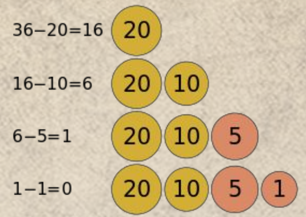
Greedy Algorithm



CSE-237 : ALGORITHM DESIGN AND ANALYSIS

EXAMPLE#1 – FINDING THE CHAMPIONS

- **Problem**: Pick ***k*** scores out of ***n*** scores such that the sum of these k scores is the largest.

- **Algorithm:**

```
FOR i = 1 to k
    pick out the largest number and
    delete this number from the input.
ENDFOR
```

EXAMPLE#2 – CONFERENCE SCHEDULING

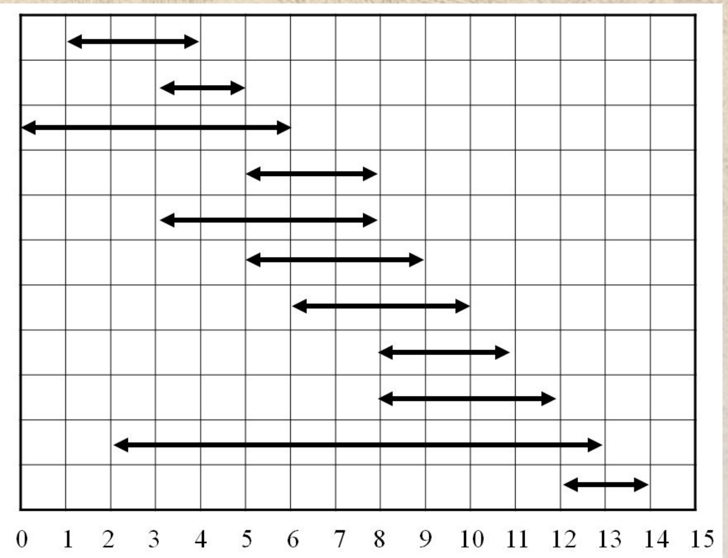
- A set of activities (conferences) – each conference has a start time and a finish time:

Conf ID	1	2	3	4	5	6	7	8	9	10	11
Start Time	1	3	0	5	3	5	6	8	8	2	12
Finish Time	4	5	6	7	8	9	10	11	12	13	14

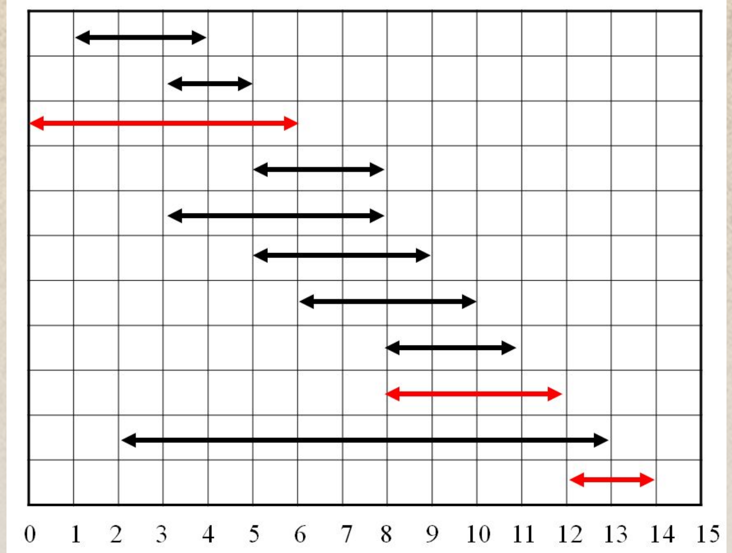
- What is the maximum number of activities that can be completed?

