



# Data Structures and Algorithms Design

**BITS Pilani**  
Hyderabad Campus

Febin.A.Vahab

# SESSION -PLAN



Session(#)	List of Topic Title	Text/Ref Book/external resource
10	<b>M5:Algorithm Design Techniques</b>  Greedy Method - Design Principles and Strategy, Fractional Knapsack Problem, Task Scheduling Problem  Minimum Spanning Tree, Shortest Path Problem - Dijkstra's Algorithm	T1: 5.1, 7.1, 7.3

# Greedy method-Change making problem



25, 10, 5, 1 ✓✓

{1, 1, 1, 1, 1, 5, 5, 10, 10, 25} ✓

32

$\begin{array}{r} 1, 25 \\ 1, 5 \\ \hline 2, 1 \end{array}$

4 {1, 1, 1, 1, 1, 10, 10, 15} ✓

20)

$\begin{array}{r} 1, 15 \\ 5, 1 \\ \hline 6 \end{array}$

2, 10

- Divide and Conquer
- Greedy method
- Dynamic
- Backtracking
- Branch & Bound



# Greedy Method



- **The greedy method** is a general algorithm design paradigm, built on the following elements:
  - **configurations**: different choices, collections, or values to find
  - **objective function**: a score assigned to configurations, which we want to either maximize or minimize

# Greedy Method

$$\underline{32} \Rightarrow \begin{array}{r} \cancel{1} \quad \cancel{25} \\ \hline 1, 5 \\ \hline 2, 1 \end{array}$$



$$7 \Rightarrow \left. \begin{array}{l} 1, 5 \\ 2, 1 \end{array} \right\}$$

- It works best when applied to problems with the **greedy-choice** property:

- ie. a globally optimal solution can be arrived at by making a locally optimal (greedy) choice.
- When a locally optimal choice can lead to a globally optimal solution, the problem has this property.

- **Optimal Substructure property** : An optimal solution to the problem contains optimal solution to the sub problems.

# Making Change-Example 2



- **Problem:** (A dollar amount to reach and a collection of coin amounts to use to get there.) 32 ,
- **Configuration:** A dollar amount yet to return to a customer plus the coins already returned
- **Objective function:** Minimize number of coins returned.
- **Greedy solution:** Always return the largest coin you can ✓✓

# Making Change



- **Example 1:** Coins are valued \$0.32, \$0.08, \$0.01
  - Has the greedy-choice property, since no amount over \$0.32 can be made with a minimum number of coins by omitting a \$0.32 coin.
- **Example 2:** Coins are valued \$0.30, \$0.20, \$0.05, \$0.01
  - Does not have greedy-choice property, since \$0.40 is best made with two \$0.20's, but the greedy solution will pick three coins (which ones?)

2, 20s

1, 30  
2, 5



# Greedy Method



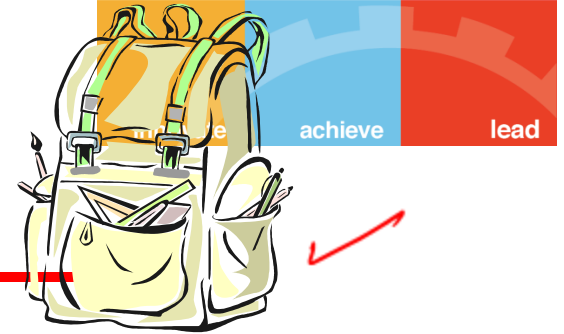
- An algorithm design strategy that always tries to find the best solution for each sub problem with the hope that it will yield a good solution for the problem as a whole.
- It makes the choice that looks best at the moment.
- Don't worry about the effect these choices have in the future.
- At each step the choice should be
  - Feasible → has to satisfy the problem constraints
  - Locally optimal → best choice at the time
  - Irrevocable → no change in subsequent steps

# Greedy Method



- Applied for optimization problems
- In order to solve a given optimization problem, we proceed by a sequence of choices.
- The sequence starts from some well-understood starting configuration, and then iteratively makes the decision that seems best from all of those that are currently possible.
- Greedy approach does not always lead to an optimal solution. It works optimally for problems with greedy-choice property

