Review & Plan
Today’s objectives

- Fault-tolerant bin packing
- Renting servers in the cloud
Fault-tolerant Server Consolidation

An application of Bin Packing:
Fault-tolerant Server Consolidation
Bins represent servers and items are clients (e.g., databases tenant o a movie on NetFlix).

Server might fail and it should not interrupt the service (clients be should always available).
Fault-tolerant Server Consolidation

Fault-tolerant Bin Packing
(Server Consolidation in the Cloud)

- Bins represent servers and items are clients (e.g., databases tenant o a movie on NetFlix).
- Server might fail and it should not interrupt the service (clients be should always available).
- Given a sequence of items, place two replicas of each item in different servers
  - Each replica of an item with load $x$ has a load of $x/2$.
  - Think of load as the number of people who watch a NetFlix movie; so each replica requires half bandwidth
Bins represent servers and items are clients (e.g., databases tenant o a movie on NetFlix).

Server might fail and it should not interrupt the service (clients be should always available).

Given a sequence of items, place two replicas of each item in different servers

- Each replica of an item with load $x$ has a load of $x/2$.
- Think of load as the number of people who watch a NetFlix movie; so each replica requires half bandwidth

In case of a server’s failure, the load of each replica is redirected to the server that hosts its partner.
Valid Solutions

- Consider sequence
  \[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]
Valid Solutions

- Consider sequence 
  \( \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle \).
- A valid packing:
Valid Solutions

Consider sequence

\[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]

A valid packing:
Valid Solutions

Consider sequence
\[
\langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle.
\]

A valid packing:
Consider sequence
\[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]
A valid packing:
Consider sequence 
\[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]

A valid packing:
Consider sequence $\langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle$.

A valid packing:
Valid Solutions

- Consider sequence 
  \[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]

- A valid packing:
Valid Solutions

- Consider sequence
  \[\langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle.\]
- A valid packing:
Valid Solutions

Consider sequence
\[\langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle.\]

An invalid packing:
Consider sequence
\[\langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle.\]

An invalid packing:
Fault-tolerant Server Consolidation

Valid Solutions

- Consider sequence 
  \[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]

- An invalid packing:
Consider two types of replicas (blue and red), and apply Best Fit for each type separately

- Assume a capacity of 1/2 for the bins (why?)

Consider sequence

\[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]
Consider two types of replicas (blue and red), and apply Best Fit for each type separately
- Assume a capacity of 1/2 for the bins (why?)

Consider sequence
\[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]
Fault-tolerant Server Consolidation

Mirroring Algorithms

- Consider two types of replicas (blue and red), and apply Best Fit for each type separately
  - Assume a capacity of 1/2 for the bins (why?)

Consider sequence
\( \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \)
Fault-tolerant Server Consolidation

Mirroring Algorithms

- Consider two types of replicas (blue and red), and apply Best Fit for each type separately
  - Assume a capacity of 1/2 for the bins (why?)

- Consider sequence
  \[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]
Consider two types of replicas (blue and red), and apply Best Fit for each type separately

- Assume a capacity of 1/2 for the bins (why?)

Consider sequence
\[
\langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle.
\]
Consider two types of replicas (blue and red), and apply Best Fit for each type separately

- Assume a capacity of 1/2 for the bins (why?)

Consider sequence

\[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]
Fault-tolerant Server Consolidation

Mirroring Algorithms

- Consider two types of replicas (blue and red), and apply Best Fit for each type separately
  - Assume a capacity of 1/2 for the bins (why?)

- Consider sequence
  \[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]
Fault-tolerant Server Consolidation

Mirroring Algorithms

- Consider two types of replicas (blue and red), and apply Best Fit for each type separately
  - Assume a capacity of 1/2 for the bins (why?)
  - The level of a bin should never be more than 0.5 (otherwise there will be an overflow in case of a bin failure)

- Consider sequence
  \[ \langle a = 0.6, b = 0.3, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle. \]
Mirroring algorithms are not better than 2-competitive

Consider sequence \( \langle 2\epsilon_1, 2\epsilon_2, \ldots, 2\epsilon_n \rangle \)

\( \text{OPT} \) can place all items so that all bins are almost full

- Each two bin share at most one item!
Like Harmonic, define *classes* for replicas.