EXAMPLE#2 – CONFERENCE SCHEDULING …

INTERVAL REPRESENTATION

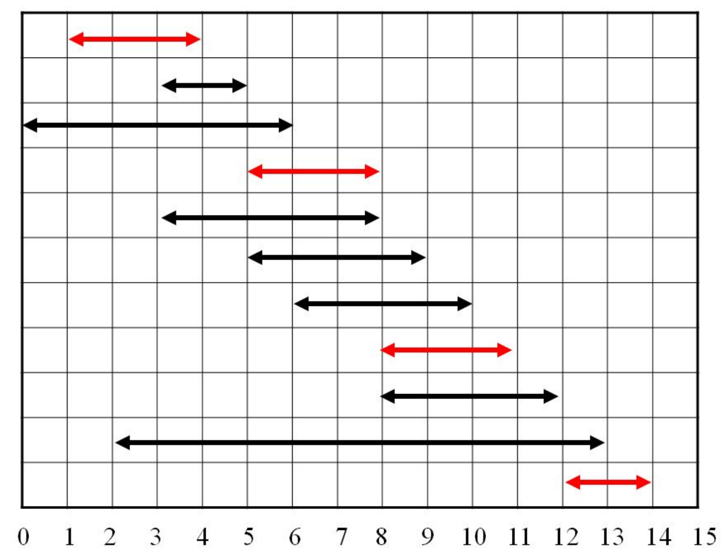


EARLY START STRATEGY

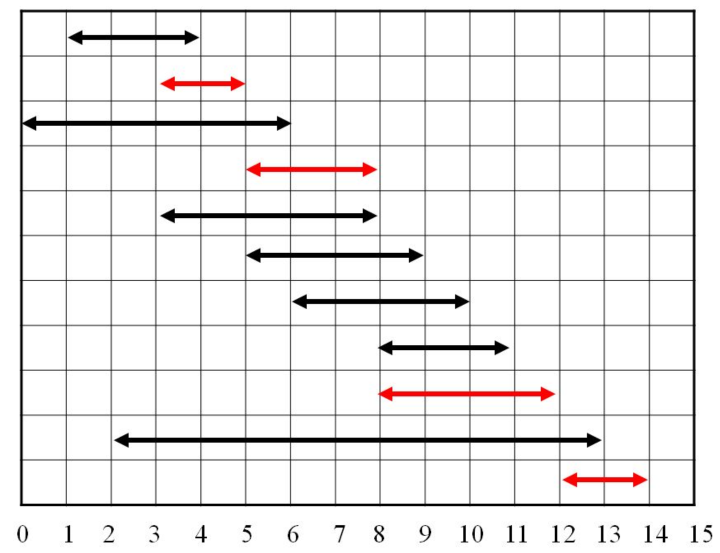


EXAMPLE#2 – CONFERENCE SCHEDULING …

EARLY FINISH STRATEGY



LATELY FINISH STRATEGY



WHY IT IS GREEDY?

- Greedy in the sense that
  - it leaves as much opportunity as possible for the remaining activities to be scheduled
- The greedy choice is the one that
  - maximizes the amount of unscheduled time remaining

OPTIMIZATION PROBLEMS

- A problem that may have many **feasible solutions** and **each solution** has a value.

- In the **maximization** problem - we wish to find a solution to maximize the value
- In the **minimization** problem - we wish to find a solution to minimize the value

- A **greedy algorithm** works in phases – taking the best solution right now, without regard for future consequences.

GREEDY METHOD

- Greedy strategy usually progresses in a top-down fashion, making one greedy choice after another, reducing each problem to a smaller one.
- Two ingredients that are exhibited by most problems that lend themselves to a greedy strategy:
  - **Greedy-Choice** property - when we have a choice to make, make the one that looks best *right now.*
  - **Optimal Substructure -** an optimal solution to the problem contains within it optimal solutions to sub-problems

GREEDY METHOD …

- Characteristics of greedy algorithm:
  - make a sequence of choices
  - each choice is the one that seems best so far, only depends on what's been done so far
  - choice produces a smaller problem to be solved

Optimal Substructure

Greedy Choice

FEATURES OF GREEDY SOLUTION

- To construct the solution in an optimal way, algorithm maintains two sets:
  - One contains chosen items **(solution/candidate set)** and the other contains rejected

- The greedy algorithm consists of four (**4**) function:
  - **Selection Function -** used to chose the best candidate to be added to the solution.
  - **Feasibility Function -** checks the feasibility of a set
  - **Objective Function -** used to assign value to a solution or partial solution
  - **Solution Function -** used to indicate whether a complete solution has been reached

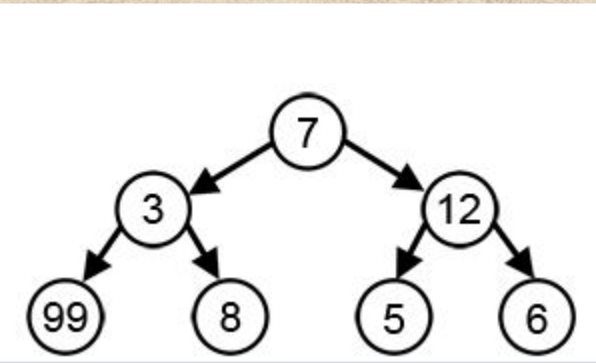
STRUCTURE OF GREEDY ALGORITHM

- Initially the set of chosen items (**solution/candidate set**) is empty.
- At each step
  - item will be added in a **solution set** by using **selection function**.
  - IF the set would no longer be feasible
    - reject items under consideration (and is never consider again).
  - ELSE IF set is still feasible THEN
    - **ADD** the current item.

