Review & Plan
Today’s objectives

- Review of list-update with a focus on advice
- An introduction to bin packing
How many bits of advice are sufficient to achieve an optimal solution for a sequence $\sigma$ of length $n$?
How many bits of advice are sufficient to achieve an optimal solution for a sequence $\sigma$ of length $n$?

- We show $\text{OPT}(\sigma) - n$ bits are sufficient.
We learned earlier that an optimal algorithm only needs to make paid exchanges.

- We can be more specific.
We learned earlier that an optimal algorithm only needs to make paid exchanges.

We can be more specific.

There is an optimal solution which only uses *subset transfers*.

Before accessing an item $x$, use paid exchanges to move a subset of preceding items to just after $x$ (we skip the proof).
We learned earlier that an optimal algorithm only needs to make paid exchanges.

- We can be more specific.

There is an optimal solution which only uses subset transfers.

- Before accessing an item \( x \), use paid exchanges to move a subset of preceding items to just after \( x \) (we skip the proof).
We learned earlier that an optimal algorithm only needs to make paid exchanges

- We can be more specific.

There is an optimal solution which only uses subset transfers

- Before accessing an item \( x \), use paid exchanges to move a subset of preceding items to just after \( x \) (we skip the proof).
We learned earlier that an optimal algorithm only needs to make paid exchanges.

We can be more specific.

There is an optimal solution which only uses subset transfers.

Before accessing an item $x$, use paid exchanges to move a subset of preceding items to just after $x$ (we skip the proof).

Guide an algorithm to maintain the same lists as $\text{OPT}$ by encoding the subset to be transferred after each access.
What is the advice? Before each access to an item $x$ at index $i$, we use one bit of advice for each item $y$ preceding $x$ to indicate whether $y$ should be transferred to after $x$. 

The size of advice: Opt incurs a cost of at least $i$ for accessing $x$ (it might be more). There will be $i - 1$ bits of advice. The total advice size is no more than the cost of $\text{cost}(\text{Opt}) - n$. For a list of length $m$, we have $\text{cost}(\text{Opt}) - n > n \cdot m / 2$ (static offline), so for lists of constant length $\text{cost}(\text{Opt}) \in O(n)$. 

How the algorithm uses advice: For each request to item $x$, it transfers the indicated subsets in the advice to right after $x$. 

Why it is optimal: the algorithm mimic Opt.
Optimal Solution with Advice

- **What is the advice?** Before each access to an item $x$ at index $i$, we use one bit of advice for each item $y$ preceding $x$ to indicate whether $y$ should be transferred to after $x$.

- **What is the size of advice?** $\text{OPT}$ incurs a cost of at least $i$ for accessing $x$ (it might be more).
  - There will be $i - 1$ bits of advice.
  - The total advice size is no more than the cost of $\text{cost}(\text{OPT}) - n$.
  - For a list of length $m$, we have $\text{cost}(\text{OPT}) - n > n \cdot m/2$ (static offline), so for lists of constant length $\text{cost}(\text{OPT}) \in O(n)$. 

Theorem

For lists of constant length, advice of size $O(n)$ bits is sufficient to achieve an optimal algorithm for any sequence of length $n$. 

Optimal Solution with Advice

- **What is the advice?** Before each access to an item $x$ at index $i$, we use one bit of advice for each item $y$ preceding $x$ to indicate whether $y$ should be transferred to after $x$.

- **What is the size of advice?** $\text{OPT}$ incurs a cost of at least $i$ for accessing $x$ (it might be more).
  - There will be $i - 1$ bits of advice.
  - The total advice size is no more than the cost of $\text{cost}(\text{OPT}) - n$.
  - For a list of length $m$, we have $\text{cost}(\text{OPT}) - n > n \cdot m/2$ (static offline), so for lists of constant length $\text{cost}(\text{OPT}) \in O(n)$.

- **How the algorithm uses advice?** For each request to item $x$, it transfers the indicated subsets in the advice to right after $x$. 

- **Why it is optimal?** The algorithm mimics $\text{OPT}$. 

- **Theorem** For lists of constant length, advice of size $O(n)$ bits is sufficient to achieve an optimal algorithm for any sequence of length $n$. 

Optimal Solution with Advice

- **What is the advice?** Before each access to an item \( x \) at index \( i \), we use one bit of advice for each item \( y \) preceding \( x \) to indicate whether \( y \) should be transferred to after \( x \).

