

# Design and Analysis of Algorithms, MTech-I (1<sup>st</sup> semester)

## Chapter 3: Greedy Algorithm Design Technique - I

September 7, 2022



Devesh C Jinwala,

Professor in CSE, SVNIT, Surat and Adjunct Professor, IIT Jammu & Dean (R&C), SVNIT  
Department of Computer Science and Engineering, SVNIT, Surat

1 Introduction

2 Motivating Case study

3 Applications

# Contents

- The basic paradigm

# Contents

- The basic paradigm
- The greedy control abstraction

# Contents

- The basic paradigm
- The greedy control abstraction
- Elements of greedy strategy

# Contents

- The basic paradigm
- The greedy control abstraction
- Elements of greedy strategy
- Characteristics

# Contents

- The basic paradigm
- The greedy control abstraction
- Elements of greedy strategy
- Characteristics
- Some optimization problems

# Contents

- The basic paradigm
- The greedy control abstraction
- Elements of greedy strategy
- Characteristics
- Some optimization problems
- Applications



# Contents

- The basic paradigm
- The greedy control abstraction
- Elements of greedy strategy
- Characteristics
- Some optimization problems
- Applications
- Greedy technique to various problems

# Introduction

## Michael Douglas in the Wall Street

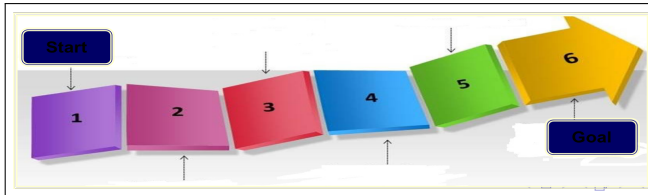
Greed is good, greed is right, greed works!!!

- Let us try to verify whether is it really so..... How ?
- Take a number of computational problems investigate the pros & cons of short-sighted greed
- How to define a greedy algorithm ?

# The Basic Greedy Criterion

## In a greedy algorithm design

- build up the small solution in small steps, while working in stages to optimize some underlying criterion
- Selection of the next step is myopic and irreversible ....????
- we build up the small solution in small steps..... while working in stages
- The goal is to optimize some underlying criterion
- Selection of the next step is myopic and irreversible



# The Basic Greedy criterion...

## Requirements

- The next step selected must ensure feasibility. How ?
- Such selection must lead to an optimal solution. How ?
- What are optimization problems ?

# Terminologies

## Greedy Design terms

- greedy criterion
- optimal solution
- feasible solution
- suboptimal solution
- constraints
- optimization problems
- heuristics
- bounded performance
- approximation algorithms

# The Thirsty Baby problem

## The Problem

An intelligent baby wants to quench her thirst. . . .!!! She has a defined number of liquids available in a defined amount, to her disposal, each with a defined satisfaction quotient.

Her capacity to drink all the liquids in all, is bounded.

The objective is to maximize her thirst, while drinking a combination of liquids. . . . .

- How to formalize the problem description ?
- Using the mathematical notations how do we state : the inputs, outputs, constraint function, optimizing function

# Greedy algorithm characteristic

## Irrevocability

What is irrevocability in this case ?

# The Container Loading Problem

## The Problem description

A Cargo Train Bogey is to be loaded with containers each having a specific weight, so as to maximize the no of containers, such that the maximum weight carrying capacity of the bogey is not exceeded.....



# The Container Loading Problem

## The Problem description

A Cargo Train Bogey is to be loaded with containers each having a specific weight, so as to maximize the no of containers, such that the maximum weight carrying capacity of the bogey is not exceeded.....

## The Formalism

- Let  $w_i$  - the weight of the container  $i$
- Let  $C$  - the maximum cargo carrying capacity of a ship
- Let  $x_i$  be a boolean variable (1=container loaded , 0= container not loaded)
- The problem is to assign values  $x_i$  such that  $\sum_{i=1}^n (x_i)$  is maximized subject to the constraint  $\sum_{i=1}^n (w_i) \leq C$

# The Container Loading Problem: An illustration

- Given that  $n = 8$ ,  $i = [1,2,3,4,5,6,7,8]$ ,  $\Sigma w_i = [100,200, 50, 90, 150, 50, 20, 80]$ ,  $C = 400$
- What is the order of loading ?
- What is the  $\Sigma x_i$ ?

# The Container Loading Problem: Algorithm

```
1.      for i=1 to n
2.          x[i] = 0
3.      t ← ALLOCATE_MEMORY(n)
4.      IndirectSort(w, t, n)
5.      while (i <=n) and w[t[i]] ≤ C
6.          x[t[i]] = 1
7.          C = C - w[t[i]]
8.          i=i+1
9.      delete t
```

# Combinatorial Optimization problems

- Broadly the problem of finding a solution that either minimizes or maximizes the value of a particular parameter, is always subject to certain constraints.
- Combinatorial Optimization problems
  - if the parameter to be optimized is discrete such as an integer, a permutation or graph from a finite (or possibly countable infinite) set.