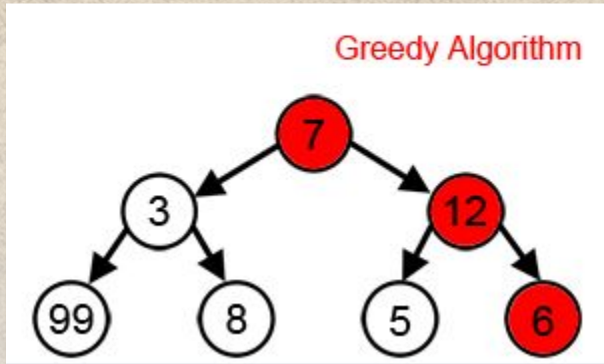
A **feasible set** (of candidates) is **promising** if it can be **extended to produce** not merely a solution, but an **optimal solution** to the problem. [An empty set is always promising why?]

FAILURE OF GREEDY ALGORITHM

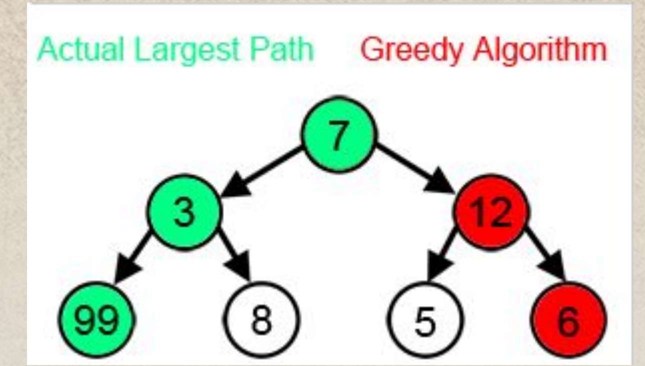
- **Problem #1**: With a goal of reaching the largest sum



Possible Solutions



Greedy Solution



Optimal Solution

FAILURE OF GREEDY ALGORITHM

- **Problem #2**: Find the minimum # of 4, 3, and 1 cent coins to make up 6 cents.

- Available Cents: 4, 3, 1

- Greedy Solution: 4+1+1

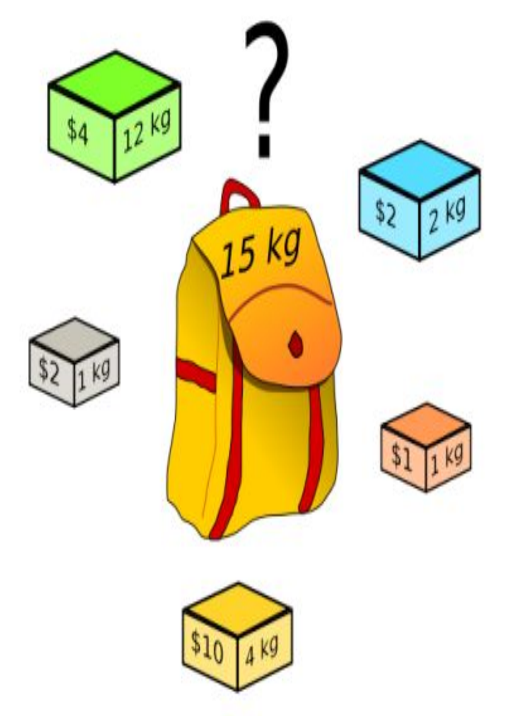
- Better Solution: 3+3

APPLICATION OF GREEDY STRATEGY

- Optimal solutions:
  - change making for “normal” coin denominations, (minimum/maximum) spanning tree, single-source shortest paths, **scheduling problems**, Huffman codes

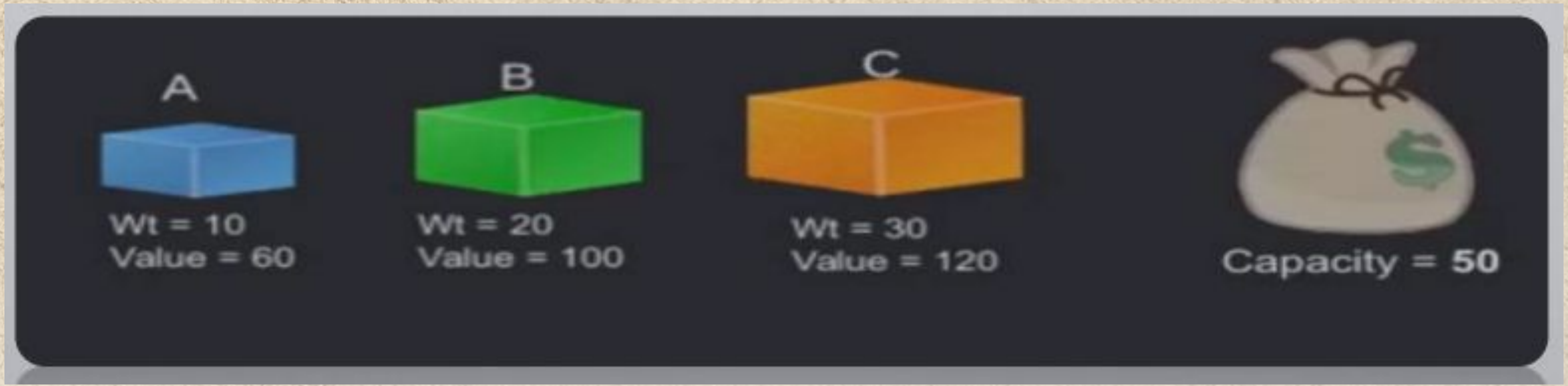
- Approximations
  - Traveling salesman problem (TSP), **knapsack problem**, other combinatorial optimization problems (CSP, WTAP, VRP)