- **What is the size of advice?** \( \text{OPT} \) incurs a cost of at least \( i \) for accessing \( x \) (it might be more).
  - There will be \( i - 1 \) bits of advice.
  - The total advice size is no more than the cost of \( \text{cost}(\text{OPT}) - n \).
  - For a list of length \( m \), we have \( \text{cost}(\text{OPT}) - n > n \cdot m/2 \) (static offline), so for lists of constant length \( \text{cost}(\text{OPT}) \in O(n) \).

- **How the algorithm uses advice?** For each request to item \( x \), it transfers the indicated subsets in the advice to right after \( x \).

- **Why it is optimal?** the algorithm mimic \( \text{OPT} \).

**Theorem**

*For lists of constant length, advice of size \( O(n) \) bits is sufficient to achieve an optimal algorithm for any sequence of length \( n \).*
How many bits of advice are required to achieve an optimal solution for a sequence $\sigma$ of length $n$?
How many bits of advice are required to achieve an optimal solution for a sequence $\sigma$ of length $n$?

- We show advice of linear size is required.
Consider a sequence of phases of requests to items $a$, $b$ which form a list of length 2.

- Each phase involves 6 requests and has type 0 or 1.
- A type 0 phase has requests $\rightarrow bbbaaa$ and a type 1 phase has requests $baabaa$.

\[
\begin{array}{cccccc}
\text{bbbaaa} & \text{baabaa} & \text{bbbaaa} & \text{bbbaaa} & \text{baabaa} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
0 & 1 & 0 & 0 & 1
\end{array}
\]
Consider a sequence of phases of requests to items $a$, $b$ which form a list of length 2.

- Each phase involves 6 requests and has type 0 or 1.
- A type 0 phase has requests $\rightarrow bbbaaa$ and a type 1 phase has requests $baabaa$.

<bbbaaa baabaa bbbaaa bbbaaa baabaa>

\[
\begin{array}{cccccc}
0 & 1 & 0 & 0 & 1
\end{array}
\]

At the beginning of each phase, the list of any reasonable algorithm looks like $a \rightarrow b$

- Otherwise, the algorithm can be improved to have that without increasing its cost.
Consider a sequence of phases of requests to items $a$, $b$ which form a list of length 2.

- Each phase involves 6 requests and has type 0 or 1.
- A type 0 phase has requests $\rightarrow bbbaaa$ and a type 1 phase has requests $baabaa$.

$< bbbaaa \ baabaa \ bbbaaa \ bbbaaa \ baabaa >$

- At the beginning of each phase, the list of any reasonable algorithm looks like $a \rightarrow b$
  - Otherwise, the algorithm can be improved to have that without increasing its cost.

- At the first request to $b$ in each phase, depending on its type, an optimal algorithm should move $b$ to front (type 0) or keep it at second position (type 1).
Consider a sequence of phases of requests to items $a$, $b$ which form a list of length 2.

- Each phase involves 6 requests and has type 0 or 1.
- A type 0 phase has requests $\rightarrow bbbaaa$ and a type 1 phase has requests $baabaa$.

$$< bbbaaa \ bbaabaa \ bbbaaa \ bbbaaa \ baabaa >$$

- At the beginning of each phase, the list of any reasonable algorithm looks like $a \rightarrow b$
- Otherwise, the algorithm can be improved to have that without increasing its cost.

- At the first request to $b$ in each phase, depending on its type, an optimal algorithm should move $b$ to front (type 0) or keep it at second position (type 1).
- The algorithm should guess the type of each phase at the first request.
Optimal Solution with Advice

\[< bbbaaa \ baabaa \ bbbaaa \ bbbaaa \ bbaabaa >\]

\[
\begin{array}{cccccc}
0 & 1 & 0 & 0 & 1 & 1 \\
\end{array}
\]

The cost of Opt is 2 + 1 + 1 + 2 + 1 + 1 = 8.

The cost of Alg is 8 if it guesses the type correctly, and \( \geq 9 \) otherwise.

Assume there are \( m \) phases and the algorithm correctly guesses half of them:

From the binary guessing lemma, we know that requires advice of size \( \Omega(m) = \Omega(n) \).

The cost of the algorithm will be at least

\[
\left( \frac{k}{2} \right) \cdot 8 + \left( \frac{k}{2} \right) \cdot 9 = 8.5k
\]

The cost of Opt will be \( k \cdot 8 \).

