \text{fill}(2i) > 1\) (for \(1 \leq 2i \leq s\))

so \[ \left\lfloor \frac{s}{2} \right\rfloor < \sum_{1 \leq 2i \leq s} \text{fill}(2i - 1) + \text{fill}(2i) \leq I \leq \text{Opt} \]
Next fit

Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$

and $s$ be the number of non-empty bins (using Next fit)

Observe that $\text{fill}(2i - 1) + \text{fill}(2i) > 1$ (for $1 \leq 2i \leq s$)

so $\left\lfloor \frac{s}{2} \right\rfloor < \sum_{1 \leq 2i \leq s} \text{fill}(2i - 1) + \text{fill}(2i) \leq I \leq \text{Opt}$

therefore $s \leq 2 \cdot \text{Opt}$
Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$ and $s$ be the number of non-empty bins (using Next fit).

Observe that $\text{fill}(2i - 1) + \text{fill}(2i) > 1$ (for $1 \leq 2i \leq s$).

So $\left\lfloor \frac{s}{2} \right\rfloor < \sum_{1 \leq 2i \leq s} \text{fill}(2i - 1) + \text{fill}(2i) \leq I \leq \text{Opt}$

Therefore $s \leq 2 \cdot \text{Opt}$ in other words the Next Fit is never worse than twice the optimal.
Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$

and $s$ be the number of non-empty bins (using Next fit)

Observe that $\text{fill}(2i - 1) + \text{fill}(2i) > 1$ (for $1 \leq 2i \leq s$)

so $\left\lfloor s/2 \right\rfloor < \sum_{1 \leq 2i \leq s} \text{fill}(2i - 1) + \text{fill}(2i) \leq I \leq \text{Opt}$

therefore $s \leq 2 \cdot \text{Opt}$
Approximation Algorithms

An algorithm $A$ is an $\alpha$-approximation algorithm for problem $P$ if,

- $A$ runs in polynomial time
- $A$ always outputs a solution with value $s$ within an $\alpha$ factor of $\text{Opt}$
Approximation Algorithms

An algorithm $A$ is an $\alpha$-approximation algorithm for problem $P$ if,

- $A$ runs in polynomial time
- $A$ always outputs a solution with value $s$ within an $\alpha$ factor of $Opt$

Here $P$ is an optimisation problem with optimal solution of value $Opt$
Approximation Algorithms

An algorithm $A$ is an $\alpha$-approximation algorithm for problem $P$ if,

- $A$ runs in polynomial time
- $A$ always outputs a solution with value $s$
  within an $\alpha$ factor of $\text{Opt}$

Here $P$ is an optimisation problem with optimal solution of value $\text{Opt}$

- If $P$ is a maximisation problem, $\frac{\text{Opt}}{\alpha} \leq s \leq \text{Opt}$
Approximation Algorithms

An algorithm $A$ is an $\alpha$-approximation algorithm for problem $P$ if,

- $A$ runs in polynomial time
- $A$ always outputs a solution with value $s$
  within an $\alpha$ factor of $\text{Opt}$

Here $P$ is an optimisation problem with optimal solution of value $\text{Opt}$

- If $P$ is a maximisation problem, $\frac{\text{Opt}}{\alpha} \leq s \leq \text{Opt}$
- If $P$ is a minimisation problem (like BINPACKING), $\text{Opt} \leq s \leq \alpha \cdot \text{Opt}$
Approximation Algorithms

An algorithm $A$ is an $\alpha$-approximation algorithm for problem $P$ if,

- $A$ runs in polynomial time
- $A$ always outputs a solution with value $s$ within an $\alpha$ factor of $\text{Opt}$

Here $P$ is an optimisation problem with optimal solution of value $\text{Opt}$

- If $P$ is a maximisation problem, $\frac{\text{Opt}}{\alpha} \leq s \leq \text{Opt}$
- If $P$ is a minimisation problem (like BINPACKING), $\text{Opt} \leq s \leq \alpha \cdot \text{Opt}$

We have seen a 2-approximation algorithm for BINPACKING
Approximation Algorithms

An algorithm $A$ is an $\alpha$-approximation algorithm for problem $P$ if,

- $A$ runs in polynomial time
- $A$ always outputs a solution with value $s$ within an $\alpha$ factor of $Opt$