# Combinatorial Optimization Problems

- Formally, a combinatorial optimization problem is a quadruple viz.  $\langle I, f, m, g \rangle$ , where
  - $I$  - is a set of problem instances
  - $f(x)$  - is the set of feasible solutions, given a specific problem instance  $x \in I$
  - $m(x, y)$  - given an instance  $x$  and a feasible solution of  $x$ ,  $m(x, y)$  denotes the measure of  $y$ , which is usually a positive real.
  - $g$  is the goal function, and is either min or max

# Combinatorial Optimization Problems

- Formally, a combinatorial optimization problem is a quadruple viz.  $\langle I, f, m, g \rangle$ , where
  - $I$  - is a set of problem instances
  - $f(x)$  - is the set of feasible solutions, given a specific problem instance  $x \in I$
  - $m(x, y)$  - given an instance  $x$  and a feasible solution of  $x$ ,  $m(x, y)$  denotes the measure of  $y$ , which is usually a positive real.
  - $g$  is the goal function, and is either min or max
- The goal is then to find for some instance  $x$ ,

# Combinatorial Optimization Problems

- Formally, a combinatorial optimization problem is a quadruple viz.  $\langle I, f, m, g \rangle$ , where
  - $I$  - is a set of problem instances
  - $f(x)$  - is the set of feasible solutions, given a specific problem instance  $x \in I$
  - $m(x, y)$  - given an instance  $x$  and a feasible solution of  $x$ ,  $m(x, y)$  denotes the measure of  $y$ , which is usually a positive real.
  - $g$  is the goal function, and is either min or max
- The goal is then to find for some instance  $x$ ,
  - an optimal solution, that is, a feasible solution with  $m(x, y) = g\{m(x, y') \mid y' \in f(x)\}$

# Combinatorial Optimization Problems

- Formally, a combinatorial optimization problem is a quadruple viz.  $\langle I, f, m, g \rangle$ , where
  - $I$  - is a set of problem instances
  - $f(x)$  - is the set of feasible solutions, given a specific problem instance  $x \in I$
  - $m(x, y)$  - given an instance  $x$  and a feasible solution of  $x$ ,  $m(x, y)$  denotes the measure of  $y$ , which is usually a positive real.
  - $g$  is the goal function, and is either min or max
- The goal is then to find for some instance  $x$ ,
  - an optimal solution, that is, a feasible solution with  $m(x, y) = g\{m(x, y') \mid y' \in f(x)\}$
- The goal is then to find for some instance  $x$ , an optimal solution, that is, a feasible solution with  $m(x, y) = g\{m(x, y') \mid y' \in f(x)\}$



# The Knapsack Problem

- is a specialization of the Container loading problem

## definition

Given  $n$  objects each with weight  $w_i$  and a value  $v_i$  a knapsack with maximum weight carrying capacity  $W$ , the goal is to optimize the value

$$\sum_{i=1}^n x_i * v_i, \quad 1 \leq i \leq n$$

such that

$$\sum_{i=1}^n w_i * x_i \leq W$$

- What could be the domain of the values  $x_i$  ?
- How is the output optimization function and the  $W$  related ?

# The Fractional Knapsack Problem : Illustrations

## Instance 1

- Let  $n = 5$ ,  $W = 100$  and  $w[i] = \langle 10 \ 20 \ 30 \ 40 \ 50 \rangle$   
 $v[i] = \langle 20 \ 30 \ 66 \ 40 \ 60 \rangle$
- What are the values of

$$\sum_{i=1}^5 w[i]$$

and

$$\sum_{i=1}^5 v[i]$$

?

- Compute the values for the *Maximum-Value-first*, *Minimum-Weight-first*, and *Maximum-value-density-per-weight-first* approaches ?
- What is the optimal answer ?
- What should be the correct approach to solve the problem ?

# The 0/1 Knapsack Problem : Illustrations

## Instance 1

- $i=[1\ 2\ 3]$   $v = \langle 20\ 15\ 15 \rangle$   $w = \langle 100\ 10\ 10 \rangle$  and  $W = 105$
  - $i = [1\ 2]$   $v = \langle 10\ 20 \rangle$   $w = \langle 5\ 100 \rangle$  and  $W = 25$
  - $i=[1\ 2\ 3]$   $v = \langle 20\ 15\ 15 \rangle$   $w = \langle 40\ 25\ 25 \rangle$  and  $W = 30$
- 
- Apply minimum weight/max value density criterion
  - Apply minimum weight criterion
  - Apply maximum value density criterion
  - Compare with the optimal solutions
  - What is the inference to be drawn ?