The competitive ratio will be \( \frac{8.5}{8} = 1.75 \).
phases of type 0 (bbbaaa):
- The cost of OPT is $2+1+1+2+1+1 = 8$.

phases of type 1 (baabaa):
- The cost of OPT is $2+1+1+2+1+1 = 8$. 
Optimal Solution with Advice

< <\text{bbbaaa} \text{baabaa} \text{bbbaaa} \text{bbbaaa} \text{baabaa}> \downarrow \downarrow \downarrow \downarrow \downarrow
0 \quad 1 \quad 0 \quad 0 \quad 1

- **phases of type 0 (\text{bbbaaa}):**
  - The cost of OPT is $2+1+1+2+1+1 = 8$.
  - The cost of Alg is 8 if it guesses the type correctly, and $\geq 9$ otherwise

- **phases of type 1 (\text{baabaa}):**
  - The cost of OPT is $2+1+1+2+1+1 = 8$.
  - The cost of Alg is 8 if it guesses the type correctly, and $\geq 9$ otherwise
Optimal Solution with Advice

\(< \text{bbbaaa baabaa bbbaaa bbbaaa baabaa} >\)

- phases of type 0 (\text{bbbaaa}): 
  - The cost of OPT is $2+1+1+2+1+1 = 8$.
  - The cost of Alg is 8 if it guesses the type correctly, and $\geq 9$ otherwise

- phases of type 1 (\text{baabaa}): 
  - The cost of OPT is $2+1+1+2+1+1 = 8$.
  - The cost of Alg is 8 if it guesses the type correctly, and $\geq 9$ otherwise

Assume there are $m$ phases and the algorithm correctly guesses half of them:
- From binary guessing lemma, we know that requires advice of size $\Omega(m) = \Omega(n)$
Review & Plan

Optimal Solution with Advice

\[ < bbbaaa \ baabaa \ bbbaaa \ bbbaaa \ baabaa > \]

↓     ↓     ↓     ↓     ↓     ↓
0     1     0     0     0     1

- phases of type 0 (bbbaaa):
  - The cost of OPT is \(2+1+1+2+1+1 = 8\).
  - The cost of Alg is 8 if it guesses the type correctly, and \(\geq 9\) otherwise

- phases of type 1 (baabaa):
  - The cost of OPT is \(2+1+1+2+1+1 = 8\).
  - The cost of Alg is 8 if it guesses the type correctly, and \(\geq 9\) otherwise

Assume there are \(m\) phases and the algorithm correctly guesses half of them:

- From binary guessing lemma, we know that requires advice of size \(\Omega(m) = \Omega(n)\)
- the cost of the algorithm will be at least \((k/2) \cdot 8 + (k/2) \cdot 9 = 8.5k\)
Optimal Solution with Advice

\(< \text{bbbbaa} \text{a baabaa bbbbaa bbbbaa baabaa} >\)

- **phases of type 0 (bbbbaa):**
  - The cost of OPT is \(2 + 1 + 1 + 2 + 1 + 1 = 8\).
  - The cost of Alg is 8 if it guesses the type correctly, and \(\geq 9\) otherwise

- **phases of type 1 (baabaa):**
  - The cost of OPT is \(2 + 1 + 1 + 2 + 1 + 1 = 8\).
  - The cost of Alg is 8 if it guesses the type correctly, and \(\geq 9\) otherwise

- Assume there are \(m\) phases and the algorithm correctly guesses half of them:
  - From binary guessing lemma, we know that requires advice of size \(\Omega(m) = \Omega(n)\)
  - the cost of the algorithm will be at least \((k/2) \cdot 8 + (k/2) \cdot 9 = 8.5k\)
  - the cost of OPT will be \(k \cdot 8\).
## Optimal Solution with Advice

Consider the sequence of phases: `<bbbaaa baabaa bbbaaa bbbaaa baabaa>`

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- **phases of type 0 (bbbaaa):**
  - The cost of $\text{OPT}$ is $2+1+1+2+1+1 = 8$.
  - The cost of $\text{Alg}$ is $8$ if it guesses the type correctly, and $\geq 9$ otherwise.

- **phases of type 1 (baabaa):**
  - The cost of $\text{OPT}$ is $2+1+1+2+1+1 = 8$.
  - The cost of $\text{Alg}$ is $8$ if it guesses the type correctly, and $\geq 9$ otherwise.

Assume there are $m$ phases and the algorithm correctly guesses half of them:

- From binary guessing lemma, we know that requires advice of size $\Omega(m) = \Omega(n)$
- the cost of the algorithm will be at least $(k/2) \cdot 8 + (k/2) \cdot 9 = 8.5k$
- the cost of $\text{OPT}$ will be $k \cdot 8$.
- the competitive ratio will be $\frac{8.5}{k} = 17/16$. 