Here $P$ is an optimisation problem with optimal solution of value $Opt$

- If $P$ is a maximisation problem, $\frac{Opt}{\alpha} \leq s \leq Opt$
- If $P$ is a minimisation problem (like BINPACKING), $Opt \leq s \leq \alpha \cdot Opt$

We have seen a 2-approximation algorithm for BINPACKING
the number of bins used, $s$ is always between $Opt$ and $2 \cdot Opt$
Approximation Algorithms

An algorithm $A$ is an $\alpha$-approximation algorithm for problem $P$ if,

- $A$ runs in polynomial time
- $A$ always outputs a solution with value $s$ within an $\alpha$ factor of $\text{Opt}$

Here $P$ is an optimisation problem with optimal solution of value $\text{Opt}$

- If $P$ is a *maximisation* problem, $\frac{\text{Opt}}{\alpha} \leq s \leq \text{Opt}$
- If $P$ is a *minimisation* problem (like BINPACKING), $\text{Opt} \leq s \leq \alpha \cdot \text{Opt}$

We have seen a 2-approximation algorithm for BINPACKING
the number of bins used, $s$ is always between $\text{Opt}$ and $2 \cdot \text{Opt}$

In the examples we consider, $\alpha$ will be a constant but it could depend on $n$ (the input size)
We have seen that Next fit is a $2$-approximation algorithm for Bin packing which runs in $O(n)$ time
can we do better?
First fit decreasing (FFD)

1

4/8  2/8  4/8  7/8  2/8  3/8
First fit decreasing (FFD)

Step 1: Sort the items into non-increasing order
First fit decreasing (FFD)

Step 1: Sort the items into non-increasing order
First fit decreasing (FFD)

Step 2: Put each item in the first \textit{(left-most)} bin it fits in
First fit decreasing (FFD)

Step 2: Put each item in the first (left-most) bin it fits in
First fit decreasing (FFD)

Step 2: Put each item in the first (left-most) bin it fits in
First fit decreasing (FFD)

Step 2: Put each item in the first (left-most) bin it fits in

1

7/8

4/8

4/8

...
First fit decreasing (FFD)

Step 2: Put each item in the first (left-most) bin it fits in

1

\[
\begin{array}{c}
\text{7/8} \\
\text{4/8} \\
\text{4/8} \\
\text{3/8} \\
\text{...}
\end{array}
\]
First fit decreasing (FFD)

Step 2: Put each item in the first (left-most) bin it fits in
First fit decreasing (FFD)

Step 2: Put each item in the first *(left-most)* bin it fits in
First fit decreasing (FFD)

Step 2: Put each item in the first (left-most) bin it fits in.

This will be important for the proof.
First fit decreasing (FFD)

Step 2: Put each item in the first (left-most) bin it fits in.

\[
\begin{align*}
\text{1} & \quad \text{7/8} \\
\text{1} & \quad \text{4/8} \\
\text{1} & \quad \text{2/8} \\
\text{1} & \quad \text{2/8} \\
\text{1} & \quad \text{3/8} \\
\end{align*}
\]
First fit decreasing (FFD)
First fit decreasing (FFD)

FFD runs in $O(n^2)$ time but how good is it?
First fit decreasing (FFD)

FFD runs in $O(n^2)$ time but how good is it?
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)
First fit decreasing (FFD)

Consider bin $j = \lceil \frac{2s}{3} \rceil$ (s is the number of bins FFD uses on this input)
Consider bin \( j = \left\lceil \frac{2s}{3} \right\rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 1: Bin \( j \) contains an item of size \( > \frac{1}{2} \)
Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 1: Bin $j$ contains an item of size $> \frac{1}{2}$
First fit decreasing (FFD)

Consider bin $j = \lceil \frac{2s}{3} \rceil$ ($s$ is the number of bins FFD uses on this input)

Case 1: Bin $j$ contains an item of size $> \frac{1}{2}$

Every bin $j' \leq j$ contains an item of size $> \frac{1}{2}$
First fit decreasing (FFD)

Consider bin $j = \lceil \frac{2s}{3} \rceil$ ($s$ is the number of bins FFD uses on this input)