# The Greedy Control Abstraction

```
Algorithm Greedy(Type a[], int n)
1. solution = EMPTY;
2. i=1;
3. for i=1 to n
4.     Type x = select(a);
5.     if feasible(solution, x)
6.         solution=solution U x;
7. return solution
```

# Applications

- Optimal solutions
  - simple scheduling problems
  - change making
  - Minimum Spanning Tree (MST)
  - Single-source shortest paths
  - Huffman codes
- Approximations
  - Traveling Salesman Problem (TSP)
  - Knapsack problem
  - other combinatorial optimization problems

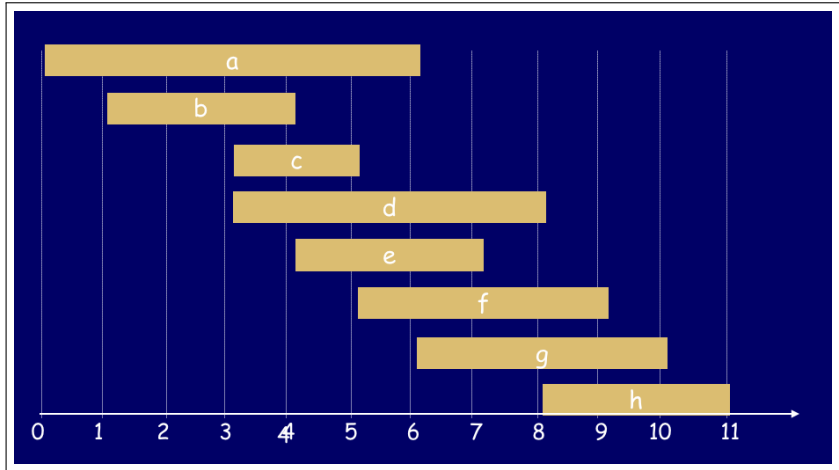
# Scheduling#1: Simple Activity Selection

Also is a type of Interval Scheduling

## The Problem

- Given  $n$  activities each with a defined start time  $s_i$  and finish time  $f_i$ , the problem is to select a maximal set of mutually compatible activities
- Mutually compatible activities: if each activity  $i$  occurs during the half open interval  $[s_i, f_i)$ , then they are compatible if,  $[s_i, f_i)$  and  $[s_j, f_j)$  do not overlap ?
- When do  $[s_i, f_i)$  and  $[s_j, f_j)$  not overlap ?

# Activity Selection Illustration



# Greedy Choices/Variations

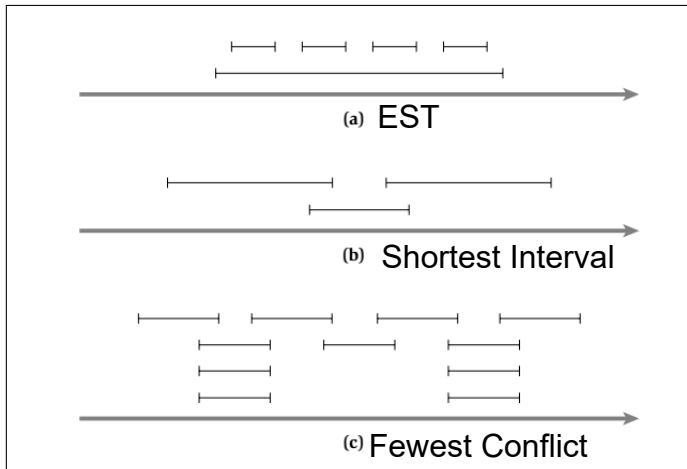
Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

- Earliest start time - Consider jobs in ascending order of start time  $s_j$ .
- Earliest finish time - Consider jobs in ascending order of finish time  $f_j$ .
- Shortest interval - Consider jobs in ascending order of interval length  $f_j - s_j$ .
- Fewest conflicts - For each job, count the number of conflicting jobs  $c_j$ .  
Schedule in ascending order of conflicts  $c_j$ .

Which one of these strategies work ?



# Failure Cases



# Dry-run of the algorithm

Dry-run

Animation in the PPT demointervalscheduling.ppt

# Solution approach

- Greedy algorithm choices
- Consider the jobs in increasing order of finish time.
- Take each job provided it is compatible with the ones already taken.

```
Algorithm Simple_ActivitySelection(Job,  $s_i$ ,  $f_i$ )  
/*A=Set of selected mutually compatible jobs*/  
1.Sort jobs by finish times so that  $f_1 \leq f_2 \leq \dots \leq f_n$ .  
2.  $A \leftarrow \phi$   
3. for  $j = 1$  to  $n$   
4.     if  $f_j \leq s_{(j+1)}$  /*(job (j+1) is compatible with A)*/  
5.      $A = A \cup \{j\}$   
6. return Selected_Jobs
```

## Complexity

Time taken by the algorithm to execute?