KNAPSACK PROBLEM



- **Statement :** A thief robbing a store and can carry a maximal weight of w into their knapsack. There are n items and i^{th} item weight is w_i and is worth v_i dollars. What items should thief take?
- **Constraint :** The knapsack weight capacity is not exceeded and the total benefit is maximal.
- **Variants** of Kanpsack Problem:
  - 0-1 knapsack **:** items are indivisible. (either take an item or not)
  - Fractional knapsack : items are divisible. (can take any fraction of an item)

KNAPSACK PROBLEM - EXAMPLE



0-1 KNAPSACK

- Take B and C
- Total weight = 50
- Total value = 220

FRACTIONAL KNAPSACK

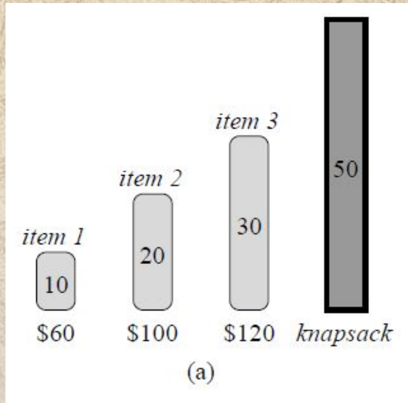
- Take A,B and 2/3rd of C
- Total weight = 50
- Total value = 240

KNAPSACK PROBLEM - SOLUTION

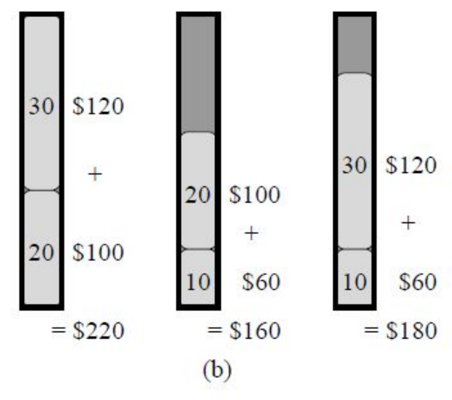
- Greedy Approach
  - Calculate the ratio value/weight for each item.
  - Sort the item on basis of this ratio.
  - Take the item with the highest ratio and add them until we can't add the next item as a whole.
  - At the end add the next item as much (fraction) as we can.

- Greedy Algorithm
  - INPUT: AN INTEGER N
    - Positive values w_i and v_i such that $1 <= i <= n$
    - Positive value W.
  - OUTPUT:
    - N values of xi such that 0 <= x+i <=1
    - Total profit P

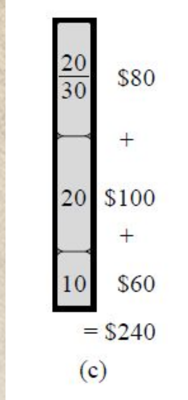
KNAPSACK PROBLEM – SOLUTION …



Knapsack Problem



0-1 Solution



Fractional Solution

FRACTIONAL KNAPSACK - ALGORITHM

p and w are arrays contain the profit and weight n objects ordered such that p[i]/w[i] >= p [i+1]/w[i+1] that is in decreasing order ,m is the knapsack size and x is the solution vector

```
GreedyKnapsack(m,n)
    for i ← 1 to n do
        x[i] ← 0
    End for
total ← m
    for i ← 1 to n do
        if (w[i]<= total)
            x[i] ← 1
            total ← total - w[i]
        else break // to exit the for-loop
        End if
    End for
    if(i<=n) x[i] ← total/w[i]
End GreedyKnapsack
```