---

### COMP 7720 - Online Algorithms
List Update with Advice & Bin Packing
We showed that, in order to achieve a competitive ratio better than 17/16, advice of size $\Omega(n)$ is required.
We showed that, in order to achieve a competitive ratio better than 17/16, advice of size $\Omega(n)$ is required.

**Theorem**

*For lists of small sizes, advice of linear size is required and sufficient to achieve an optimal solution.*
Bin Packing Problem
Introduction (Bin Packing)

Bin Packing Problem

- The input is a multi-set of items of various sizes in range $(0,1]$.
- The goal is to pack these items into a minimum number of bins of uniform capacity.

E.g., $S =$

$$\{0.1, 0.2, 0.2, 0.3, 0.3, 0.4, 0.4, 0.5, 0.5, 0.5, 0.6, 0.8, 0.8, 0.9\}$$
Introduction (Bin Packing)

**Bin Packing Problem**

- The input is a multi-set of items of various sizes in range \((0,1]\).
- The goal is to pack these items into a minimum number of bins of uniform capacity.
  - E.g., \(S = \{0.1, 0.2, 0.2, 0.3, 0.3, 0.4, 0.4, 0.4, 0.5, 0.5, 0.5, 0.6, 0.8, 0.8, 0.9\}\)
- In the online setting, items arrive form a sequence and arrive one by one
  - An online algorithm has to place each item into a bin without prior knowledge about forthcoming items.
Bin Packing Application

- Bins can be servers (with uniform capacity in terms of memory, bandwidth, cpu, etc.) & items can be jobs/files/data assigned to servers.
Introdution (Bin Packing)

Bin Packing Application

- Bins can be servers (with uniform capacity in terms of memory, bandwidth, cpu, etc.) & items can be jobs/files/data assigned to servers.

- Cutting stock: bins can be ‘cakes’ and ‘items’ can be kids with different amount of requests.
  - or ‘bottles of water’ with different amount of water requests.
**Bin Packing Application**

- Bins can be servers (with uniform capacity in terms of memory, bandwidth, cpu, etc.) & items can be jobs/files/data assigned to servers.

- Cutting stock: bins can be ‘cakes’ and ‘items’ can be kids with different amount of requests.
  - or ‘bottles of water’ with different amount of water requests

- Bins can be trucks of uniform weight capacity and items can be commodities of different weights to be moved between two cities
Next Fit Algorithm

Next Fit: Maintain one *open* bin at any given time.
- Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.
Next Fit Algorithm

- Next Fit: Maintain one *open* bin at any given time.
  - Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

- Next Fit: Maintain one *open* bin at any given time.
  - Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

- **Next Fit:** Maintain one *open* bin at any given time.
  - Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

Next Fit: Maintain one *open* bin at any given time.
- Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

- Next Fit: Maintain one *open* bin at any given time.
  - Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

\[
< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 >
\]
Next Fit Algorithm

- Next Fit: Maintain one *open* bin at any given time.
  - Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

- **Next Fit:** Maintain one *open* bin at any given time.
  - Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

- Next Fit: Maintain one *open* bin at any given time.
  - Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

- **Next Fit**: Maintain one *open* bin at any given time.
  - Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

- Next Fit: Maintain one *open* bin at any given time.
  - Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

Next Fit: Maintain one *open* bin at any given time.

- Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
**Next Fit Algorithm**

Next Fit: Maintain one *open* bin at any given time.

- Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
**Next Fit Algorithm**

- **Next Fit**: Maintain one *open* bin at any given time.
  - Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

Next Fit: Maintain one *open* bin at any given time.
- Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit: Maintain one *open* bin at any given time.

Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

Next Fit: Maintain one open bin at any given time.

- Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
Next Fit Algorithm

Next Fit: Maintain one *open* bin at any given time.

- Place an incoming item in the open bin if there is space there; otherwise close the bin and open a new bin.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 >
First Fit Algorithm

- First Fit: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.
First Fit Algorithm

- First Fit: place an incoming item in the first bin which has enough space for the item.
  - Open a new bin if such bin does not exist.

\[
<0.9 \; 0.3 \; 0.8 \; 0.5 \; 0.1 \; 0.1 \; 0.3 \; 0.2 \; 0.4 \; 0.2 \; 0.4 \; 0.5 \; 0.5 \; 0.8 \; 0.6 \; 0.4 \; 0.5 \ldots >
\]
First Fit: place an incoming item in the first bin which has enough space for the item.