# The Fractional Knapsack Problem



- Given: A knapsack of capacity  $M$  and  $n$  objects of weights  $w_1, w_2, \dots, w_n$  with profits  $p_1, p_2, p_3, \dots, p_n$ .
- **Goal: Choose items with maximum total benefit but with weight at most  $M$ .**

# The Fractional Knapsack Problem



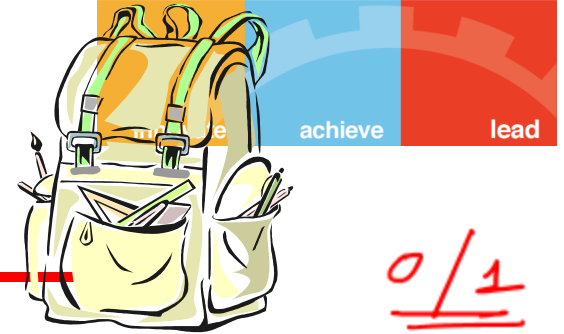
- If we are allowed to take fractional amounts, then this is the fractional knapsack problem.
- Let  $x_1, x_2, \dots, x_n$  be the fractions of objects that are supposed to be added to knapsack.
  - Maximize  $\sum_{i=1}^n p_i x_i$
  - Constraint :  $\sum_{i=1}^n w_i x_i \leq M$

# The Fractional Knapsack Problem



- Greedy choice:
  - **Keep taking item with highest value (profit to weight ratio)**
- Algorithm: Iteratively picks the item with the greatest value(  $p_i/w_i$  ).
- If, at the end, the knapsack cannot fit the entire last item with greatest value among the remaining items, we will take a fraction of it to fill the knapsack.

# The Fractional Knapsack Problem



0/1

- Let  $n=3$   $M=40$

Weight( $w_i$ )	Profit( $P_i$ )
20	30 ✓
25	40 ✓
10	35 ✓

① Arrange objects in the decreasing order of its value  
profit/weight

$$\frac{P_1}{w_1} = \frac{30}{20} = \underline{1.5}$$

$$\frac{P_2}{w_2} = \frac{40}{25} = 1.6$$

$$\frac{P_3}{w_3} = \frac{35}{10} = \underline{3.5}$$

$$0_3 > 0_2 > 0_1$$

# The Fractional Knapsack Problem



- Let  $n=3$   $M=40$

Weight( $w_i$ )	Profit( $P_i$ )
20	30
25	40 ✓
10 ✓	35 ✓

$$O_3 > O_2 > O_1$$

$obj(i)$	$w_i$	$P_i$	$x=1$ or $x = \frac{rc}{w_i}$	$Prof = x * P_i$	$rc = rc - w_i * x_i$ $rc = 40$
$O_3$	<u>10</u>	<u>35</u>	<u>1</u>	$1 \times 35 = 35$	$40 - 10 = \underline{30} \checkmark$
$O_2$	<u>25</u>	40	<u>1</u>	$1 \times 40 = 40$	$30 - 25 = \underline{5} \checkmark$
$O_1$	<u>20</u>	30	$\frac{5}{20} = \underline{0.25}$	$0.25 \times 30 = \underline{7.5}$	$5 - (20 \times 0.25) = \underline{0}$
				Total Profit = <u>82.5</u>	$\left. \begin{matrix} 1, O_3 \\ 1, O_2 \\ 0.25, O_1 \end{matrix} \right\}$

# The Fractional Knapsack Algorithm



## Algorithm

*fractionalKnapsack*( $m, n, w, x, p$ )

### Input:

$m$ -Capacity of Knapsack ,

$n$ - no:of objects,

$w$ -array of weights of all  $n$  objects,

$p$ -array of profits of all  $n$  objects

### Output:

$x$  -array containing the solution

$O(n \log n)$

Arrange the objects in the decreasing order of  $p_i/w_i$

for  $i \leftarrow 1$  to  $n$

$x[i] = 0$

end for

$rc \leftarrow m$

for  $i \leftarrow 1$  to  $n$

if ( $w[i] > rc$ ) break;

$x[i] \leftarrow 1$

$rc \leftarrow rc - w[i]$

//  $p \leftarrow p + p[i]$

end for

if ( $i \leq n$ )