# Analysis and Proof: Why the algorithm works?: Proof by Contradiction

- Proof: (by contradiction) .....i.e.  $m > k$  in the following:

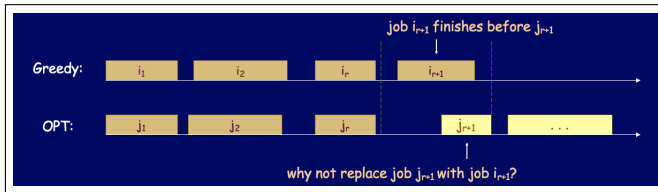


Figure: There is a request  $j_{k+1}$  in the possible set of job requests after the job  $i_k$  ends?

# Analysis and Proof: Why the algorithm works?: Proof by Contradiction

- Proof: (by contradiction) .....i.e.  $m > k$  in the following:
  - Assume greedy is not optimal, and let's see what happens.

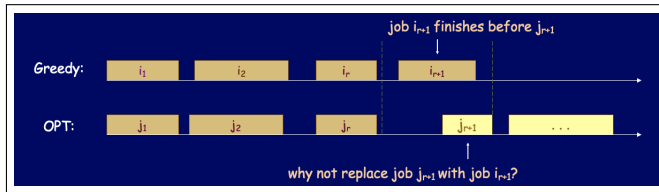


Figure: There is a request  $j_{k+1}$  in the possible set of job requests after the job  $i_k$  ends?

# Analysis and Proof: Why the algorithm works?: Proof by Contradiction

- Proof: (by contradiction) .....i.e.  $m > k$  in the following:
  - Assume greedy is not optimal, and let's see what happens.
  - Let  $i_1, i_2, \dots, i_k$  denote a set of jobs selected by greedy.

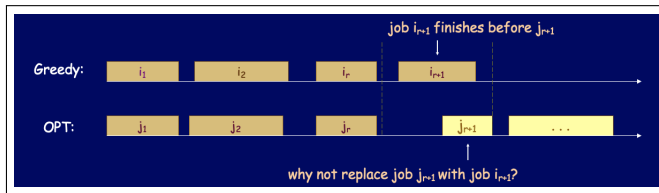


Figure: There is a request  $j_{k+1}$  in the possible set of job requests after the job  $i_k$  ends?

# Analysis and Proof: Why the algorithm works?: Proof by Contradiction

- Proof: (by contradiction) .....i.e.  $m > k$  in the following:
  - Assume greedy is not optimal, and let's see what happens.
  - Let  $i_1, i_2, \dots, i_k$  denote a set of jobs selected by greedy.
  - Let  $j_1, j_2, \dots, j_m$  denote a set of jobs in the optimal solution with  $i_1=j_1, i_2=j_2, \dots, i_r=j_r$  for the largest possible value of  $r$ .

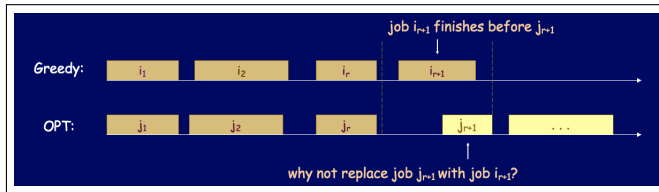


Figure: There is a request  $j_{k+1}$  in the possible set of job requests after the job  $i_k$  ends?

# Analysis and Proof: Why the algorithm works?: Proof by Contradiction

- Proof: (by contradiction) .....i.e.  $m > k$  in the following:
  - Assume greedy is not optimal, and let's see what happens.
  - Let  $i_1, i_2, \dots, i_k$  denote a set of jobs selected by greedy.
  - Let  $j_1, j_2, \dots, j_m$  denote a set of jobs in the optimal solution with  $i_1=j_1, i_2=j_2, \dots, i_r=j_r$  for the largest possible value of  $r$ .
  - Note that  $m > k$

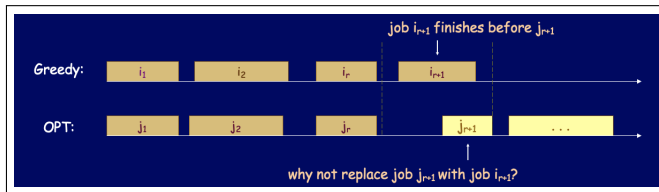


Figure: There is a request  $j_{k+1}$  in the possible set of job requests after the job  $i_k$  ends?