- \( \left[ \frac{1}{3}, \frac{1}{2} \right] , \left[ \frac{1}{4}, \frac{1}{3} \right] , \ldots , \left[ \frac{1}{K} , \frac{1}{K-1} \right] , (0, \frac{1}{K}] \) (E.g., \( K = 30 \)).

Treat members of each class separately.
Horizontal Harmonic (HH) Algorithm

- Like Harmonic, define \textit{classes} for replicas.
  - \((\frac{1}{3}, \frac{1}{2}], (\frac{1}{4}, \frac{1}{3}], \ldots, (\frac{1}{K}, \frac{1}{K-1}], (0, \frac{1}{K}] \) (E.g., \( K = 30 \)).
- Treat members of each class separately.
  - No two bins share more than one replica.
Consider sequence $\langle a_1, a_2, \ldots a_m \rangle$ of replicas of the same class (E.g., for class 3, replicas lie in the range $(1/5, 1/4]$).
Consider sequence \( \langle a_1, a_2, \ldots, a_m \rangle \) of replicas of the same class (E.g., for class 3, replicas lie in the range \((1/5, 1/4]\)).

Place \( i \) blue replicas of class \( i < K \) in the same bin.
Horizontal Harmonic (HH) Algorithm

Consider sequence \( \langle a_1, a_2, \ldots, a_m \rangle \) of replicas of the same class (E.g., for class 3, replicas lie in the range \((1/5, 1/4]\)).

Place \( i \) blue replicas of class \( i < K \) in the same bin.

Place red replicas whose partners are in the same bin in different bins.

This ensures a valid packing.
Consider sequence $\langle a_1, a_2, \ldots, a_m \rangle$ of replicas of the same class (E.g., for class 3, replicas lie in the range $(1/5, 1/4]$).

- Place $i$ blue replicas of class $i < K$ in the same bin.
- Place red replicas whose partners are in the same bin in different bins.
  - This ensures a valid packing.
Consider sequence $\langle a_1, a_2, \ldots, a_m \rangle$ of replicas of the same class (E.g., for class 3, replicas lie in the range $(1/5, 1/4]$).

Place $i$ blue replicas of class $i < K$ in the same bin.

Place red replicas whose partners are in the same bin in different bins.

This ensures a valid packing.
Consider sequence \( \langle a_1, a_2, \ldots, a_m \rangle \) of replicas of the same class (E.g., for class 3, replicas lie in the range \((1/5, 1/4]\)).

- Place \( i \) blue replicas of class \( i < K \) in the same bin.
- Place red replicas whose partners are in the same bin in different bins.
  - This ensures a valid packing.
Consider sequence \( \langle a_1, a_2, \ldots, a_m \rangle \) of replicas of the same class (E.g., for class 3, replicas lie in the range \((1/5, 1/4]\)).

Place \( i \) blue replicas of class \( i < K \) in the same bin.

Place red replicas whose partners are in the same bin in different bins.

- This ensures a valid packing.
Horizontal Harmonic (HH) Algorithm

Consider sequence $\langle a_1, a_2, \ldots, a_m \rangle$ of replicas of the same class (E.g., for class 3, replicas lie in the range $(1/5, 1/4]$).

Place $i$ blue replicas of class $i < K$ in the same bin.

Place red replicas whose partners are in the same bin in different bins.

This ensures a valid packing.
Horizontal Harmonic (HH) Algorithm

Consider sequence \( \langle a_1, a_2, \ldots a_m \rangle \) of replicas of the same class (E.g., for class 3, replicas lie in the range \([1/5, 1/4])\).

Place \( i \) blue replicas of class \( i < K \) in the same bin.

Place red replicas whose partners are in the same bin in different bins.

This ensures a valid packing.
Horizontal Harmonic (HH) Algorithm

Consider sequence \( \langle a_1, a_2, \ldots, a_m \rangle \) of replicas of the same class (E.g., for class 3, replicas lie in the range \((1/5, 1/4]\)).

Place \( i \) blue replicas of class \( i < K \) in the same bin.

Place red replicas whose partners are in the same bin in different bins.

This ensures a valid packing.
Horizontal Harmonic (HH) Algorithm

Consider sequence \( \langle a_1, a_2, \ldots a_m \rangle \) of replicas of the same class (E.g., for class 3, replicas lie in the range \([1/5, 1/4]\)).

Place \( i \) blue replicas of class \( i < K \) in the same bin.