- Open a new bin if such bin does not exist.
First Fit Algorithm

First Fit: place an incoming item in the first bin which has enough space for the item.

- Open a new bin if such bin does not exist.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 ... >

\[
\begin{array}{c}
0.9 \\
0.3
\end{array}
\]
First Fit Algorithm

- First Fit: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 ... >

```
<table>
<thead>
<tr>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
</tr>
</tbody>
</table>
```
```
| 0.8 |
```

```
First Fit Algorithm

- First Fit: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 ... >
First Fit Algorithm

First Fit: place an incoming item in the first bin which has enough space for the item.

Open a new bin if such bin does not exist.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 ... >
First Fit Algorithm

- First Fit: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 ... >
Introduction (Bin Packing)

First Fit Algorithm

- First Fit: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.

< 0.9  0.3  0.8  0.5  0.1  0.1  0.3  0.2  0.4  0.2  0.4  0.5  0.5  0.8  0.6  0.4  0.5 ... >

```
+-------------------+-------------------+-------------------+
|   0.9             |      0.5          |      0.8          |
|  0.1              |      0.1          |                  |
|  0.3              |                  |  0.3              |
+-------------------+-------------------+-------------------+
```
First Fit Algorithm

- First Fit: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.

< 0.9  0.3  0.8  0.5  0.1  0.1  0.3  0.2  0.4  0.2  0.4  0.5  0.5  0.8  0.6  0.4  0.5 ... >
First Fit Algorithm

- First Fit: place an incoming item in the first bin which has enough space for the item.
  - Open a new bin if such bin does not exist.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 ... >
**First Fit Algorithm**

- **First Fit**: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 ... >
First Fit Algorithm

First Fit: place an incoming item in the first bin which has enough space for the item.

Open a new bin if such bin does not exist.
**First Fit Algorithm**

- First Fit: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 ... >

<table>
<thead>
<tr>
<th>0.1</th>
<th>0.1</th>
<th>0.2</th>
<th>0.2</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>0.5</td>
<td>0.8</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
</tbody>
</table>
First Fit Algorithm

- First Fit: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.
First Fit Algorithm

- First Fit: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 ... >

![Diagram of First Fit Algorithm](image)
First Fit Algorithm

- **First Fit**: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.

< 0.9  0.3  0.8  0.5  0.1  0.1  0.3  0.2  0.4  0.2  0.4  0.5  0.5  0.8  0.6  0.4  0.5 ... >

![Diagram showing the First Fit algorithm with items of different sizes being placed into bins.](attachment:image.png)
First Fit: place an incoming item in the first bin which has enough space for the item.

- Open a new bin if such bin does not exist.

< 0.9 0.3 0.8 0.5 0.1 0.1 0.3 0.2 0.4 0.2 0.4 0.5 0.5 0.8 0.6 0.4 0.5 ... >
First Fit Algorithm

- **First Fit**: place an incoming item in the first bin which has enough space for the item.
- Open a new bin if such bin does not exist.

![Diagram showing First Fit Algorithm with item sizes and bin capacities]
Best Fit Algorithm

Place an incoming item in the bin with the highest level (used space) which has enough space for the item.
Best Fit Algorithm

Place an incoming item in the bin with the highest level (used space) which has enough space for the item.
Best Fit Algorithm

Place an incoming item in the bin with the highest level (used space) which has enough space for the item.
Best Fit Algorithm

Place an incoming item in the bin with the highest level (used space) which has enough space for the item.
Best Fit Algorithm

Place an incoming item in the bin with the highest level (used space) which has enough space for the item.