Case 1: Bin $j$ contains an item of size $> \frac{1}{2}$

Every bin $j' \leq j$ contains an item of size $> \frac{1}{2}$

because we packed big things first and each thing was packed in the lowest numbered bin
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 1: Bin $j$ contains an item of size $> \frac{1}{2}$

Every bin $j' \leq j$ contains an item of size $> \frac{1}{2}$

because we packed big things first and each thing was packed in the lowest numbered bin
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 1: Bin $j$ contains an item of size $> \frac{1}{2}$

Every bin $j' \leq j$ contains an item of size $> \frac{1}{2}$
First fit decreasing (FFD)

Consider bin $j = \lceil \frac{2s}{3} \rceil$ ($s$ is the number of bins FFD uses on this input)

Case 1: Bin $j$ contains an item of size $> \frac{1}{2}$

Every bin $j' \leq j$ contains an item of size $> \frac{1}{2}$

each of these items has to be in a different bin (even in $\text{Opt}$)
First fit decreasing (FFD)

Consider bin \( j = \left\lceil \frac{2s}{3} \right\rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 1: Bin \( j \) contains an item of size \( > \frac{1}{2} \)

Every bin \( j' \leq j \) contains an item of size \( > \frac{1}{2} \)

each of these items has to be in a different bin (even in Opt)

So Opt uses at least \( \frac{2s}{3} \) bins
First fit decreasing (FFD)

Consider bin $j = \lceil \frac{2s}{3} \rceil$ ($s$ is the number of bins FFD uses on this input)

Case 1: Bin $j$ contains an item of size $> \frac{1}{2}$

Every bin $j' \leq j$ contains an item of size $> \frac{1}{2}$

Each of these items has to be in a different bin (even in Opt)

So Opt uses at least $\frac{2s}{3}$ bins

Or... $s \leq \frac{3\text{Opt}}{2}$
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq \frac{1}{2}$

when FFD packed the first item into bin $j$, 
First fit decreasing (FFD)

Consider bin $j = \lceil \frac{2s}{3} \rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

when FFD packed the first item into bin $j$, 

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ (s is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq \frac{1}{2}$

when FFD packed the first item into bin $j$,
First fit decreasing (FFD)

Consider bin \( j = \left\lceil \frac{2s}{3} \right\rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 2: Bin \( j \) contains only items of size \( \leq \frac{1}{2} \)

when FFD packed the first item into bin \( j \),

1. all bins \( j, (j + 1), \ldots, (s - 2), (s - 1) \) were empty
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

when FFD packed the first item into bin $j$,

1. all bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ were empty

2. and all unpacked items had size $\leq 1/2$

(because we pack in non-increasing order)
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input).

Case 2: Bin $j$ contains only items of size $\leq 1/2$

when FFD packed the first item into bin $j$,

1. all bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ were empty

2. and all unpacked items had size $\leq 1/2$

(because we pack in non-increasing order)

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items
First fit decreasing (FFD)

Consider bin \( j = \left\lceil \frac{2s}{3} \right\rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 2: Bin \( j \) contains only items of size \( \leq 1/2 \)

when FFD packed the first item into bin \( j \),

1. all bins \( j, (j + 1), \ldots, (s - 2), (s - 1) \) were empty

2. and all unpacked items had size \( \leq 1/2 \) 
   (because we pack in non-increasing order)

so Bins \( j, (j + 1), \ldots, (s - 2), (s - 1) \) each contain at least two items
Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq \frac{1}{2}$ when FFD packed the first item into bin $j$,

1. all bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ were empty
2. and all unpacked items had size $\leq \frac{1}{2}$

(because we pack in non-increasing order)

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items

(we only use a new bin when the item won’t fit in any previous bin)
Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq \frac{1}{2}$

when FFD packed the first item into bin $j$,

1. all bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ were empty

2. and all unpacked items had size $\leq \frac{1}{2}$
   (because we pack in non-increasing order)

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ (s is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

when FFD packed the first item into bin $j$,

1. all bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ were empty
2. and all unpacked items had size $\leq 1/2$

(because we pack in non-increasing order)

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items
and bin $s$ contains at least one item
First fit decreasing (FFD)