Place red replicas whose partners are in the same bin in different bins.

This ensures a valid packing.
Consider sequence \( \langle a_1, a_2, \ldots, a_m \rangle \) of replicas of the same class (E.g., for class 3, replicas lie in the range \( (1/5, 1/4] \)).

Place \( i \) blue replicas of class \( i < K \) in the same bin.

Place red replicas whose partners are in the same bin in different bins.

This ensures a valid packing.
Consider sequence $\langle a_1, a_2, \ldots, a_m \rangle$ of replicas of the same class (E.g., for class 3, replicas lie in the range $(1/5, 1/4]$).

- Place $i$ blue replicas of class $i < K$ in the same bin.
- Place red replicas whose partners are in the same bin in different bins.
  - This ensures a valid packing.
Horizontal Harmonic (HH) Algorithm

Horizontal Harmonic (HH) in a Nutshell

- Class $i < k$: place $i$ replicas of the range $\left(\frac{1}{i+2}, \frac{1}{i+1}\right]$ in the same bin.
  - No two bins share more than one replica
- Use mirroring for items of class $k$
Horizontal Harmonic (HH) Algorithm

Analysis of Horizontal Harmonic

**Summary of weighting argument to prove an algorithm has c.r. at most *J*:**

- **Step I:** Define a weight function \( w(x) \geq x \) for an item of size \( x \)
- **Step II:** Prove that any bin of the online algorithm has weight 1.
- **Step III:** Prove that it is not possible to place a total weight more than *J* in any empty bin
Define the weight of an item of class $i$ to be $1/i$.

- Weight of the bins of that type in HH packing will be 1

Define the weight of an item of class $k$ to be $\frac{2(k+1)}{k-1}$.

- Items of class $k$ are no larger than $1/(k + 1)$
- Level of these bins is at least $1/2 - 1/(k + 1) = \frac{k-1}{2(k+1)}$

Steps I and II are done.
Step III: how much the weight of an optimal bin can be?

- Similar to the case of Harmonic, it is better to fill the bin with the larger replicas → They have higher density!
- The reserved space should be no less than the largest replica!

Using case analysis we can show the maximum weight for large $k$ is

$$w(1/3+\epsilon) + w(1/4+\epsilon) + w(1/13+\epsilon) + \ldots = 1 + 1/2 + 1/12 + \ldots \approx 1.597$$
Horizontal Harmonic (HH) Algorithm

Analysis of Horizontal Harmonic

Theorem

*Competitive ratio of Horizontal Harmonic is at most 1.597 for large values of K.*

- This bound is tight!
A ‘real-world’ application of bin packing

Mirroring algorithms used in practice have c.r. of 2

Horizontal Harmonic packs replicas more tightly and has a c.r. of at most 1.597

Real-world implementation of Horizontal-Harmonic shows promising performance.

The algorithms works well in both worst-case and average-case.
Renting Servers in the Cloud

Application II:
Renting Servers in the Cloud
Renting Servers in the Cloud

Buying vs. Renting

- When you buy servers, the goal is to minimize the total number of opened (purchased) servers.
Renting Servers in the Cloud

Buying vs. Renting

- When you **buy** servers, the goal is to minimize the total number of opened (purchased) servers.

- When you **rent** servers, the goal is to minimize the total **time** you have rented servers.
  - Each item has an arrival and a departure time.
  - The difference is the **Length** of the item.
Renting Servers in the Cloud

**First Fit Algorithm**

- Apply First Fit algorithm to place items.
Renting Servers in the Cloud

First Fit Algorithm

- Apply First Fit algorithm to place items.
- Release the bin when all items depart.

< $a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots>$

Time: 0
Renting Servers in the Cloud

First Fit Algorithm

- Apply First Fit algorithm to place items.
- **Release** the bin when all items depart.

< $a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots>$

Time: 1
Renting Servers in the Cloud

First Fit Algorithm

- Apply First Fit algorithm to place items.
- **Release** the bin when all items depart.

< \( a = (0.3, 1, 7), \) \( b = (0.4, 2, 7), \) \( c = (0.4, 3, 7), \) \( d = (0.5, 4, 7), \) \( e = (0.3, 5, 20), \) \( f = (0.1, 6, 20), \ ... > 

Time: 2
Renting Servers in the Cloud

First Fit Algorithm

- Apply First Fit algorithm to place items.
- **Release** the bin when all items depart.