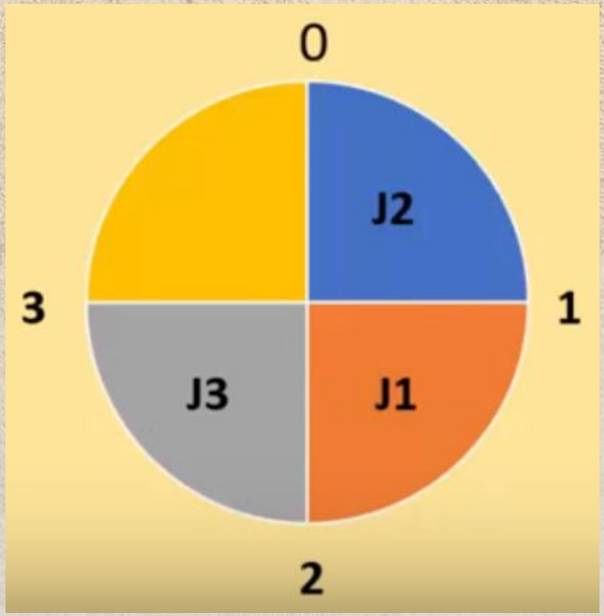
FRACTIONAL KNAPSACK - EXAMPLE

- 3 items:
  - item 1 weighs 10 lbs, worth $60 ($6/lb)
  - item 2 weighs 20 lbs, worth $100 ($5/lb)
  - item 3 weighs 30 lbs, worth $120 ($4/lb)
- knapsack can hold 50 lbs
- greedy strategy:
  - take item 1 [1] (10 lbs)
  - take item 2 [1] (20 lbs)
  - take item 3 [⅔] (20 lbs)

FRACTIONAL KNAPSACK - COMPLEXITY

- If the items are already sorted into decreasing order of *vi* / *wi,* then the while-loop takes a time in ***O(n)***;
- As main time taking step is sorting, the whole problem can be solved in
  - O(n log n) using merge/quick sort => sort: O(n log n), loop: O(n)
  - O(n^2) using selection/bubble sort => sort: O(n^2), loop: O(n)
  - O(n) using max-heap sort => heap: O(n) loop: O(log n)

JOB SCHEDULING WITH DEADLINE



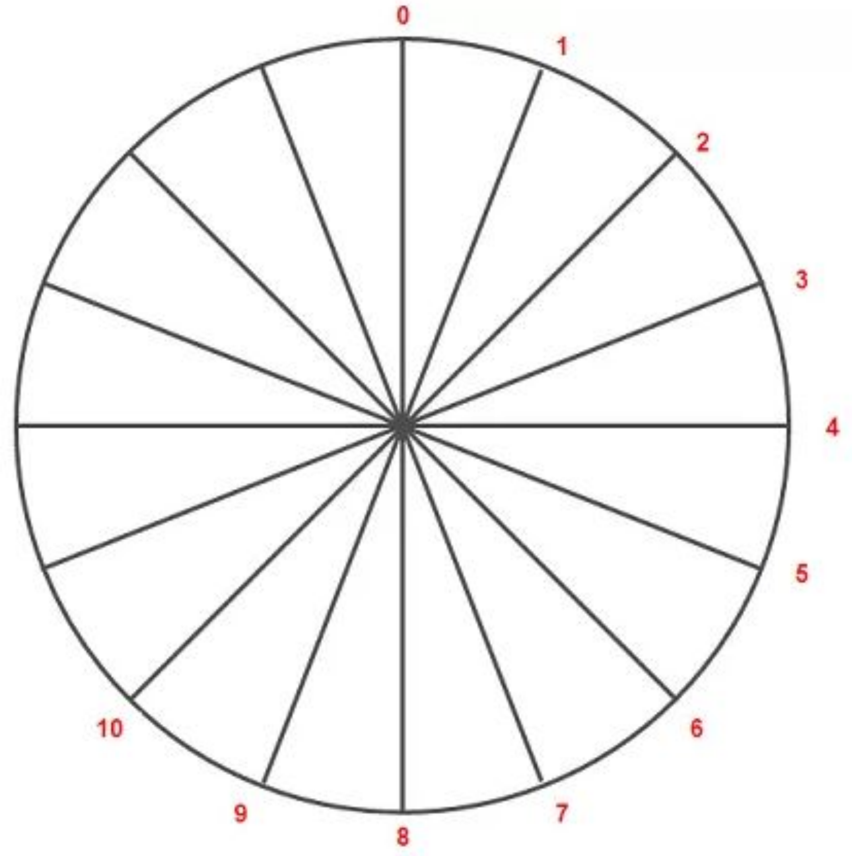
- **Statement :** If there are a set of jobs which are associated with deadline $d_i >= 0$ and profit $p_i > 0$. For any job i the profit is earned if and only if the job is completed by its deadline.
- **Objective :** Find a sequence of jobs, which is completed within their deadlines and gives maximum profit.
- **Constraint :** Any job takes single unit of time to execute, any job can't execute beyond its deadline, and only one job can be executed at a time.

