```
// You need to
//    1. Read the programming assignment in homework #1.
//    2. Implement function GetStudentName.
//    3. Implement function FindKey.
//    4. Compile your code as explained in the PDF file.
//    5. Run the executable on small and large unit tests.
//    6. Test and debug your code. Make sure that your program does not have
//       any memory leaks.
//    7. Remove all commented out code. Double check that your program does not
//       print any debug information on the screen.
//    8. Submit your source code ("student_code_1.h") on Canvas.
```

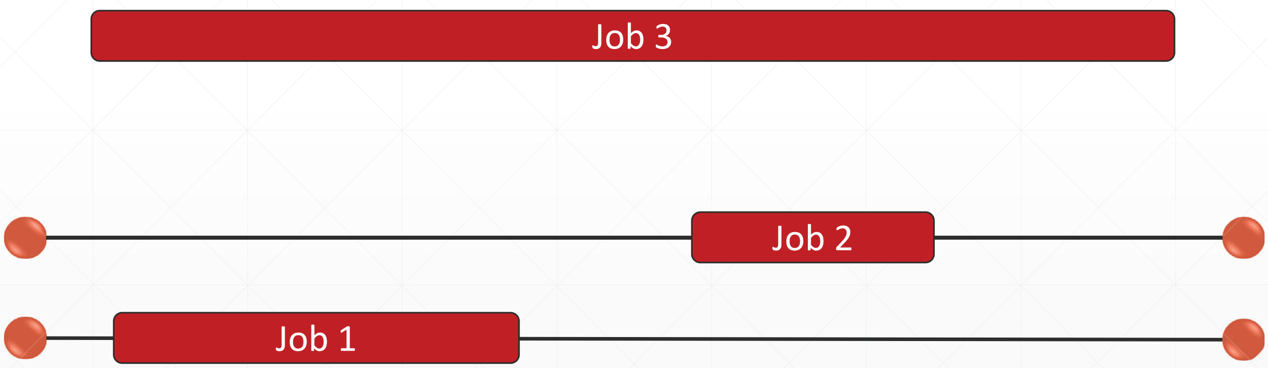
# Multiple Machines

Can we use greedy for more than two machines?

## Can we use greedy for multiple machines?


## Can we use greedy for multiple machines?

## Can we use greedy for multiple machines?




## Two Problems to Solve

- We need to
  - Select jobs that we want to run.
  - Assign jobs to machines.
- Let's solve them separately
  - Find a set of jobs $\boldsymbol{A}$ such that at every point of time no more than $m$ jobs are scheduled.
  - Assign jobs in $\boldsymbol{A}$ to machines (using the Machine Minimization algorithm).

## Algorithm for Selecting Jobs

Create a set of jobs $A = \emptyset$, which we are going to run.
Sort all jobs in $J$ by their finish time $f(j)$.

for every job $j$ in $J$:

If we can add $j$ to A without *overloading* machines, do it;
Otherwise, discard the job.

return $A$

## Selection Problem

- $\boldsymbol{A} = \{a_1, \dots, a_k\}$ – solution returned by **Alg**.
- $\boldsymbol{O} = \{o_1, \dots, o_{k'}\}$ — optimal solution.

**Claim:** For all $i \in k$, there exists an optimal solution $H_i$
i.e. $|H_i| = |O|$ that contains the first $i$ jobs in $\boldsymbol{A}$**:**

$$a_1, \dots, a_i \in H_i$$

## Selection Problem

- $\boldsymbol{A} = \{a_1, \dots, a_k\}$ – solution returned by **Alg**.
- $\boldsymbol{O} = \{o_1, \dots, o_{k'}\}$ — optimal solution.

**Claim:** For all $i \in k$, there exists an optimal solution $H_i$ i.e. $|H_i| = |O|$ that contains the first $i$ jobs in $\boldsymbol{A}$:

$$a_1, \dots, a_i \in H_i$$

**Proof by induction.**

Base case: $i = 0$. $\mathrm{H}_0 = \boldsymbol{O}$ is such a solution.