\(< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots >\)

Time: 3
Renting Servers in the Cloud

First Fit Algorithm

- Apply First Fit algorithm to place items.
- Release the bin when all items depart.

< \( a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots > \\

Time: 4
Renting Servers in the Cloud

First Fit Algorithm

- Apply First Fit algorithm to place items.
- **Release** the bin when all items depart.

\[
< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots >
\]

*Time: 5*
Renting Servers in the Cloud

First Fit Algorithm

- Apply First Fit algorithm to place items.
- **Release** the bin when all items depart.

\[ a = (0.3, 1, 7), \quad b = (0.4, 2, 7), \quad c = (0.4, 3, 7), \quad d = (0.5, 4, 7), \quad e = (0.3, 5, 20), \quad f = (0.1, 6, 20), \ldots \\]

**Time:** 6
Renting Servers in the Cloud

First Fit Algorithm

- Apply First Fit algorithm to place items.
- **Release** the bin when all items depart.

\[ a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), ... \]

Time: 7
Renting Servers in the Cloud

First Fit Algorithm

- Apply First Fit algorithm to place items.
- **Release** the bin when all items depart.
- The cost of First Fit is \((20-1) + (20-3) = 36\) (assuming no other item arrives till time 20).

\(< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots >\)

Time: 7
Renting Servers in the Cloud

First Fit Algorithm

- Apply First Fit algorithm to place items.
- **Release** the bin when all items depart.
- The cost of First Fit is \((20-1) + (20-3) = 36\) (assuming no other item arrives till time 20).

\(< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots \>

Time: 7
Renting Servers in the Cloud

**First Fit Algorithm**

- Apply First Fit algorithm to place items.
- **Release** the bin when all items depart.
- The cost of First Fit is \((20-1) + (20-3) = 36\) (assuming no other item arrives till time 20).

\(< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots >\)

Time: 7
No Any-Fit algorithm can be better than $\mu$ competitive

$\mu$ is the ratio between the length of the largest and the smallest item LiTang14.

First Fit is at most $2\mu + 13$-competitive LiTang14.

Best Fit is not competitive.
Renting Servers in the Cloud

**Next Fit Algorithm**

- Apply Next Fit algorithm to place items.

The cost of Next Fit is $(7-1) + (7-3) + (20-5) = 25$ (assuming no other item arrives till time 20).
Renting Servers in the Cloud

**Next Fit Algorithm**

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.

\[
\begin{align*}
< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots > \\
\text{Time: 0}
\end{align*}
\]
Renting Servers in the Cloud

**Next Fit Algorithm**

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.

< $a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots >$

Time: 1
Next Fit Algorithm

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.

< $a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots \rangle$

Time: **2**
Renting Servers in the Cloud

**Next Fit Algorithm**

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.

< $a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), ...$>

Time: 3
Renting Servers in the Cloud

Next Fit Algorithm

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.

< \( a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots \)>

Time: 4
Renting Servers in the Cloud

Next Fit Algorithm

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.

\[
< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots >
\]

Time: 5
Renting Servers in the Cloud

Next Fit Algorithm

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.

< $a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots>$

Time: 6
Renting Servers in the Cloud

Next Fit Algorithm

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.

< \(a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots\)>

Time: 7
Renting Servers in the Cloud

Next Fit Algorithm

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.
- The cost of Next Fit is \((7-1) + (7-3) + (20-5) = 25\) (assuming no other item arrives till time 20).

\(< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), … >\)

Time: 7
Renting Servers in the Cloud

Next Fit Algorithm

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.
- The cost of Next Fit is \((7-1) + (7-3) + (20-5) = 25\) (assuming no other item arrives till time 20).

\(< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), … >\)

Time: 7
Renting Servers in the Cloud

Next Fit Algorithm

- Apply Next Fit algorithm to place items.
- **Release** the bin when all items depart.
- The cost of Next Fit is \((7-1) + (7-3) + (20-5) = 25\) (assuming no other item arrives till time 20).

