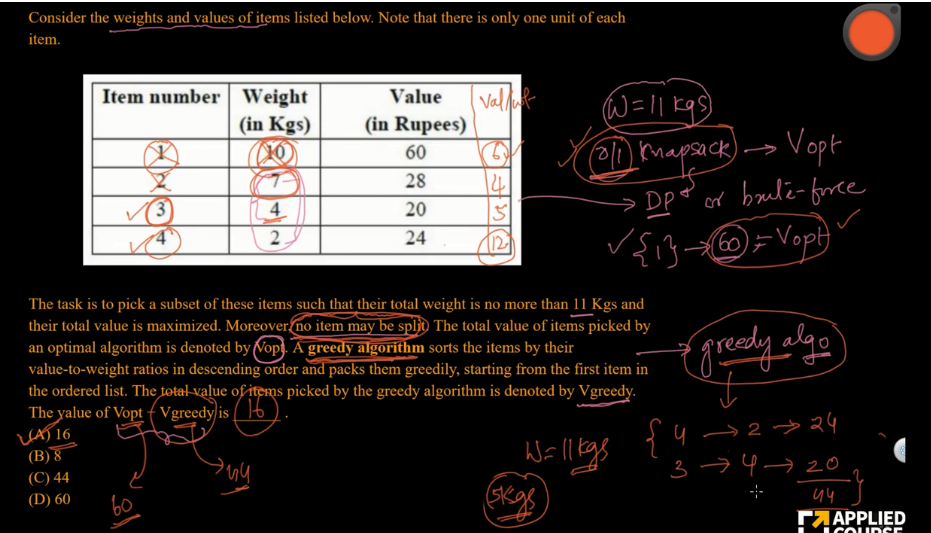


Here it is saying that no items can be split. So in the greedy algorithm, we cannot split the items.



**def** fractional\_knapsack(value, weight, capacity):

"""Return maximum value of items and their fractional amounts.

(max\_value, fractions) is returned where max\_value is the maximum value of

items with total weight not more than capacity.

fractions is a list where fractions[i] is the fraction that should be taken

of item i, where 0 <= i < total number of items.

value[i] is the value of item i and weight[i] is the weight of item i

for 0 <= i < n where n is the number of items.

capacity is the maximum weight.

"""

*# index = [0, 1, 2, ..., n - 1] for n items*

index = list(range(len(value)))

*# contains ratios of values to weight*

ratio = [v/w **for** v, w **in** zip(value, weight)]

*# index is sorted according to value-to-weight ratio in decreasing order*

index.sort(key=**lambda** i: ratio[i], reverse=True)

max\_value = 0

fractions = [0]\*len(value)

**for** i **in** index:

**if** weight[i] <= capacity:

fractions[i] = 1

max\_value += value[i]

capacity -= weight[i]

**else**:

fractions[i] = capacity/weight[i]

max\_value += value[i]\*capacity/weight[i]

**break**

**return** max\_value, fractions

n = int(input('Enter number of items: '))

value = input('Enter the values of the {} item(s) in order: '

.format(n)).split()

value = [int(v) **for** v **in** value]

weight = input('Enter the positive weights of the {} item(s) in order: '

.format(n)).split()

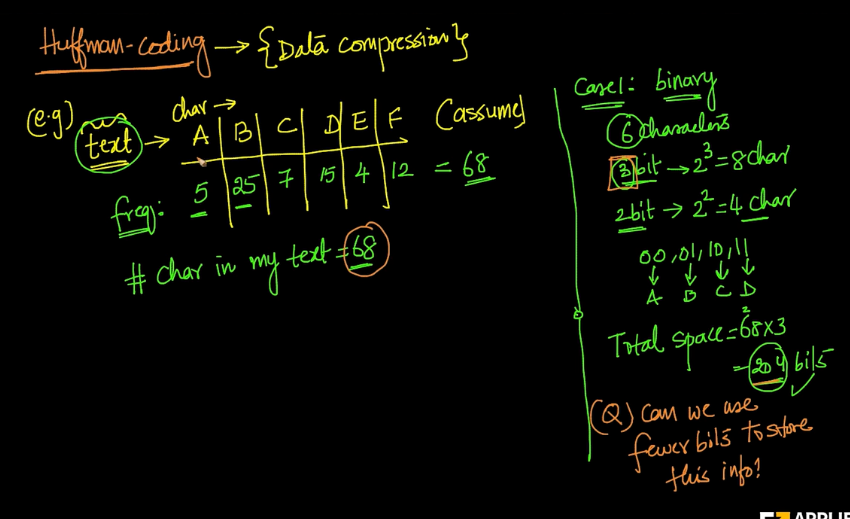
weight = [int(w) **for** w **in** weight]

capacity = int(input('Enter maximum weight: '))

max\_value, fractions = fractional\_knapsack(value, weight, capacity)