# Analysis and Proof: Why the algorithm works?: Proof by Contradiction

- Proof: (by contradiction) .....i.e.  $m > k$  in the following:
  - Assume greedy is not optimal, and let's see what happens.
  - Let  $i_1, i_2, \dots, i_k$  denote a set of jobs selected by greedy.
  - Let  $j_1, j_2, \dots, j_m$  denote a set of jobs in the optimal solution with  $i_1=j_1, i_2=j_2, \dots, i_r=j_r$  for the largest possible value of  $r$ .
  - Note that  $m > k$
  - How are  $f(i_k)$  and  $f(j_k)$  related?

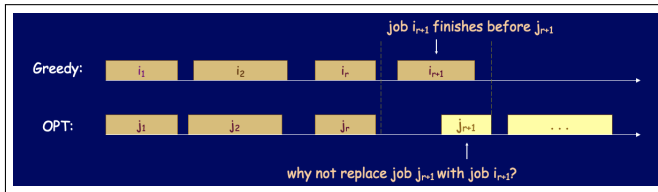


Figure: There is a request  $j_{k+1}$  in the possible set of job requests after the job  $i_k$  ends?

# Analysis and Proof: Why the algorithm works?: Proof by Contradiction

- Proof: (by contradiction) .....i.e.  $m > k$  in the following:
  - Assume greedy is not optimal, and let's see what happens.
  - Let  $i_1, i_2, \dots, i_k$  denote a set of jobs selected by greedy.
  - Let  $j_1, j_2, \dots, j_m$  denote a set of jobs in the optimal solution with  $i_1=j_1, i_2=j_2, \dots, i_r=j_r$  for the largest possible value of  $r$ .
  - Note that  $m > k$
  - How are  $f(i_k)$  and  $f(j_k)$  related?
  - The possible set of job requests still contains the request  $f(j_{k+1})$  after all the requests  $i_1, i_2, \dots, i_k$  end or are deleted. Why ?

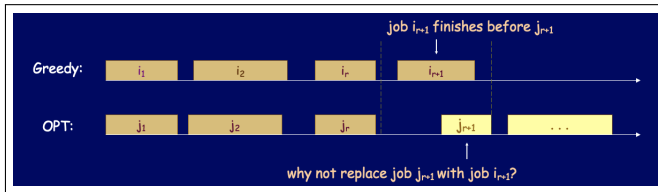


Figure: There is a request  $j_{k+1}$  in the possible set of job requests after the job  $i_k$  ends ?

## Analysis and Proof: Why the algorithm works?

- For the interval scheduling problem, let the set  $A = i_1, i_2, i_3, \dots, i_k$  with  $|A| = k$  be the set of intervals returned by the algorithm as the answer and

## Analysis and Proof: Why the algorithm works?

- For the interval scheduling problem, let the set  $A = i_1, i_2, i_3, \dots, i_k$  with  $|A| = k$  be the set of intervals returned by the algorithm as the answer and
- Let the set  $O = j_1, j_2, j_3, \dots, j_m$  be the optimal set of intervals that would be returned by an oracle.

# Analysis and Proof: Why the algorithm works?

- For the interval scheduling problem, let the set  $A = i_1, i_2, i_3, \dots, i_k$  with  $|A| = k$  be the set of intervals returned by the algorithm as the answer and
- Let the set  $O = j_1, j_2, j_3, \dots, j_m$  be the optimal set of intervals that would be returned by an oracle.
- What should be our goal to prove that our interval scheduling algorithm is optimal ?

# Analysis and Proof: Why the algorithm works?

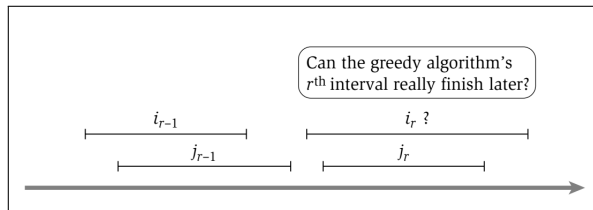
- For the interval scheduling problem, let the set  $A = i_1, i_2, i_3, \dots, i_k$  with  $|A| = k$  be the set of intervals returned by the algorithm as the answer and
- Let the set  $O = j_1, j_2, j_3, \dots, j_m$  be the optimal set of intervals that would be returned by an oracle.
- What should be our goal to prove that our interval scheduling algorithm is optimal ?
- The goal is to prove that  $k = m$  i.e. to prove that the  $r^{th}$  accepted request in the algorithm's finishes no later than the  $r^{th}$  request in the optimal schedule.