## Selection Problem

- $\boldsymbol{A} = \{a_1, \dots, a_k\}$ – solution returned by **Alg**.
- $\boldsymbol{O} = \{o_1, \dots, o_{k'}\}$ — optimal solution.

**Claim:** For all $i \in k$, there exists an optimal solution $H_i$ i.e. $|H_i| = |O|$ that contains the first $i$ jobs in $\boldsymbol{A}$:

$$a_1, \dots, a_i \in H_i$$

**Proof by induction.**

Inductive step: $H_i$ is an optimal solution containing $a_1, \dots, a_i$. Need to construct $H_{i+1}$.

**Claim:** For all $i \in k$, there exists an optimal solution $H_i$ i.e. $|H_i| = |O|$ that contains the first $i$ jobs in $\boldsymbol{A}$:

$$a_1, \dots, a_i \in H_i$$

**Proof by induction.**

Inductive step: $H_i$ is an optimal solution containing $a_1, \dots, a_i$. Need to construct $H_{i+1}$.

**Claim:** For all $i \in k$, there exists an optimal solution $H_i$ i.e. $|H_i| = |O|$ that contains the first $i$ jobs in $\boldsymbol{A}$:

$$a_1, \dots, a_i \in H_i$$

**Proof by induction.**

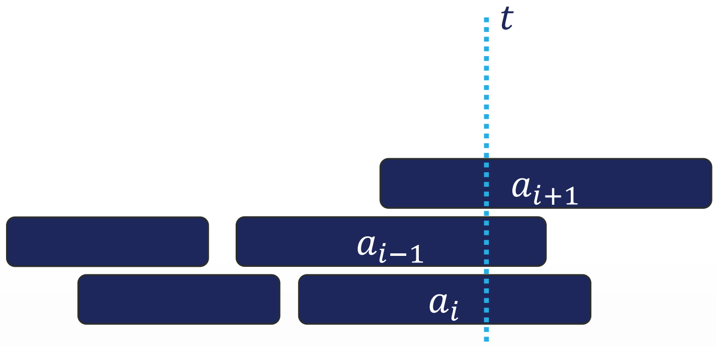
Inductive step: $H_i$ is an optimal solution containing $a_1, \dots, a_i$. Need to construct $H_{i+1}$.

- If $a_{i+1} \in H_i$, then let $H_{i+1} = H_i$ and we are done.
- Otherwise: insert $a_{i+1}$ and remove $b \in H_i \backslash \{a_1, \dots, a_i\}$ with the earliest start time $s(b)$.

$$H_{i+1} = H_i + \{a_{i+1}\} - \{b\}$$

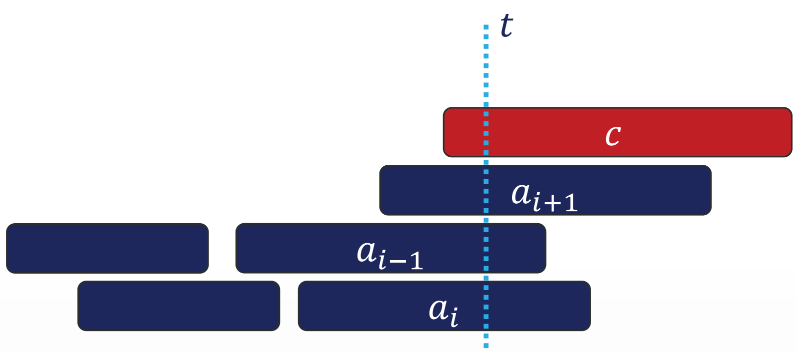
We need to show that $H_{i+1}$ is a ***feasible*** solution for the selection problem.