Consider bin \( j = \left\lceil \frac{2s}{3} \right\rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 2: Bin \( j \) contains only items of size \( \leq \frac{1}{2} \)

when FFD packed the first item into bin \( j \),

1. all bins \( j, (j + 1), \ldots, (s - 2), (s - 1) \) were empty

2. and all unpacked items had size \( \leq \frac{1}{2} \)

(because we pack in non-increasing order)

so Bins \( j, (j + 1), \ldots, (s - 2), (s - 1) \) each contain at least two items

and bin \( s \) contains at least one item
First fit decreasing (FFD)

Consider bin \( j = \left\lceil \frac{2s}{3} \right\rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 2: Bin \( j \) contains only items of size \( \leq \frac{1}{2} \)

so Bins \( j, (j + 1), \ldots, (s - 2), (s - 1) \) each contain at least two items and bin \( s \) contains at least one item
First fit decreasing (FFD)

Consider bin \( j = \lceil \frac{2s}{3} \rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 2: Bin \( j \) contains only items of size \( \leq 1/2 \)
Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ (s is the number of bins FFD uses on this input)

**Case 2:** Bin $j$ contains only items of size $\leq 1/2$

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items

and bin $s$ contains at least one item
Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items
and bin $s$ contains at least one item

This gives a total of $2(s - j) + 1$ items, none of which fits into bins $1, 2, 3, \ldots, (j - 1)$
Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items and bin $s$ contains at least one item

This gives a total of $2(s - j) + 1$ items, none of which fits into bins $1, 2, 3, \ldots, (j - 1)$ otherwise we would have packed them there
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items
and bin $s$ contains at least one item

This gives a total of $2(s - j) + 1$ items, none of which fits into bins $1, 2, 3, \ldots, (j - 1)$
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items
and bin $s$ contains at least one item

This gives a total of $2(s - j) + 1$ items, none of which fits into bins $1, 2, 3, \ldots, (j - 1)$

so $I > \min\{j - 1, 2(s - j) + 1\}$
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items
and bin $s$ contains at least one item

This gives a total of $2(s - j) + 1$ items, none of which fits into bins $1, 2, 3, \ldots, (j - 1)$

so $I > \min\{j - 1, 2(s - j) + 1\}$

recall $I$ is the total weight of all items
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input).

Case 2: Bin $j$ contains only items of size $\leq 1/2$

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items

and bin $s$ contains at least one item

This gives a total of $2(s - j) + 1$ items, none of which fits into bins $1, 2, 3, \ldots, (j - 1)$

so $I > \min\{j - 1, 2(s - j) + 1\}$

recall $I$ is the total weight of all items
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items and bin $s$ contains at least one item

This gives a total of $2(s - j) + 1$ items, none of which fits into bins $1, 2, 3, \ldots, (j - 1)$

so $I > \min\{j - 1, 2(s - j) + 1\}$

recall $I$ is the total weight of all items
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

**Case 2:** Bin $j$ contains only items of size $\leq 1/2$

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items

and bin $s$ contains at least one item

This gives a total of $2(s - j) + 1$ items, none of which fits into bins $1, 2, 3, \ldots, (j - 1)$

so $I > \min\{j - 1, 2(s - j) + 1\}$

recall $I$ is the total weight of all items
First fit decreasing (FFD)

Consider bin \( j = \left\lceil \frac{2s}{3} \right\rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 2: Bin \( j \) contains only items of size \( \leq \frac{1}{2} \)

so Bins \( j, (j + 1), \ldots, (s - 2), (s - 1) \) each contain at least two items
and bin \( s \) contains at least one item

This gives a total of \( 2(s - j) + 1 \) items, none of which fits into bins \( 1, 2, 3, \ldots, (j - 1) \)

so \( I > \min\{j - 1, 2(s - j) + 1\} \)
Consider bin \( j = \left\lceil \frac{2s}{3} \right\rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 2: Bin \( j \) contains only items of size \( \leq 1/2 \)

so Bins \( j, (j + 1), \ldots, (s - 2), (s - 1) \) each contain at least two items

and bin \( s \) contains at least one item

This gives a total of \( 2(s - j) + 1 \) items, none of which fits into bins \( 1, 2, 3, \ldots, (j - 1) \)

so \( I > \min\{j - 1, 2(s - j) + 1\} \geq \left\lceil 2s/3 \right\rceil - 1 \)
First fit decreasing (FFD)

Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

so Bins $j, (j + 1), \ldots, (s - 2), (s - 1)$ each contain at least two items

and bin $s$ contains at least one item

This gives a total of $2(s - j) + 1$ items, none of which fits into bins $1, 2, 3, \ldots, (j - 1)$

so $I > \min\{j - 1, 2(s - j) + 1\} \geq \left\lceil \frac{2s}{3} \right\rceil - 1$

by plugging in $j = \left\lceil \frac{2s}{3} \right\rceil$
First fit decreasing (FFD)

Consider bin \( j = \lceil \frac{2s}{3} \rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 2: Bin \( j \) contains only items of size \( \leq 1/2 \)

As \( \lceil 2s/3 \rceil - 1 < I \)
Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input)

Case 2: Bin $j$ contains only items of size $\leq 1/2$

As $\left\lceil \frac{2s}{3} \right\rceil - 1 < I$ and $I \leq \text{Opt}$
Consider bin $j = \left\lceil \frac{2s}{3} \right\rceil$ ($s$ is the number of bins FFD uses on this input).

Case 2: Bin $j$ contains only items of size $\leq 1/2$

As $\left\lceil \frac{2s}{3} \right\rceil - 1 < I$ and $I \leq \text{Opt}$

we have that $\left\lceil \frac{2s}{3} \right\rceil - 1 < \text{Opt}$
First fit decreasing (FFD)

Consider bin \( j = \lceil \frac{2s}{3} \rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 2: Bin \( j \) contains only items of size \( \leq 1/2 \)

As \( \lceil \frac{2s}{3} \rceil - 1 < I \) and \( I \leq \text{Opt} \)

we have that \( \lceil \frac{2s}{3} \rceil - 1 < \text{Opt} \)

... but both sides are integers...

so \( \lceil \frac{2s}{3} \rceil \leq \text{Opt} \)

finally ... \( 2s/3 \leq \lceil 2s/3 \rceil \leq \text{Opt} \)

or \( s \leq (3/2)\text{Opt} \)
First fit decreasing (FFD)

Consider bin \( j = \left\lceil \frac{2s}{3} \right\rceil \) (\( s \) is the number of bins FFD uses on this input)
Consider bin \( j = \left\lceil \frac{2s}{3} \right\rceil \) (\( s \) is the number of bins FFD uses on this input)

Case 1: Bin \( j \) contains an item of size \( > \frac{1}{2} \)

Case 2: Bin \( j \) contains only items of size \( \leq \frac{1}{2} \)

in both cases… \( s \leq \frac{3\text{Opt}}{2} \)
First fit decreasing (FFD)

Consider bin $j = \lceil \frac{2s}{3} \rceil$ ($s$ is the number of bins FFD uses on this input)

Case 1: Bin $j$ contains an item of size $> \frac{1}{2}$

Case 2: Bin $j$ contains only items of size $\leq \frac{1}{2}$

in both cases... $s \leq \frac{3\text{Opt}}{2}$

So FFD is a $3/2$-approximation algorithm for BINPACKING
Approximation Algorithms Summary

An algorithm $A$ is an $\alpha$-approximation algorithm for problem $P$ if,

- $A$ runs in polynomial time
- $A$ always outputs a solution with value $s$ within an $\alpha$ factor of $\text{Opt}$

Here $P$ is an optimisation problem with optimal solution of value $\text{Opt}$

If $P$ is a maximisation problem, $\frac{\text{Opt}}{\alpha} \leq s \leq \text{Opt}$

If $P$ is a minimisation problem (like BINPACKING), $\text{Opt} \leq s \leq \alpha \cdot \text{Opt}$

We have seen Next Fit which is a $2$-approximation algorithm for BINPACKING

which runs in $O(n)$ time

and First Fit Decreasing which is a $3/2$-approximation algorithm for BINPACKING

which runs in $O(n^2)$ time

Bin Packing is NP-hard so solving it exactly in polynomial time would prove that $P = \text{NP}$