# Analysis and Proof: Why the algorithm works?

- For the interval scheduling problem, let the set  $A = i_1, i_2, i_3, \dots, i_k$  with  $|A| = k$  be the set of intervals returned by the algorithm as the answer and
- Let the set  $O = j_1, j_2, j_3, \dots, j_m$  be the optimal set of intervals that would be returned by an oracle.
- What should be our goal to prove that our interval scheduling algorithm is optimal ?
- The goal is to prove that  $k = m$  i.e. to prove that the  $r^{th}$  accepted request in the algorithm's finishes no later than the  $r^{th}$  request in the optimal schedule.
- What is this equivalent to proving ?

# Analysis and Proof by Mathematical Induction

- For the interval scheduling problem, let the set  $A = i_1, i_2, i_3, \dots, i_k$  with  $|A| = k$  be the set of intervals returned by the algorithm as the answer and
- Let the set  $O = j_1, j_2, j_3, \dots, j_m$  be the optimal set of intervals that would be returned by an oracle.



- How are  $fj_{(r-1)}$  and  $sj_{(r-1)}$  related ?
- How are  $fi_{(r-1)}$  and  $sj_r$  related ?



# Machine scheduling problem II

## Minimize Average Completion time

Jobs  $j_1, j_2, j_3, \dots, j_n$  with running times  $t_1, t_2, t_3, \dots, t_n$  to be scheduled on a single processor such as to minimize the avg completion time

Process ID	Execution Time in units
$J_1$	15
$J_2$	8
$J_3$	3
$J_4$	10

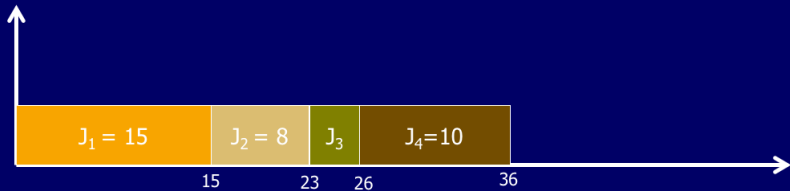
How to implement the above algorithm ? What is the time complexity?

## Machine scheduling problem II...

### Minimize Average Completion time

Jobs  $j_1, j_2, j_3, \dots, j_n$  with running times  $t_1, t_2, t_3, \dots, t_n$  to be scheduled on a single processor such as to minimize the avg completion time

#### Schedule 1:



# Shortest Job First Scheduling

## Proof of the algorithm

Prove that the shortest job first assignment for  $J$ Jobs  $j_1, j_2, j_3, \dots, j_n$  with running times  $t_1, t_2, t_3, \dots, t_n$  to be scheduled on a single processor such as to minimize the avg completion time is an optimal assignment.

# Machine scheduling problem III

## Minimum Average Completion time with multiprocessors

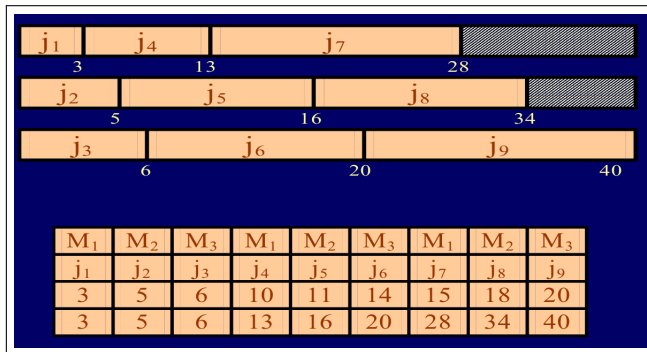
- Jobs  $j_1, j_2, j_3, \dots, j_n$  with running times  $t_1, t_2, t_3, \dots, t_n$  to be scheduled on multiprocessors such as to minimize the average completion time

$j_1$	$j_2$	$j_3$	$j_4$	$j_5$	$j_6$	$j_7$	$j_8$	$j_9$
3	5	6	10	11	14	15	18	20

# Machine scheduling problem III

## Minimum Average Completion time with multiprocessors

- Jobs  $j_1, j_2, j_3, \dots, j_n$  with running times  $t_1, t_2, t_3, \dots, t_n$  to be scheduled on multiprocessors such as to minimize the average completion time



# Machine Scheduling Problem IV

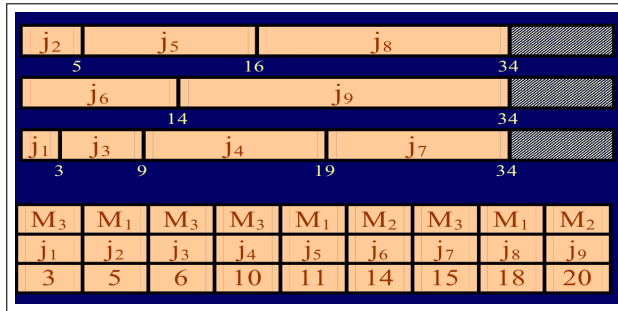
## Minimizing the Final Completion time with multiprocessors