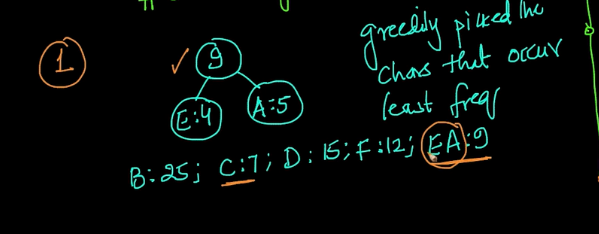
**print**('The maximum value of items that can be carried:', max\_value)

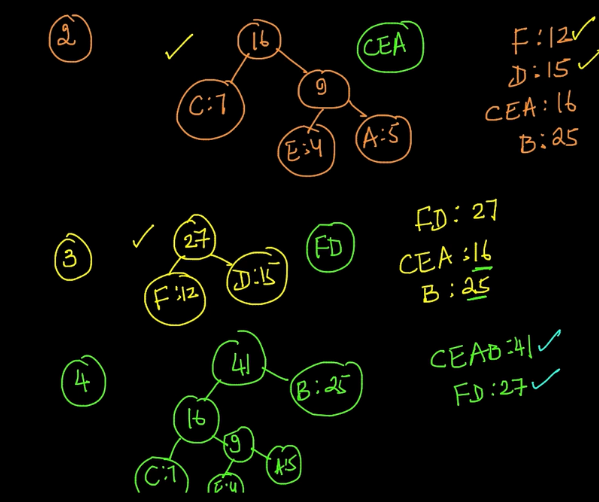
**print**('The fractions in which the items should be taken:', fractions)

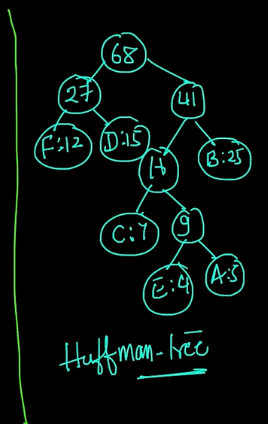


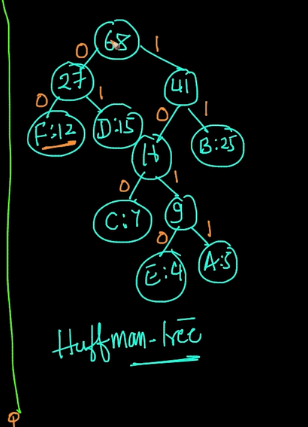
I will start greedyly and find out the characters which are least freqeunt.

Here in the above data, we have E and A.



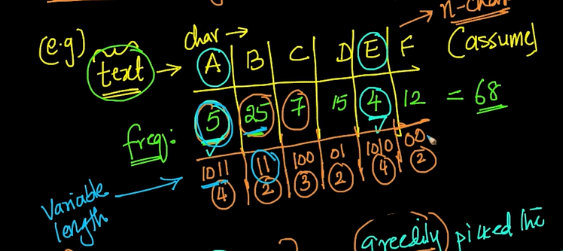




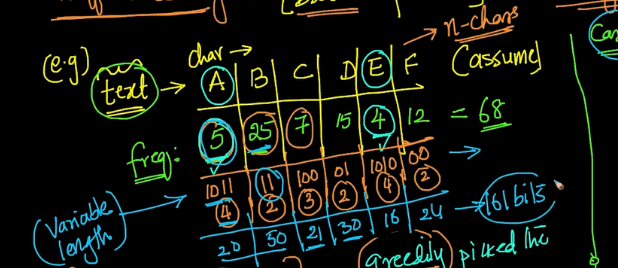


Item on the left, we will give a value of 0,for every value in the right, we will give a value of 1.

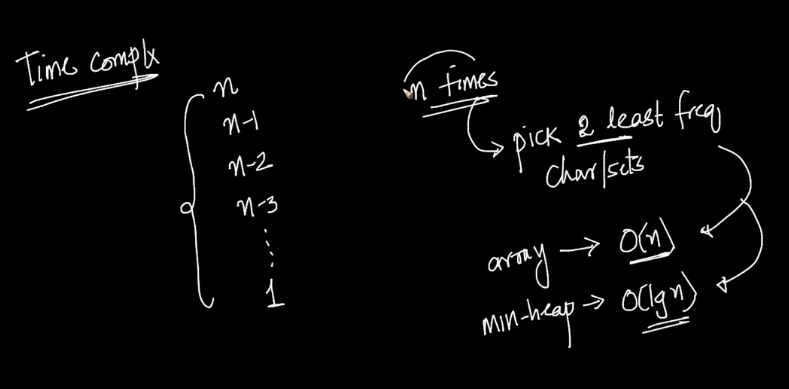
So Here F will be represented as 00. D is represented as 01. B is represented as 11. C is represented as 100.



Every character is represnted as variable length.