JOB SCHEDULING WITH DEADLINE - SOLUTION

- **Standard** Greedy Approach
  - SORT all the jobs based on the profit in an increasing order.
  - Let **d** be the maximum deadline that will define the size of **solution** array.
  - Create a solution array **S** with **d** slots.
  - Initialize the content of array **S** with zero.
- Check for all jobs.
  - /* Nothing is gained by scheduling it earlier, and scheduling it earlier could prevent another more profitable job from being done */
  - If scheduling is possible using **r**th slot of array **S** to job **i** having a deadline **r**.
  - Otherwise look for location (**r-1**), (**-2**)…**1**.
  - Schedule the job if possible else reject.
- Return array S as the answer.

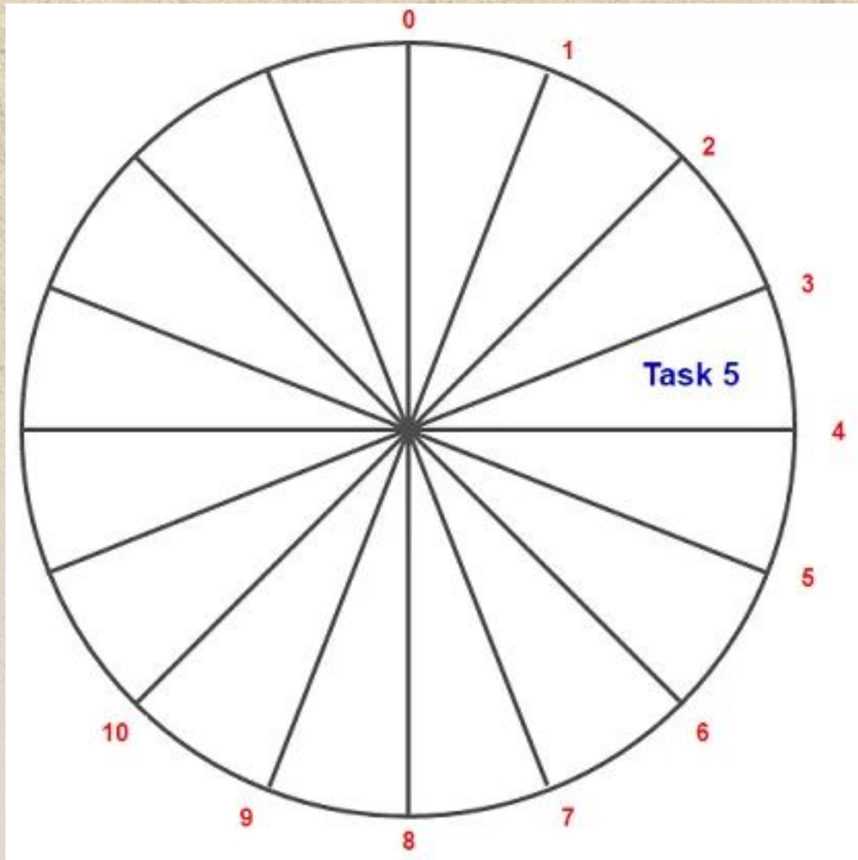
JOB SCHEDULING WITH DEADLINE - EXAMPLE

Task	Deadline	Profit
1	9	15
2	2	2
3	5	18
4	7	1
5	4	25
6	2	20
7	5	8
8	7	10
9	4	12
10	3	5