- Jobs  $j_1, j_2, j_3, \dots, j_n$  with running times  $t_1, t_2, t_3, \dots, t_n$  to be scheduled on multiprocessors such that
    - no machine processes more than one process at a time
    - no process is executed by more than one machine
    - there is non-preemptive scheduling
    - the final completion time is minimized.
- What is the final completion time ?
- How to solve this for the given jobmix ?

$j_1$	$j_2$	$j_3$	$j_4$	$j_5$	$j_6$	$j_7$	$j_8$	$j_9$
3	5	6	10	11	14	15	18	20

# Machine Scheduling Problem IV...

Understanding what is Final Completion time....



- What is the final completion time for this job mix?
- How to solve this ?

# Machine Scheduling Problem IV...

## Minimizing the Final Completion time with multiprocessors

- What is the final completion time for this job mix?

j <sub>1</sub>	j <sub>2</sub>	j <sub>3</sub>	j <sub>4</sub>	j <sub>5</sub>	j <sub>6</sub>	j <sub>7</sub>	j <sub>8</sub>	j <sub>9</sub>
3	5	6	10	11	14	15	18	20

- What is the final completion time for this job mix?
- How to solve this ?



# Minimizing the final completion time

The problem is NP-hard

- No polynomial time algorithm to run in  $O(n^k m^l)$  for any constants  $k$  and  $l$
- With an approximation algorithm the schedule lengths are though not optimal but at the most  $(\frac{4}{3} - \frac{1}{3m})$  of the optimal schedule

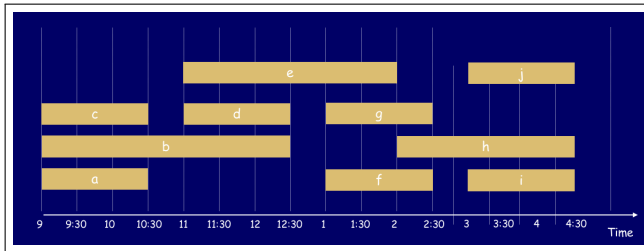
# Review

Reviewing the summary of the scheduling variations discussed

# Machine Scheduling V

## Scheduling a class time table

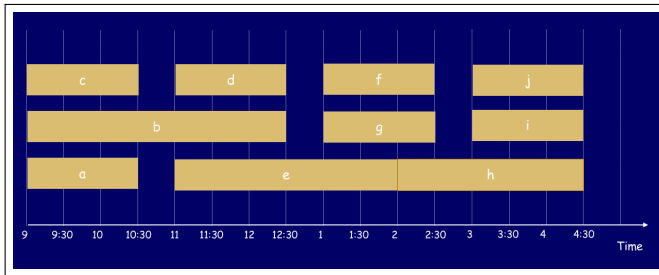
- But the goal is to minimize the number of classrooms used
- Assume that a lecture  $j$  starts at  $s_j$  and finishes at  $f_j$ , then the goal is to find the minimum number of classrooms to schedule all lectures, so that no two occur at the same time in the same room.
- An example schedule with 4 classrooms to schedule 10 lectures



# Machine Scheduling V...

## Scheduling a class time table

- Lecture  $j$  starts at  $s_j$  and finishes at  $f_j$ .
- This schedule uses only 3.



# Machine Scheduling V...

## One Solution approach

- Arrange the lectures in their increasing order of start times
- Keep track of availability times of classrooms i.e. let availability time of a classroom be  $M1$ .
- Then, assign the lecture  $i_1$ , completing at time  $t_1$ , mark its availability time as  $t_1$  and check compatibility, when scheduling the next lecture  $i_2$

# Machine Scheduling V...

A typical schedule and approach

task	start_ time	finish_ time	time required	availability time now on respective classroom	scheduling order
A	0	2	2	2(M1)	1
B	3	7	4	7(M1)	3
C	4	7	3	7(M3)	4
D	9	11	2	11(M3)	7
E	7	10	3	10(M1)	6
F	1	5	4	5(M2)	2
G	6	8	2	8(M2)	5

# Solution strategy

## Algorithm approach

- Consider lectures in increasing order of start time
- assign lecture to any compatible classroom.
- For each classroom  $k$ , maintain the finish time of the last job added.
- Keep the classrooms in a priority queue.