{  $x[i] = rc/w[i]$  }

//  $p = (p[i] * x[i]) + p$

$\rightarrow O(n \log n)$  ✓

$O(n)$

$O(1)$

$O(n)$

$O(1)$



# The Fractional Knapsack Algorithm



- Time complexity
- **$O(n \log n)$**
- Why?
- Specifically, we use a heap-based priority queue to store the items of  $S$ , where the key of each item is its value. With this data structure, each greedy choice, which removes the item with greatest value, takes  $O(\log n)$  time. Process is repeated for  $n$  items. Hence  $O(n \log n)$

**Greedy Choice Property:** Let  $i$  be the index of the item with maximum  $p_i/w_i$ . Then there exists an optimal solution in which you take as much of item  $i$  as possible.

Given a set of  $n$  items  $\{1, 2, \dots, n\}$ .

- Assume items sorted by per-pound values:  $\rho_1 \geq \rho_2 \geq \dots \geq \rho_n$ .

Let the greedy solution be  $G = \langle x_1, x_2, \dots, x_k \rangle$

- $x_i$  indicates fraction of item  $i$  taken (all  $x_i = 1$ , except possibly for  $i = k$ ).

Consider any optimal solution  $O = \langle y_1, y_2, \dots, y_n \rangle$

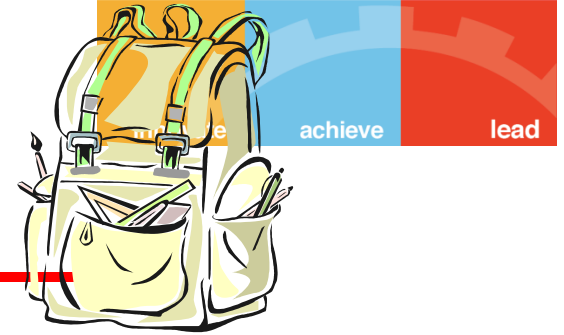
- $y_i$  indicates fraction of item  $i$  taken in  $O$  (for all  $i$ ,  $0 \leq y_i \leq 1$ ).
- Knapsack must be full in both  $G$  and  $O$ :  
$$\sum_{i=1}^n x_i w_i = \sum_{i=1}^n y_i w_i = K.$$

Consider the first item  $i$  where the two selections differ.

- By definition, solution  $G$  takes a greater amount of item  $i$  than solution  $O$  (because the greedy solution always takes as much as it can). Let  $x = x_i - y_i$ .

- There must exist some item  $j \neq i$  with  $p_j/w_j < p_i/w_i$  that is in the knapsack.
- We can therefore take a piece of  $j$ , with  $x$  weight, out of the knapsack, and put a piece of  $i$  with  $x$  weight in.
- This increases the knapsack's value by
- $x[p_i/w_i] - x[p_j/w_j] = x[(p_i/w_i) - (p_j/w_j)] > 0$
- **Contradiction** to the original solution being optimal.

# Applications



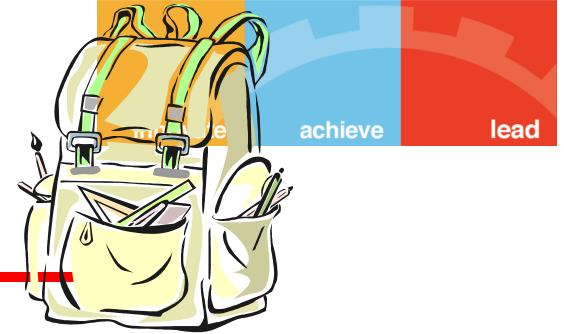
- Portfolio optimization:
- Cutting stock problems
- Huffman encoding for text compression } Compute high frequency characters with short code words.
- Web auction Optimization
- **K Means procedure in clustering**: It can be viewed as a greedy algorithm for partitioning the  $n$  examples into  $k$  clusters so as to minimize the sum of the squared distances to the cluster centers








# Web Auction Optimization

- Suppose you are designing a new online auction website that is intended to process bids for multi-lot auctions.
- This website should be able to handle a single auction for 100 units of the same digital camera or 500 units of the same smartphone, where bids are of the form, “ $x$  units for \$ $y$ ,” meaning that the bidder wants a quantity of  $x$  of the items being sold and is willing to pay \$ $y$  for all  $x$  of them.
- The challenge for your website is that it must allow for a large number of bidders to place such multi-lot bids and it must decide which bidders to choose as the winners.
- Naturally, one is interested in designing the website so that it always chooses a set of winning bids that maximizes the total amount of money paid for the items being auctioned. **So how do you decide which bidders to choose as the winners?**