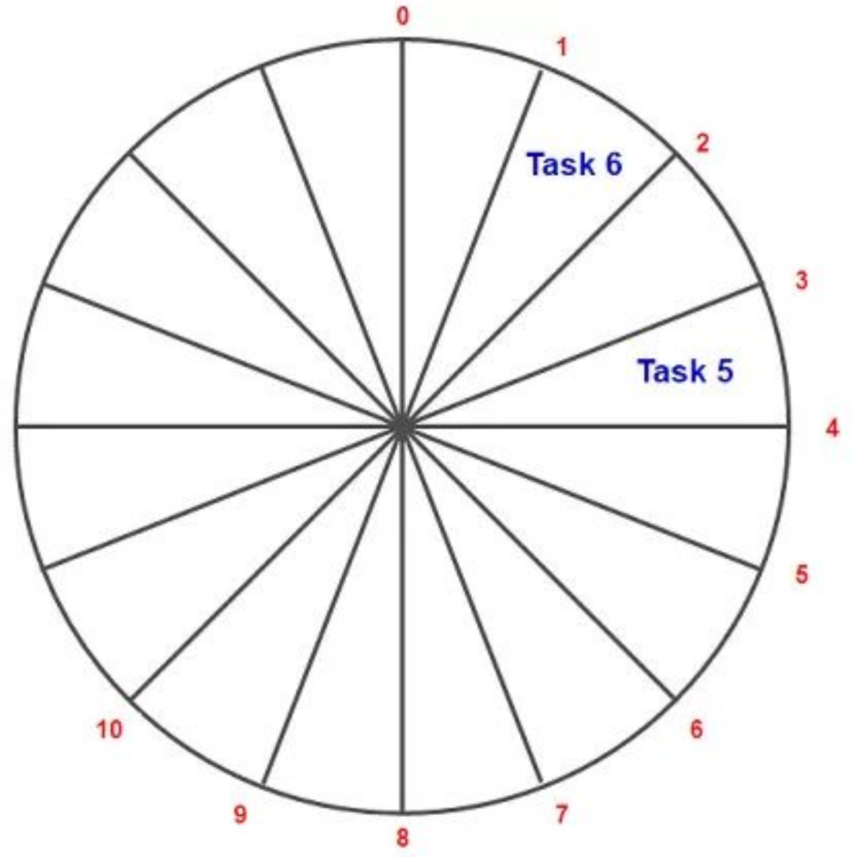
JOB SCHEDULING WITH DEADLINE - EXAMPLE

Task	Deadline	Profit
5	4	25
6	2	20
3	5	18
1	9	15
9	4	12
8	7	10
7	5	8
10	3	5
2	2	2
4	7	1



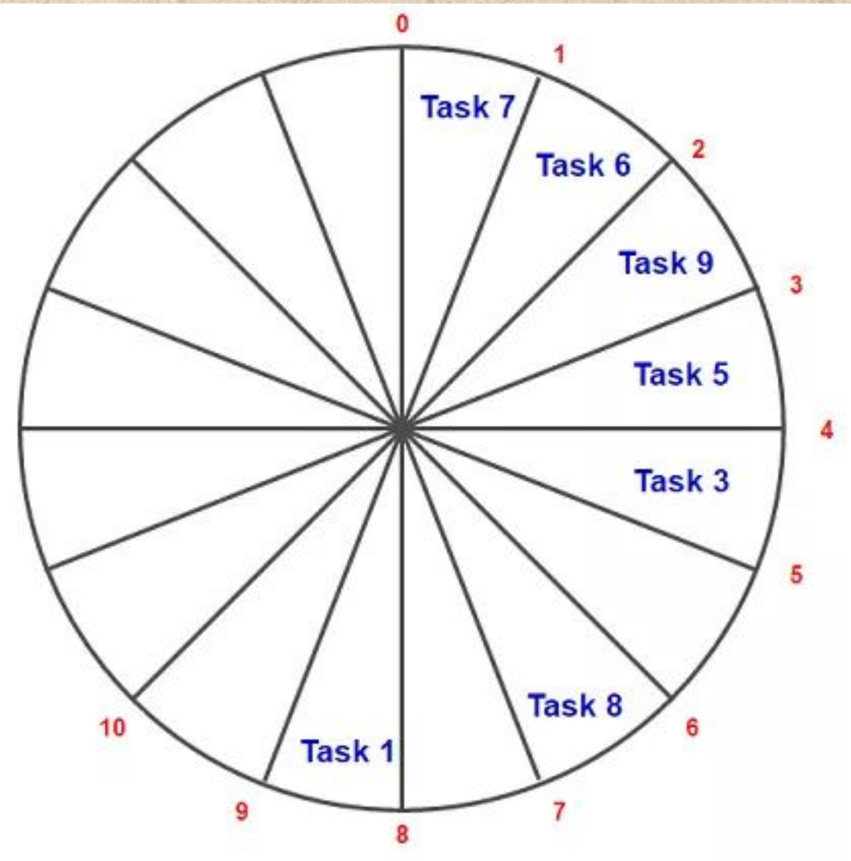
JOB SCHEDULING WITH DEADLINE - EXAMPLE

Task	Deadline	Profit
5	4	25
6	2	20
3	5	18
1	9	15
9	4	12
8	7	10
7	5	8
10	3	5
2	2	2
4	7	1