# Solution approach

Algorithm Classroom\_Scheduling(Interval[], s[], f[])

1. Sort intervals by starting time so that  $s_1 \leq s_2 \leq s_3 \leq \dots s_{(n-1)} \leq s_n$
2.  $d \leftarrow 0$
3. for  $j = 1$  to  $n$
4. if (lecture  $j$  is compatible with some classroom  $k$ )
5.     schedule lecture  $j$  in classroom  $k$
6. else
7.     allocate a new classroom  $d + 1$
8.     schedule lecture  $j$  in classroom  $d + 1$
9.      $d \leftarrow (d + 1)$

## Complexity

Time taken by the algorithm to execute?



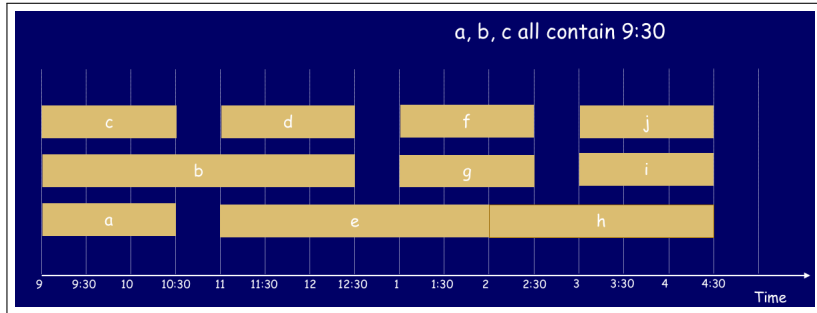
# IP: Lower Bound on Optimal Solution

- def: The depth of a set of open intervals is the maximum number of the intervals contained, at a unique time
- Key observation: Prove that the number of resources (classrooms) needed is at least the depth.

# Algorithm Correctness

## Theorem

IF we use the greedy algorithm above, every lecture will be assigned a classroom and no two overlapping lectures will receive the same classroom



# The Optimal Tape Storage Problem

- Given  $n$  files of length  $m_1, m_2, m_3, m_4, \dots, m_n$  find the best order in which the files can be stored on a sequential storage device.
- e.g. if  $n=3$  and  $m_1=5, m_2=10$  and  $m_3=3$
- There can be  $3!$  possible orderings.
- Which one is the best ?

# Scheduling to minimize the lateness

- Single resource processes one job at a time.
- Job  $j$  requires  $t_j$  units of processing time and is due at time  $d_j$ . If  $j$  starts at time  $s_j$ , it finishes at time  $f_j = s_j + t_j$ .
- Lateness:  $l_j = \max 0, f_j - d_j$ .
- Goal: schedule all jobs to minimize the maximum lateness  $L = \text{maximum lateness}$

■ e.g.

	1	2	3	4	5	6
$t_j$	3	2	1	4	3	2
$d_j$	6	8	9	9	14	15



Figure: Lateness of different jobs in a schedule

# How to minimize maximum lateness ?

Greedy template. Consider jobs in some order.

- Shortest processing time first: Consider jobs in ascending order of processing time  $t_j$ .
- Earliest deadline first: Consider jobs in ascending order of deadline  $d_j$ .
- Smallest slack: Consider jobs in ascending order of slack  $d_j - t_j$ .

# Minimizing lateness: Counterexamples

Shortest processing time first does not work

	1	2
$t_j$	1	10
$d_j$	100	10

Figure: Can we and if so how can one achieve lateness 0 above?

Shortest slack time first does not work

	1	2
$t_j$	1	10
$d_j$	2	10

Figure: Can we and if so how can one achieve lateness 0 above?

# Solution approach

Algorithm EarliestDeadlineFirst( $Job[]$ ,  $s[]$ ,  $f[]$ )

1. Sort  $n$  jobs by deadline so that  $d_1 \leq d_2 \leq \dots \leq d_n$
2.  $t \leftarrow 0$
3. for  $j = 1$  to  $n$
4.     Assign job  $j$  to interval  $[t, t + t_j]$
5.      $s_j \leftarrow t$ ,  $f_j \leftarrow t + t_j$ ,  $t \leftarrow t + t_j$
6. output intervals  $[s_j, f_j]$

# Proving Correctness of a greedy algorithm

- Greedy algorithm stays ahead.
  - Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.
- Exchange argument
  - Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.
- Structural
  - Discover a simple *structural* bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.



# Two vital properties of a Greedy Algorithm

- Greedy-choice property
  - a globally optimal solution can be arrived from a locally optimal choice.
  - algorithm proceeds in a top down fashion – reducing the given problem instance into smaller ones
- Optimal Sub-structure property
  - an optimal solution to a problem contains within it other optimal solutions to smaller subproblems

# Blank

# Blank