# Exercise



Items:					
	1	2	3	4	5
Weight:	4 ml	8 ml	2 ml	6 ml	1 ml
Benefit:	\$12	\$32	\$40	\$30	\$50
Value: (\$ per ml)	3	4	20	5	50



"knapsack"

10 ml

Solution:

- 1 ml of 5
- 2 ml of 3
- 6 ml of 4
- 1 ml of 2



## Task Scheduling Problem

- Job sequencing With deadlines ✓
- Interval Scheduling ✓

# Job sequencing with deadlines

- Given a set of  $n$  jobs and associated with each job  $i$  is an integer deadline  $d_i \geq 0$  and profit  $p_i > 0$ , it is required to find the set of jobs such that all the chosen jobs should be completed within their deadlines and the profit earned should be maximum with the following constraints.
- Only one machine is available for processing jobs
- Only one job must be processed at any point of time
- A job is said to be completed if it is processed on a machine for one unit time.
- **Greedy choice property:** Arrange the job in decreasing order of profits and pick the highest profit job first.



# Job sequencing with deadlines



- Example

i	1	2	3	4	5
pi	20	10	15	1	5
di	2	1	2	3	3

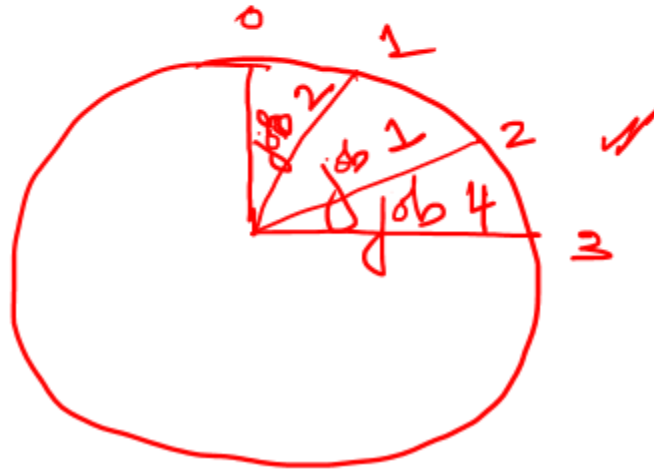
Arrange  
objects in  
decreasing  
order of  
profit

i	1	3	2	5	4
pi	20	15	10	5	1
di	2	2	1	3	3

# Job sequencing with deadlines

- Example

i	1	2	3	4	5
pi	20	15	10	5	1
di	2	2	1	3	3



$$20 + 15 + 5 = \underline{\underline{40}}$$



# Job sequencing with deadlines



**Algorithm JobSeq(job[1..n], p[1..n], d[1..n], list[1..n], n)**

→ Arrange the objects in decreasing order of profits. →  $n \log n$

Initialise list with 0s

Profit=0

for i=1 to n do

k=d[i]

while (k>0)

if(list[k]=0)

list[k]=job[i]

add p[i] to profit

break;

end if

decrement k by 1

end while

End for

Print list

$n \times k$

$n \times n$



$O(n^2)$

# Job sequencing with deadlines



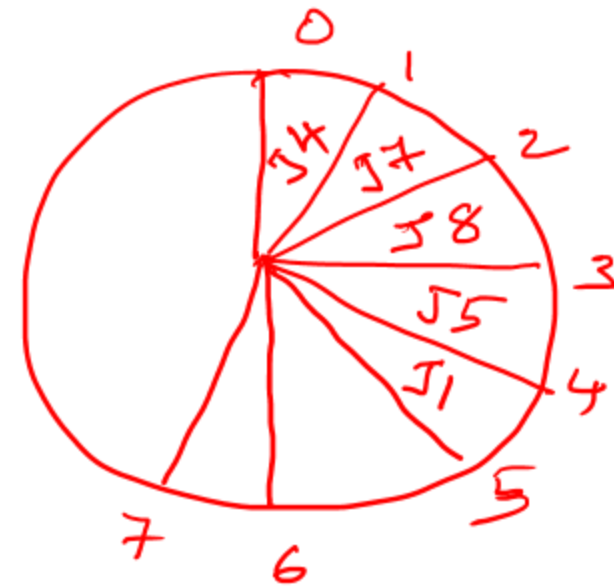
- Time complexity
- $O(n^2)$
- Why?