\(< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots >\)

Time: 7

\[
\begin{array}{c}
\text{f0.1} \\
\text{e0.3}
\end{array}
\]
New Results for Renting Servers

- No algorithm can be better than $\mu$-competitive.
- Next Fit is at most $2\mu + 1$-competitive.
- If the value of $\mu$ is known, one can achieve a $\mu + 2$-competitive algorithm.
Renting Servers in the Cloud

Boosting Average-Case Performance

- On average, Best Fit is still better than Next Fit and First Fit.
- We introduce a new algorithm Move To Front.
  - An Any Fit algorithm that applies after placing an item, moves the bin to the front.
Renting Servers in the Cloud

Move To Front Algorithm

- We introduce a new algorithm Move To Front.
Renting Servers in the Cloud

Move To Front Algorithm

- We introduce a new algorithm Move To Front.
- An Any Fit algorithm that applies after placing an item, moves the bin to the front.
  - Intuitively, items that arrive together are more likely to depart at the same time.

\[< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots>\]

Time: 0
Renting Servers in the Cloud

Move To Front Algorithm

- We introduce a new algorithm Move To Front.
- An Any Fit algorithm that applies after placing an item, moves the bin to the front.
  - Intuitively, items that arrive together are more likely to depart at the same time.

\[ \langle a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots \rangle \]

Time: 1

\[ a = (0.3, 0.3) \]
Renting Servers in the Cloud

Move To Front Algorithm

- We introduce a new algorithm Move To Front.

- An Any Fit algorithm that applies after placing an item, moves the bin to the front.
  - Intuitively, items that arrive together are more likely to depart at the same time.

\[ a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots \]

Time: 2

\[ \begin{array}{c}
\text{b} \\
0.4 \\
\hline
\text{a} \\
0.3
\end{array} \]
Renting Servers in the Cloud

Move To Front Algorithm

- We introduce a new algorithm Move To Front.
- An Any Fit algorithm that applies after placing an item, moves the bin to the front.
  - Intuitively, items that arrive together are more likely to depart at the same time.

\(<\ a\ =\ (0.3,\ 1,\ 7),\ b\ =\ (0.4,\ 2,\ 7),\ c\ =\ (0.4,\ 3,\ 7),\ d\ =\ (0.5,\ 4,\ 7),\ e\ =\ (0.3,\ 5,\ 20),\ f\ =\ (0.1,\ 6,\ 20),\ ...>\)

Time: 3
We introduce a new algorithm Move To Front.

An Any Fit algorithm that applies after placing an item, moves the bin to the front.

Intuitively, items that arrive together are more likely to depart at the same time.

< \(a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots\) >

Time: 3
Renting Servers in the Cloud

Move To Front Algorithm

- We introduce a new algorithm Move To Front.
- An Any Fit algorithm that applies after placing an item, moves the bin to the front.
  - Intuitively, items that arrive together are more likely to depart at the same time.

< \( a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots > 

Time: 4
Renting Servers in the Cloud

Move To Front Algorithm

- We introduce a new algorithm Move To Front.
- An Any Fit algorithm that applies after placing an item, moves the bin to the front.
  - Intuitively, items that arrive together are more likely to depart at the same time.

\[ <a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), \ldots > \]

Time: 5
Renting Servers in the Cloud

Move To Front Algorithm

- We introduce a new algorithm Move To Front.
- An Any Fit algorithm that applies after placing an item, moves the bin to the front.
  - Intuitively, items that arrive together are more likely to depart at the same time.

\(<a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), ...>\)

Time: 5
Renting Servers in the Cloud

**Move To Front Algorithm**

- We introduce a new algorithm Move To Front.
- An Any Fit algorithm that applies after placing an item, moves the bin to the front.
  - Intuitively, items that arrive together are more likely to depart at the same time.

\[ <a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), ... > \]

Time: 6
Renting Servers in the Cloud

Move To Front Algorithm

- We introduce a new algorithm Move To Front.
- An Any Fit algorithm that applies after placing an item, moves the bin to the front.
  - Intuitively, items that arrive together are more likely to depart at the same time.

\(< a = (0.3, 1, 7), b = (0.4, 2, 7), c = (0.4, 3, 7), d = (0.5, 4, 7), e = (0.3, 5, 20), f = (0.1, 6, 20), ... >\)

Time: 6
Renting Servers in the Cloud

Average-Case Performance of Online Algorithms

- Competitive ratio of Move To Front is at most $6\mu + 7$.
- On average (sequences with uniform size and length), Move To Front outperforms all algorithms.