## What can go wrong?



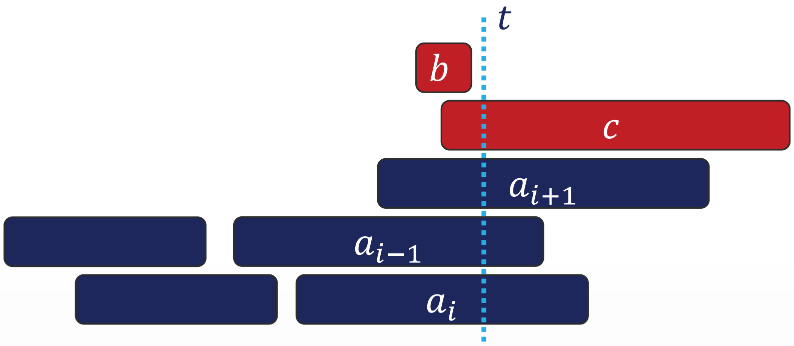

- By inserting job $a_{i+1}$ in $H_i$, we can potentially overload machines at time $t$.
- Jobs $a_1, \dots, a_i, a_{i+1}$ are in **A.** They do not overload machines on their own, because **A** is feasible. Thus, there must be another job $c$ in $H_i$ active at time $t$.

## What can go wrong?



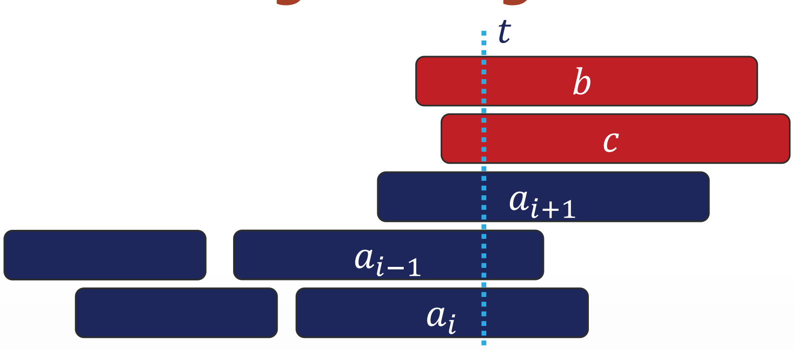

- Jobs $a_1, \dots, a_i, a_{i+1}$ are in **A.** They do not overload machines on their own, because **A** is feasible. Thus, there must be another job $c$ in $H_i$ active at time $t$.
- However, job $b$ starts before job $c$.

## What can go wrong?




- Jobs $a_1, \ldots, a_i, a_{i+1}$ are in **A.** They do not overload machines on their own, because **A** is feasible. Thus, there must be another job $c$ in $H_i$ active at time $t$.
- However, job $b$ starts before job $c$. It also finishes after job $a_{i+1}$!

## What can go wrong?




- Jobs $a_1, \ldots, a_i, a_{i+1}$ are in **A.** They do not overload machines on their own, because **A** is feasible. Thus, there must be another job $c$ in $H_i$ active at time $t$.
- Therefore, $b$ is active at time $t$. Since it is removed from $H_i$, the total number of active jobs at time $t$ is at most $m$.

# Sum of Weighted Completion Times

CS 336: Design and Analysis of Algorithms


## Sum of Weighted Completion Times

- We have $n$ jobs $1, \ldots, n$ with processing times $p_1, .., p_n$ and weights $w_1, \ldots, w_n$. We need to schedule them on one machine one after another so as to minimize

$$\sum_j w_j C_j$$

where $C_j$ is the completion time of job $j$.

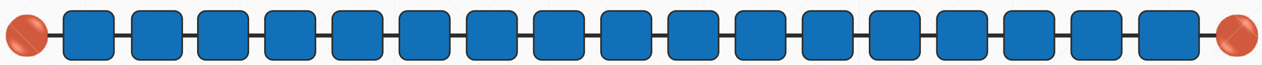
**Our goal it to find the optimal ordering of jobs.**

## Discussion

- Assume that all jobs have the same processing time:

$$p_j = 1$$

- Then, we need to schedule one job at time 0, one job at time 1, etc.