# Example



- Given a set of 9 jobs where each job has a deadline and profit associated to it. Each job takes 1 unit of time to complete and only one job can be scheduled at a time. We earn the profit if and only if the job is completed by its deadline. The task is to find the maximum profit and the number of jobs done.

Jobs	Profit	Deadline
J1	85	5
→ J2	25	4
J3	16	3
J4	40	3
J5	55	4
J6	19	5
J7 →	92	2
J8	80	3
J9	15	7



# Task Scheduling Problem

## [Interval Partitioning]



- Given: a set  $T$  of  $n$  tasks, each having:
  - A start time,  $s_i$
  - A finish time,  $f_i$  (where  $s_i < f_i$ )
- Goal: Perform all the tasks using a minimum number of “machines.”

# Task Scheduling Algorithm

## [Interval Partitioning]



- Suppose we are given a set  $T$  of  $n$  tasks, such that each task  $i$  has a start time,  $s_i$ , and a finish time,  $f_i$  (where  $s_i < f_i$ ). Task  $i$  must start at time  $s_i$  and it is guaranteed to be finished by time  $f_i$ .
- Each task has to be performed on a machine and each machine can execute only one task at a time.
- Two tasks  $i$  and  $j$  are nonconflicting if  $f_i \leq s_j$  or  $f_j \leq s_i$ .
- Two tasks can be scheduled to be executed on the same machine only if they are nonconflicting.