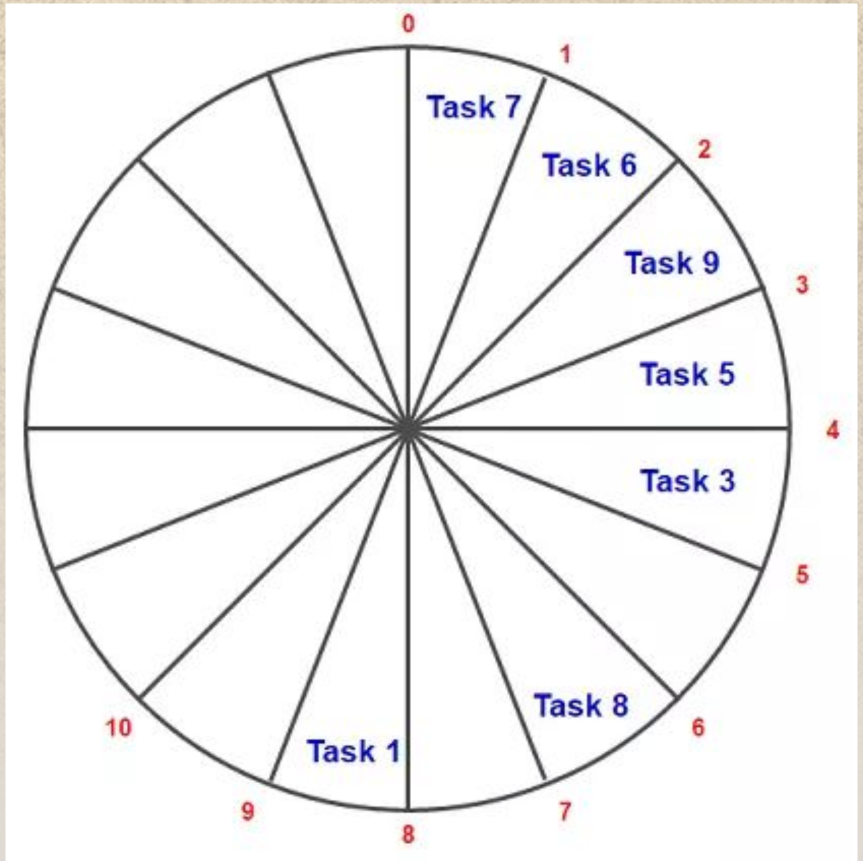
JOB SCHEDULING WITH DEADLINE - EXAMPLE

Task	Deadline	Profit
5	4	25
6	2	20
3	5	18
1	9	15
9	4	12
8	7	10
7	5	8
10	3	5
2	2	2
4	7	1



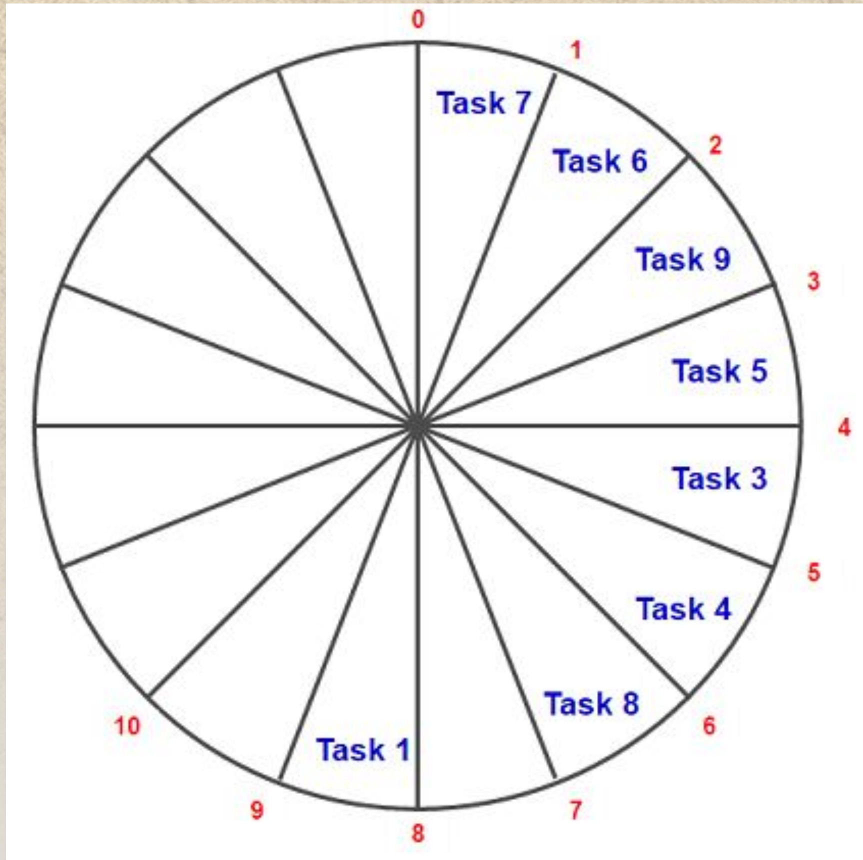
JOB SCHEDULING WITH DEADLINE - EXAMPLE

Task	Deadline	Profit
5	4	25
6	2	20
3	5	18
1	9	15
9	4	12
8	7	10
7	5	8
10	3	5
2	2	2
4	7	1



JOB SCHEDULING WITH DEADLINE - EXAMPLE

Task	Deadline	Profit
5	4	25
6	2	20
3	5	18
1	9	15
9	4	12
8	7	10
7	5	8
10	3	5
2	2	2
4	7	1



JOB SCHEDULING WITH DEADLINE - EXAMPLE

Task	Deadline	Profit
5	4	25
6	2	20
3	5	18
1	9	15
9	4	12
8	7	10
7	5	8
10	3	5
2	2	2
4	7	1

- The scheduled jobs are
  - 7, 6, 9, 5, 3, 4, 8, 1
- Total profit is 109
- Time complexity: O(n2)

DYNAMIC PROGRAMMING

Explore it on NEXT DAY