Harmonic Algorithm

- Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).
- Place members of each class separately from others.
Harmonic Algorithm

Harmonic Algorithm classes: $(\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]$.

Place members of each class separately from others.

$$Harmonic \quad K = 4$$

$$< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ldots >$$

$$x > \frac{1}{2} \quad \quad \frac{1}{3} < x \leq \frac{1}{2} \quad \quad \frac{1}{4} < x \leq \frac{1}{3} \quad \quad x \leq \frac{1}{4}$$
Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).

- Place members of each class separately from others.

\[\text{Harmonic} \quad K = 4\]

\[< 0.9 \quad 0.3 \quad 0.8 \quad 0.5 \quad 0.1 \quad 0.1 \quad 0.3 \quad 0.2 \quad 0.4 \quad 0.2 \quad 0.4 \quad 0.5 \quad 0.5 \quad 0.8 \quad 0.6 \quad 0.4 \quad 0.5 \quad \ldots >\]

- \(x > \frac{1}{2}\)
- \(\frac{1}{3} < x \leq \frac{1}{2}\)
- \(\frac{1}{4} < x \leq \frac{1}{3}\)
- \(x \leq \frac{1}{4}\)
Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).

Place members of each class separately from others.

Harmonic \(K = 4\)

\[
\begin{align*}
< 0.9 & 0.3 & 0.8 & 0.5 & 0.1 & 0.1 & 0.3 & 0.2 & 0.4 & 0.2 & 0.4 & 0.5 & 0.5 & 0.8 & 0.6 & 0.4 & 0.5 & \ldots > \\
\end{align*}
\]

- \(x > \frac{1}{2}\)
- \(\frac{1}{3} < x \leq \frac{1}{2}\)
- \(\frac{1}{4} < x \leq \frac{1}{3}\)
- \(x \leq \frac{1}{4}\)
Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).

Place members of each class separately from others.

\[
\text{Harmonic} \quad K = 4
\]

\[
< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ \ldots >
\]
Harmonic Algorithm

- Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).
- Place members of each class separately from others.

\[
\text{Harmonic } \quad K = 4
\]

\[
< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ \ldots >
\]
Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}].\)

- Place members of each class separately from others.

\[
\text{Harmonic} \quad K = 4
\]

\[
< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ldots >
\]

\[
\begin{align*}
&\text{0.9} &\text{0.8} &\text{0.5} &\text{0.3} &\text{0.2} &\text{0.4} &\text{0.5} &\text{0.8} &\text{0.6} &\text{0.4} &\text{0.5} \\
&x > \frac{1}{2} &\frac{1}{3} < x \leq \frac{1}{2} &\frac{1}{4} < x \leq \frac{1}{3} &x \leq \frac{1}{4}
\end{align*}
\]
Harmonic Algorithm

- Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).
- Place members of each class separately from others.

Harmonic  \( K = 4 \)

\(< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.4 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ ... >\)
**Harmonic Algorithm**

- Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).
- Place members of each class separately from others.

\[\text{Harmonic} \quad K = 4\]

\(< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ldots >\]
Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).

Place members of each class separately from others.

**Harmonic** \(K = 4\)

\(< 0.9, 0.3, 0.8, 0.5, 0.1, 0.1, 0.3, 0.2, 0.4, 0.2, 0.4, 0.5, 0.5, 0.8, 0.6, 0.4, 0.5 \ldots >\)
Harmonic Algorithm

Harmonic Algorithm classes: $(\frac{1}{2}, 1]$, $(\frac{1}{3}, \frac{1}{2}]$, $\ldots$, $(\frac{1}{K}, \frac{1}{K-1}]$, $(0, \frac{1}{K}]$.

- Place members of each class separately from others.

For $K = 4$:

\[
< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ ... >
\]
Harmonic Algorithm classes: \( \left( \frac{1}{2}, 1 \right], \left( \frac{1}{3}, \frac{1}{2} \right], \ldots, \left( \frac{1}{K}, \frac{1}{K-1} \right], (0, \frac{1}{K}] \).

Place members of each class separately from others.

Harmonic Algorithm

\( K = 4 \)

\(< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ldots >\)
Harmonic Algorithm classes: \( \left( \frac{1}{2}, 1 \right], \left( \frac{1}{3}, \frac{1}{2} \right], \ldots, \left( \frac{1}{K}, \frac{1}{K-1} \right], (0, \frac{1}{K}] \).

Place members of each class separately from others.

Harmonic Algorithm

\[ K = 4 \]

\(< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ \ldots > \]