# Task Scheduling Problem

## [Interval Partitioning]

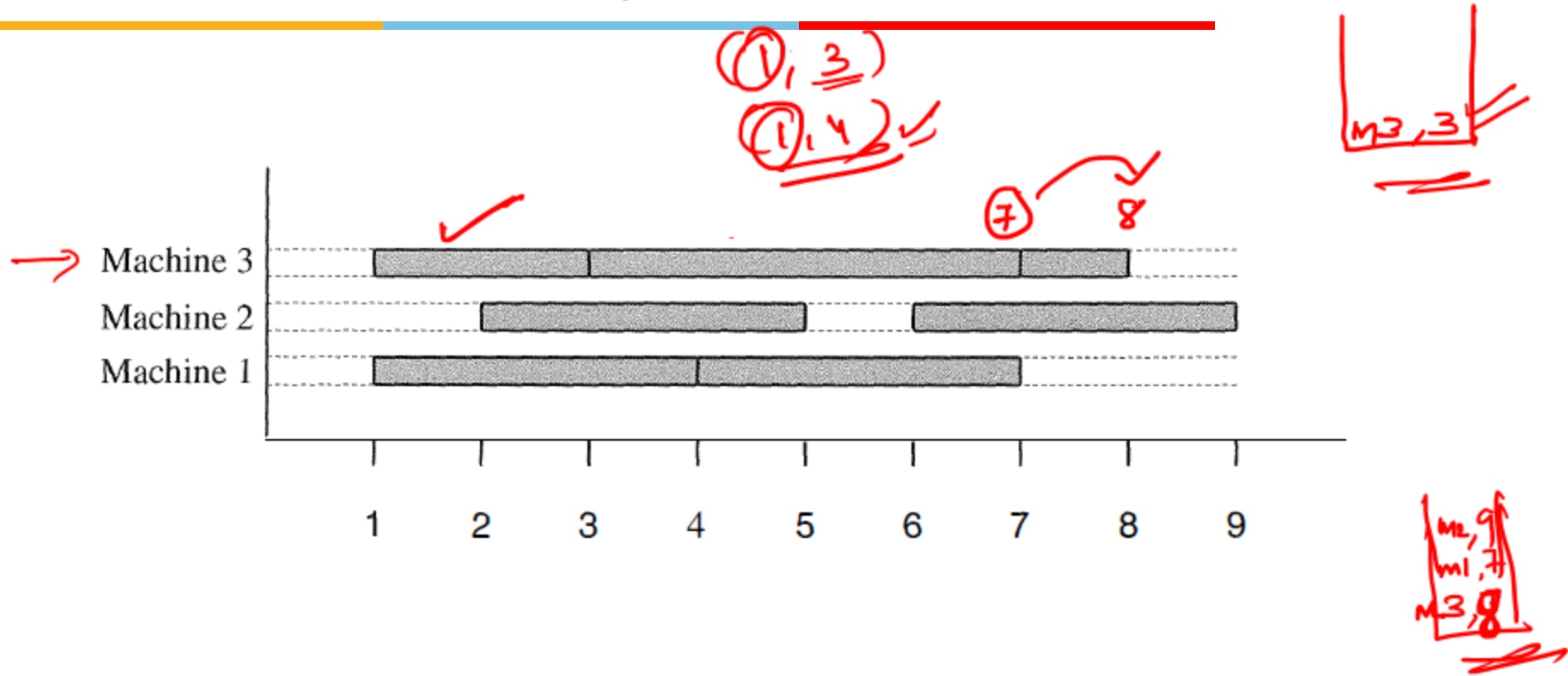


- The task scheduling problem we consider here is to schedule all the tasks in  $T$  on the fewest machines possible in a nonconflicting way. ✓
- Greedy choice. ✓
  - [Earliest start time] Consider tasks in ascending order of  $s_j$ .
  - [Earliest finish time] Consider tasks in ascending order of  $f_j$
  - [Shortest interval] Consider tasks in ascending order of  $f_j - s_j$ . ✓



# Task Scheduling Problem

## [Interval Partitioning]



An illustration of a solution to the task scheduling problem, for tasks whose collection of pairs of start times and finish times is  $\{ (1, 3), (1, 4), (2, 5), (3, 7), (4, 7), (6, 9), (7, 8) \}$ .

# Task Scheduling Problem

## [Interval Partitioning]-Algorithm



- Consider tasks in increasing order of start time:

**Algorithm** TaskSchedule( $T$ ):

**Input:** A set  $T$  of tasks, such that each task has a start time  $s_i$  and a finish time  $f_i$

**Output:** A nonconflicting schedule of the tasks in  $T$  using a minimum number of machines

$m \leftarrow 0$  {optimal number of machines}

**while**  $T \neq \emptyset$  **do**

remove from  $T$  the task  $i$  with smallest start time  $s_i$

**if** there is a machine  $j$  with no task conflicting with task  $i$  **then**

schedule task  $i$  on machine  $j$

**else**

$m \leftarrow m + 1$  {add a new machine}

schedule task  $i$  on machine  $m$

$$\begin{cases} \underline{f_i} \leq s_j \\ \underline{f_j} \leq s_i \end{cases}$$

# Task Scheduling Problem [Interval Partitioning]-



- Time complexity :  $O(n \log n)$
- Sorting by start times takes  $O(n \log n)$  time.
- Store machines in a priority queue (key = finish time of its last job).
- To allocate a new machine, INSERT machine onto priority queue.
- To schedule Task  $j$  in machine  $k$ , INCREASE-KEY of classroom  $k$  to  $f_j$ .
- To determine whether Task  $j$  is compatible with any machine, compare  $s_j$  to FIND-MIN.
- Total # of priority queue operations is  $O(n)$ ; each takes  $O(\log n)$  time.
- This implementation chooses a machine  $k$  whose finish time of its last job is the earliest.

# Sample problem1-Meeting Room Scheduling



- Schedule the meetings in as few conference rooms as possible.
- *Given a list of meeting times, checks if any of them overlaps. The follow-up question is to return the minimum number of rooms required to accommodate all the meetings.*
- Suppose we use interval (start, end) to denote the start and end time of the meeting, we have the following meeting times: (1, 4), (5, 6), (8, 9), (2, 6).

*(1,4) (2,6), (5,6), (8,9)*

# Sample problem1



- In the above example, apparently we should return true for the first question since (1, 4) and (2, 6) have overlaps.
- For the follow-up question, we need at least 2 meeting rooms so that (1, 4), (5, 6) will be put in one room and (2, 6), (8, 9) will be in the other.

