# Algorithmic Strategies 2024/25

# Week 6 – Greedy Algorithms


· U C ·

Universidade de Coimbra

# Interval partitioning problem

- A set $S = \{1, \dots, n\}$ of activities needs a room. Each room can only be used by one activity at a time.

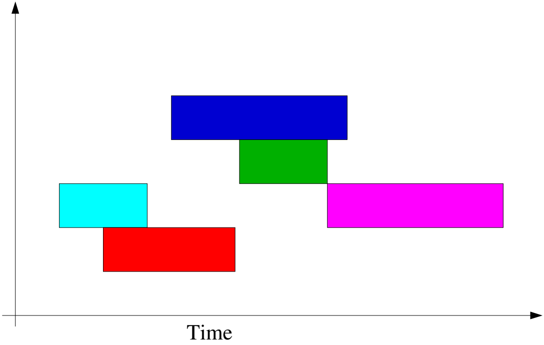
- Each activity $i$ takes place during the interval $[s_i, f_i)$.

- Two activities are compatible if their intervals do not overlap.

Select the minimum number of rooms required to schedule all activities.

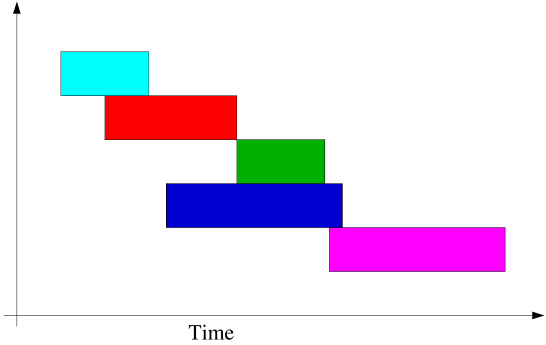
(see Chapter 4.1 of Kleinberg and Tardos, Algorithm Design)

# Interval partitioning problem



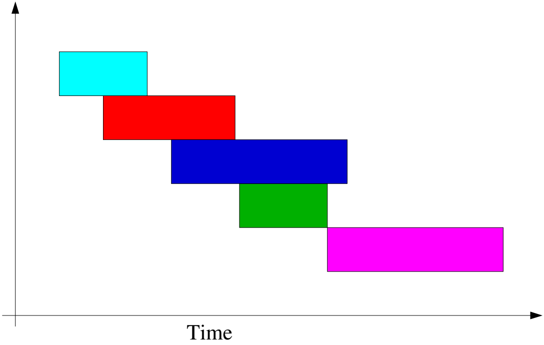

How to solve the problem?

# Interval partitioning problem




Sorting in non-decreasing order by finishing times?

# Interval partitioning problem




Sorting in non-decreasing order by starting times?

# Interval partitioning problem

---

**Function** *partition*$(S)$

sort $S$ into nondecreasing order by starting times $s_i$

$R_1 = R_2 = \ldots = R_n = \emptyset$ $\{n$ rooms available$\}$

$d = 0$ $\{d$ is the number of rooms used$\}$

**for** $i$ in $S$ **do**

  **if** $i$ can be assigned to some room $k \leq d$ **then**

    $R_k = R_k \cup \{i\}$

  **else**

    $R_{d+1} = R_{d+1} \cup \{i\}$

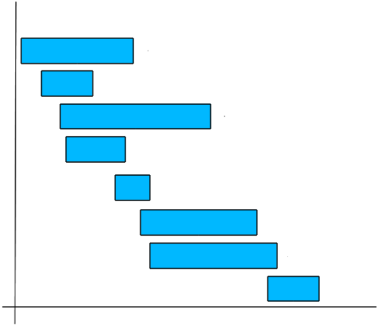
    $d = d + 1$

**return** $d$

---

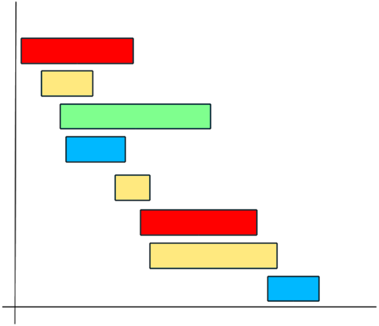
- Greedy choice: choose the next earliest starting time activity
- Why it works?