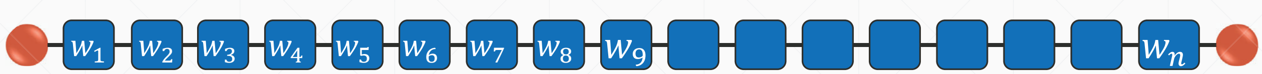


## Discussion

- Assume that all jobs have the same processing time:

$$p_j = 1$$

- Then, we need to schedule one job at time 0, one job at time 1, etc.
- The objective equals to $\sum_{j=1}^{n} j \cdot w_j$.

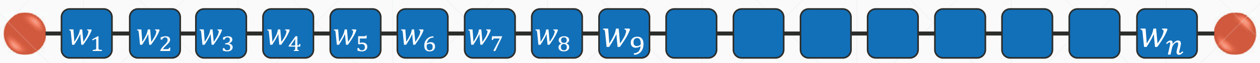


## Discussion

- Assume that all jobs have the same processing time:

$$p_j = 1$$

- Then, we need to schedule one job at time 0, one job at time 1, etc.
- The objective equals to $\sum_{j=1}^{n} j \cdot w_j$.
- **Solution:** Sort $w_j$'s in decreasing order.

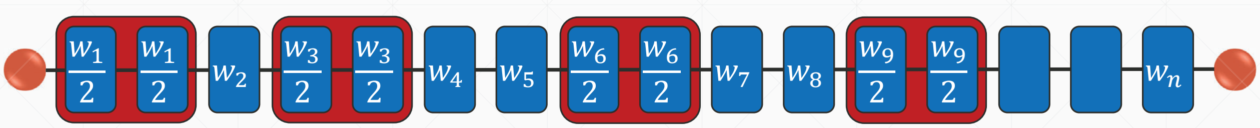



## Discussion

- Assume that there are only two different processing times:

$$p_j = 1 \text{ or } p_j = 2$$

- **Solution:** Sort jobs in decreasing order of $w_j / p_j$.



## Smith's Rule

Compute $x_j = w_j / p_j$ for all $j$.

Sort jobs in decreasing order of $x_j$.

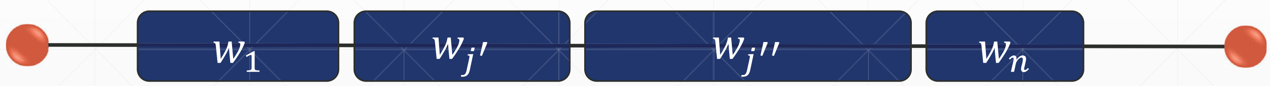
- $x_j$ is the money per minute.
- The rule is simple and fast!

## Proof uses Bubble Sort!

- Assume that all $x_j$ are distinct.
- Bubble sort compares two consecutive elements $j'$ and j'' in the array and swaps them if $x_{j'} < x_{j''}$.
- Bubble sort is slow, but it works.
- Let's sort the optimal solution using Bubble sort.
- What happens when we swap jobs $j'$ and $j''$?

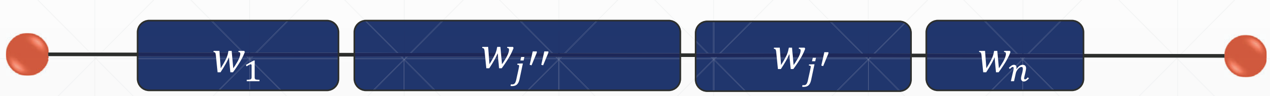
## Swaps

- Swaps affect only jobs $j'$ and $j''$.




## Swaps

- Swaps affect only jobs $j'$ and $j''$.



# Swaps

- Swaps affect only jobs $j'$ and $j''$.
- Decrease the objective function by $p_{j'}w_{j''} - p_{j''}w_{j'}$.

$$p_{j'}w_{j''} - p_{j''}w_{j'} > 0 \Leftrightarrow \frac{w_{j'}}{p_{j'}} < \frac{w_{j''}}{p_{j''}}$$