Harmonic Algorithm

Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).

- Place members of each class separately from others.

Harmonic

\[ K = 4 \]

\(< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ldots >\]
Harmonic Algorithm

- Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).
- Place members of each class separately from others.

\[\text{Harmonic } K = 4\]

\(<0.9, 0.3, 0.8, 0.5, 0.1, 0.1, 0.3, 0.2, 0.4, 0.2, 0.4, 0.5, 0.5, 0.8, 0.6, 0.4, 0.5 \ldots >\]

\(x > \frac{1}{2}\)

\(\frac{1}{3} < x \leq \frac{1}{2}\)

\(\frac{1}{4} < x \leq \frac{1}{3}\)

\(x \leq \frac{1}{4}\)
Harmonic Algorithm

- Harmonic Algorithm classes: \( \left( \frac{1}{2}, 1 \right], \left( \frac{1}{3}, \frac{1}{2} \right], \ldots, \left( \frac{1}{K}, \frac{1}{K-1} \right], (0, \frac{1}{K}] \).
- Place members of each class separately from others.

\[ \text{Harmonic} \quad K = 4 \]

\[ < 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ ... > \]
Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).

- Place members of each class separately from others.

For \(K = 4\):

\[
\begin{align*}
&< 0.9, 0.3, 0.8, 0.5, 0.1, 0.1, 0.3, 0.2, 0.4, 0.2, 0.4, 0.5, 0.5, 0.8, 0.6, 0.4, 0.5 \ldots > \\
&x > \frac{1}{2} \\
&\frac{1}{3} < x \leq \frac{1}{2} \\
&\frac{1}{4} < x \leq \frac{1}{3} \\
&x \leq \frac{1}{4}
\end{align*}
\]
Harmonic Algorithm

- Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K})\).
- Place members of each class separately from others.

For \(K = 4\):

\[
< 0.9 \ 0.3 \ 0.8 \ 0.5 \ 0.1 \ 0.1 \ 0.3 \ 0.2 \ 0.4 \ 0.2 \ 0.4 \ 0.5 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.5 \ldots >
\]

\[
\begin{array}{c|c|c|c|c}
0.9 & 0.8 & 0.8 & 0.6 \\
\hline
0.4 & 0.5 & 0.4 \\
\hline
0.3 \\
\end{array}
\]

- \(x > \frac{1}{2}\)
- \(\frac{1}{3} < x \leq \frac{1}{2}\)
- \(\frac{1}{4} < x \leq \frac{1}{3}\)
- \(x \leq \frac{1}{4}\)
Harmonic Algorithm

Harmonic Algorithm classes: \((\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}]\).

Place members of each class separately from others.

\[ Harmonic \quad K = 4 \]

\[
< 0.9 \quad 0.3 \quad 0.8 \quad 0.5 \quad 0.1 \quad 0.1 \quad 0.3 \quad 0.2 \quad 0.4 \quad 0.2 \quad 0.4 \quad 0.5 \quad 0.5 \quad 0.8 \quad 0.6 \quad 0.4 \quad 0.5 \ldots >
\]
Analysis Measures

- Compare the performance of an online algorithm $A$ with an optimal offline algorithm $OPT$:
  - $OPT$ knows the whole sequence in the beginning.

Note that $OPT(\sigma) \geq S(\sigma)$ where $S(\sigma)$ is the total size of items. Even if all bins are packed tightly, $S(\sigma)$ bins are required.
Introduction (Bin Packing)

Analysis Measures

- Compare the performance of an online algorithm $A$ with an optimal offline algorithm $\text{OPT}$:
  - $\text{OPT}$ knows the whole sequence in the beginning.
  - Competitive ratio of $A$ is the maximum value of $A(\sigma)/\text{OPT}(\sigma)$ among all sequences $\sigma$.
    - We are interested in the asymptomatic competitive ratio where $\text{OPT}(\sigma)$ is arbitrary large.

Note that $\text{OPT}(\sigma) \geq S(\sigma)$ where $S(\sigma)$ is the total size of items.
Analysis Measures

- Compare the performance of an online algorithm $A$ with an optimal offline algorithm $\text{OPT}$:
  - $\text{OPT}$ knows the whole sequence in the beginning.
  - Competitive ratio of $A$ is the maximum value of $A(\sigma)/\text{OPT}(\sigma)$ among all sequences $\sigma$.
    - We are interested in the asymptomatic competitive ratio where $\text{OPT}(\sigma)$ is arbitrary large.
  - Note that $\text{OPT}(\sigma) \geq S(\sigma)$ where $S(\sigma)$ is the total size of items.
Introduction (Bin Packing)

Analysis Measures

- Compare the performance of an online algorithm $A$ with an optimal **offline** algorithm $\text{OPT}$:
  - $\text{OPT}$ knows the whole sequence in the beginning.
- Competitive ratio of $A$ is the maximum value of $A(\sigma)/\text{OPT}(\sigma)$ among all sequences $\sigma$.
  - We are interested in the **asymptomatic** competitive ratio where $\text{OPT}(\sigma)$ is arbitrary large.