# Interval partitioning problem



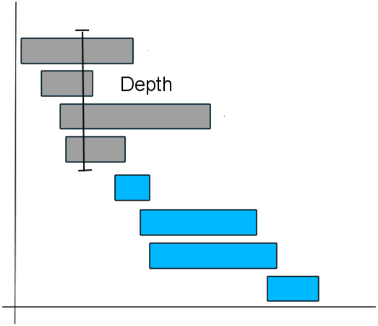
Sorting in non-decreasing order by starting times.

# Interval partitioning problem



A solution to the problem.

# Interval partitioning problem




Depth $d$ is the maximum overlap of activities that can exist, which is the minimum number of rooms required.

# Interval partitioning problem

Claim: The greedy algorithm finds $d$ rooms, where $d$ is the depth

- Assume that the greedy algorithm finds more than $d$ rooms
- Then, at some point in the algorithm, an activity $j$ had requested the $d+1$-th room.
- This can only happen if there exists $d$ activities that are not compatible among themselves (have requested $d$ rooms) that started before activity $j$ and overlap with its starting time. This contradicts the fact that depth is $d$.

This is easier to show than by using optimal substructure and greedy choice properties.

# Interval partitioning problem

**Function** *partition*$(S)$

sort $S$ into nondecreasing order by starting times $s_i$
$R_1 = R_2 = \ldots = R_n = \emptyset$ $\{n$ rooms available$\}$
$d = 0$ $\{d$ is the number of rooms used$\}$
**for** $i$ in $S$ **do**
  **if** $i$ can be assigned to some room $k \leq d$ **then**
    $R_k = R_k \cup \{i\}$
  **else**
    $R_{d+1} = R_{d+1} \cup \{i\}$
    $d = d + 1$
**return** $d$

Naïve approach in $O(n^2)$ time: search for all available rooms at each step.

# Interval partitioning problem

**Function** *partition*$(S)$

```
sort S into nondecreasing order by starting times s_i
R_1 = R_2 = ... = R_n = ∅                      {n rooms available}
d = 0                               {d is the number of rooms used}
for i in S do
    if i can be assigned to some room k ≤ d then
        R_k = R_k ∪ {i}
    else
        R_{d+1} = R_{d+1} ∪ {i}
        d = d + 1
return d
```

$O(n \log n)$ time with a min-priority queue that keeps track of the room that has the earliest finish time; see `priority_queue` in `C++`.

## Approximation algorithm to the knapsack problem

1. Sort the items in nonincreasing order of the ratio $v_j/w_j$
2. Let $k$ be the first item such that $\sum_{i=1}^{k} w_i > W$
3. Let $x$ be a solution with items $1, \ldots, k-1$ and value $v(x) = \sum_{i=1}^{k-1} v_i$
4. Let $x'$ be a solution that contains only item $k$ and value $v(x') = v_k$
5. Return $\max(v(x), v(x'))$

This greedy algorithm returns returns 1/2-approximation to the knapsack (the value is 1/2 of the optimal value in the worst case).

Why?

## Approximation algorithm to the knapsack problem

- Let $x^*$ be the optimal solution for the knapsack problem and let $z^*$ be the optimal solution for the fractional knapsack problem. Note that $v(x^*) \leq v(z^*)$.
- Note that $z^* = (1, \ldots, 1, \alpha, 0, \ldots, 0)$, where $\alpha \in [0, 1)$ at $k$.
- Note that $x = (1, \ldots, 1, 0, 0, \ldots, 0)$.
- Note that $x' = (0, \ldots, 0, 1, 0, \ldots, 0)$.

$$v(x^*) \leq v(z^*) = v(x) + \alpha v_k \leq v(x) + v(x') \leq 2 \max(v(x), v(x'))$$

$$\max(v(x), v(x')) \geq \frac{v(x^*)}{2}$$