# Sample problem2

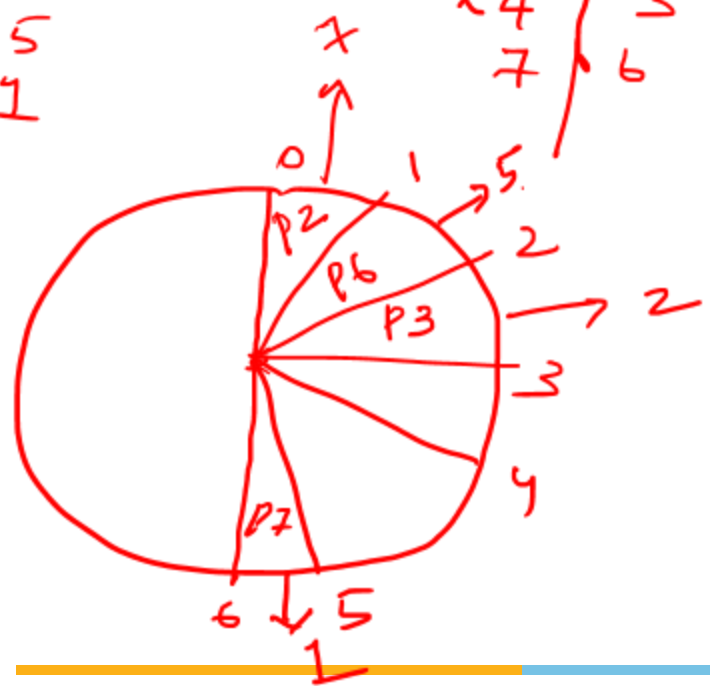


- A teacher assign homework at the beginning of the first day of class. Each problem carries 1 mark + bonus if submitted within specified number of days. The deadline for submitting and bonus marks are different for each problem. Each problem takes exactly one day to complete. Maximum bonus one can obtain is 15. Your task is to find a schedule to complete all homework problems so as to maximize bonus marks..

Problem number	1	2	3	4	5	6	7
Deadline for bonus	1	1	3	3	2	2	6
Bonus	6	7	2	1	4	5	1

P.no	D	Bonus
1	2	6
2	1	7
3	3	2
4	3	1
5	2	4
6	2	5
7	6	1

P.no	D	Bonus
✓2	1	7
x1	1	6 → X
✓6	2	5
x5	2	4 → X
✓3	3	2
x4	3	1 } → X
7	6	1 }



$$\begin{aligned}
 &5 + 7 + 2 + 1 \\
 &= 15 \\
 &\underline{\underline{14}}
 \end{aligned}$$

# Sample problem3



- A large ship is to be loaded with cargo. The cargo is containerized, and all containers are the same size. Different containers may have different weights. Let  $w_i$  be the weight of the  $i$ th container,  $1 \leq i \leq n$ . The cargo capacity of the ship is  $c$ . We wish to load the ship with the maximum number of containers.



# Sample problem3



Formulation of the problem :

- Variables :  $x_i$  ( $1 \leq i \leq n$ ) is set to 0 if the container  $i$  is not to be loaded and 1 in the other case.
- Constraints :  $\sum_{i=1}^n w_i x_i \leq c.$
- Optimization function :  $\sum_{i=1}^n x_i$

# Sample problem3



- Using the greedy algorithm the ship may be loaded in stages; one container per stage.
- At each stage, the greedy criterion used to decide which container to load is the following :
  - *From the remaining containers, select the one with least weight.*
- This order of selection will keep the total weight of the selected containers minimum and hence leave maximum capacity for loading more containers.

# Sample problem3

- Suppose that :
  - $n = 8$ ,
  - $[w_1, \dots, w_8] = [100, 200, 50, 90, 150, 50, 20, 80]$ ,
  - and  $c = 400$ .
- When the greedy algorithm is used, the containers are considered for loading in the order 7,3,6,8,4,1,5,2.
- Containers 7,3,6,8,4 and 1 together weight 390 units and are loaded.
- The available capacity is now 10 units, which is inadequate for any of the remaining containers.
- In the greedy solution we have :

$$[x_1, \dots, x_8] = [1, 0, 1, 1, 0, 1, 1, 1] \text{ and } \sum x_i = 6.$$

# Sample problem4



- Lecture  $j$  starts at  $s_j$  and finishes at  $f_j$ .
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.