- Note that $\text{OPT}(\sigma) \geq S(\sigma)$ where $S(\sigma)$ is the total size of items
  - Even if all bins are packed tightly, $S(\sigma)$ bins are required.
Competitive Analysis of Next Fit

- Competitive ratio of NextFit is at most 2.
Competitive Analysis of Next Fit

- Competitive ratio of NextFit is at most 2.
  - In the final packing, total size of items in each two consecutive bins is larger than 1.
Competitive Analysis of Next Fit

- Competitive ratio of NextFit is at most 2.
  - In the final packing, total size of items in each two consecutive bins is larger than 1.
  - The cost of NextFit for serving $\sigma$ is smaller than $2S(\sigma)$ where $S(\sigma)$ is the total size of items in $\sigma$.
    - Assume $cost(\text{NextFit}) = k$
    - Each two consecutive bins have total size $> 1 \rightarrow$ total-size $S(\sigma)$ of items in $\sigma$ is more than $k/2$
Competitive Analysis of Next Fit

- Competitive ratio of NextFit is at most 2.
  - In the final packing, total size of items in each two consecutive bins is larger than 1.
  - The cost of NextFit for serving $\sigma$ is smaller than $2S(\sigma)$ where $S(\sigma)$ is the total size of items in $\sigma$.
    - Assume $\text{cost}(\text{NextFit}) = k$
    - Each two consecutive bins have total size $\geq 1 \rightarrow$ total-size $S(\sigma)$ of items in $\sigma$ is more than $k/2$
  - $\text{OPT}(\sigma) \geq S(\sigma)$: Even when $\text{OPT}$ packs items tightly (with no wasted space), $S(\sigma)$ bins are required.
Competitive Analysis of Next Fit

- Competitive ratio of NextFit is at most 2.
  - In the final packing, total size of items in each two consecutive bins is larger than 1.
  - The cost of NextFit for serving $\sigma$ is smaller than $2S(\sigma)$ where $S(\sigma)$ is the total size of items in $\sigma$.
    - Assume $\text{cost}(\text{NextFit}) = k$
    - Each two consecutive bins have total size $> 1 \rightarrow$ total-size $S(\sigma)$ of items in $\sigma$ is more than $k/2$
  - $\text{OPT}(\sigma) \geq S(\sigma)$: Even when $\text{OPT}$ packs items tightly (with no wasted space), $S(\sigma)$ bins are required.
Competitive Analysis of Next Fit

- Competitive ratio of NextFit is at least 2.
Competitive Analysis of Next Fit

- Competitive ratio of NextFit is at least 2.
  - Consider sequence \( \sigma = < 0.5, \epsilon, 0.5, \epsilon, \ldots > \).
Competitive Analysis of Next Fit

- Competitive ratio of NextFit is at least 2.
  - Consider sequence $\sigma = < 0.5, \epsilon, 0.5, \epsilon, \ldots >$.
  - The cost of NextFit for serving $\sigma$ is roughly $n/2$ ($n$ is the length of $\sigma$).
  - The cost of $\text{OPT}$ is roughly $n/4$. 
**Competitive Analysis of Next Fit**

- Competitive ratio of NextFit is at least 2.
  - Consider sequence $\sigma = <0.5, \epsilon, 0.5, \epsilon, \ldots>$.
  - The cost of NextFit for serving $\sigma$ is roughly $n/2$ ($n$ is the length of $\sigma$).
  - The cost of OPT is roughly $n/4$.

\[
\begin{array}{cccccccc}
5 & 5 & 5 & 5 & 5 & 5 & \epsilon & \epsilon \\
5 & 5 & 5 & 5 & 5 & 5 & \epsilon & \epsilon \\
\end{array}
\]

NextFit

\[
\begin{array}{cccc}
5 & 5 & 5 & \epsilon \\
\end{array}
\]

OPT

\[
\begin{array}{cccc}
5 & 5 & 5 & \epsilon \\
\end{array}
\]
Introduction (Bin Packing)

Competitive Analysis of Next Fit

- Competitive ratio of NextFit is at least 2.
  - Consider sequence $\sigma = < 0.5, \epsilon, 0.5, \epsilon, \ldots >$.
  - The cost of NextFit for serving $\sigma$ is roughly $n/2$ ($n$ is the length of $\sigma$).
  - The cost of $OPT$ is roughly $n/4$.

\[ \begin{array}{cccccc}
\epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\
5 & 5 & 5 & 5 & 5 & 5 \\
\end{array} \quad \begin{array}{cccc}
5 & 5 & 5 & \\
5 & 5 & 5 & \\
\end{array} \]

NextFit

OPT

Theorem

Competitive ratio of NextFit is exactly 2.
In the next class, we visit an ‘easy’ lower bound argument that shows no online algorithm can achieve a competitive ratio better than $4/3$.

This bound can be greatly improved.