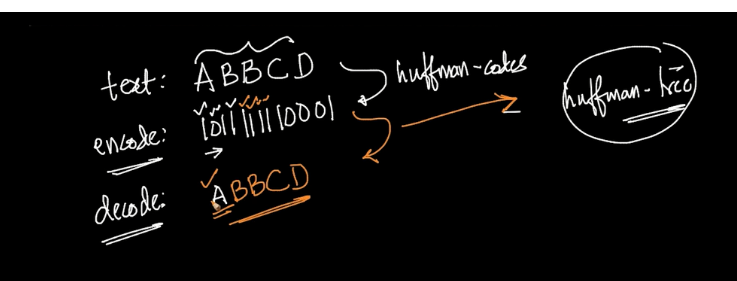


It is having becuase those character which are occuring more frequently is given less bits and the chracters occuring less frequently is given more bits.

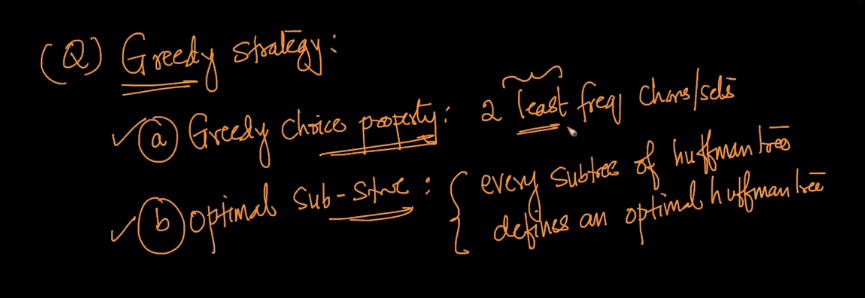


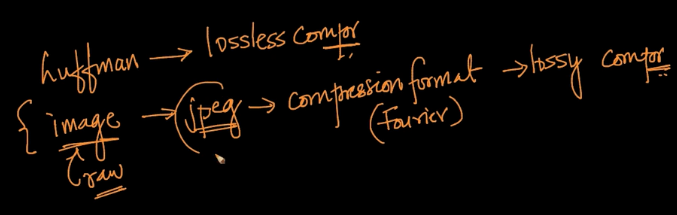
We are running the algorithm n times and each time we need to take the two least frequent words.

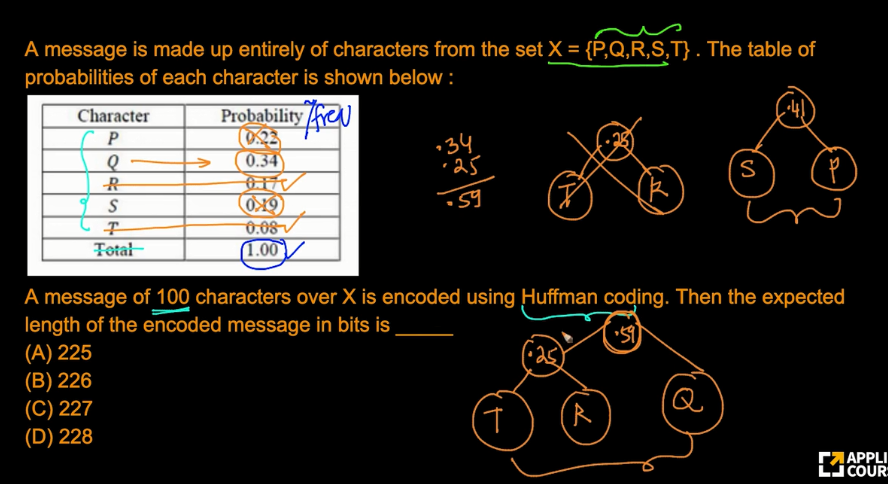
time complexity will be O(nlogn).

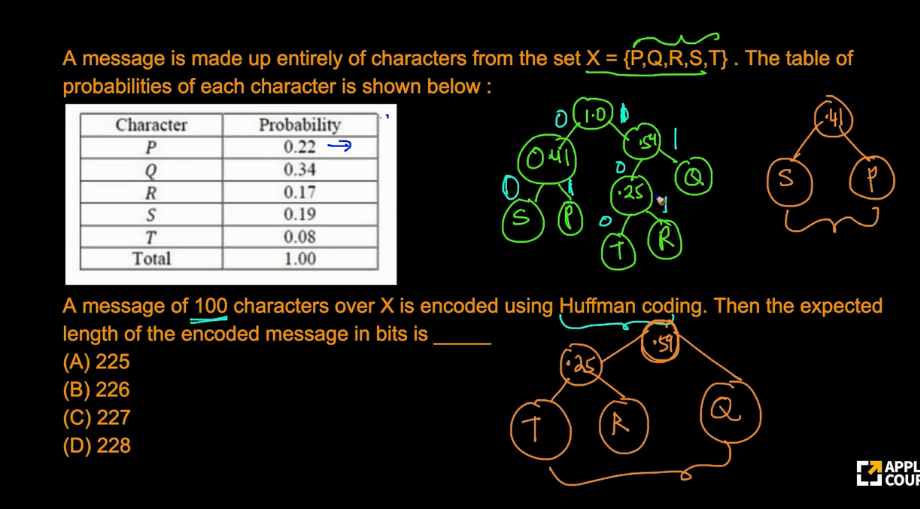


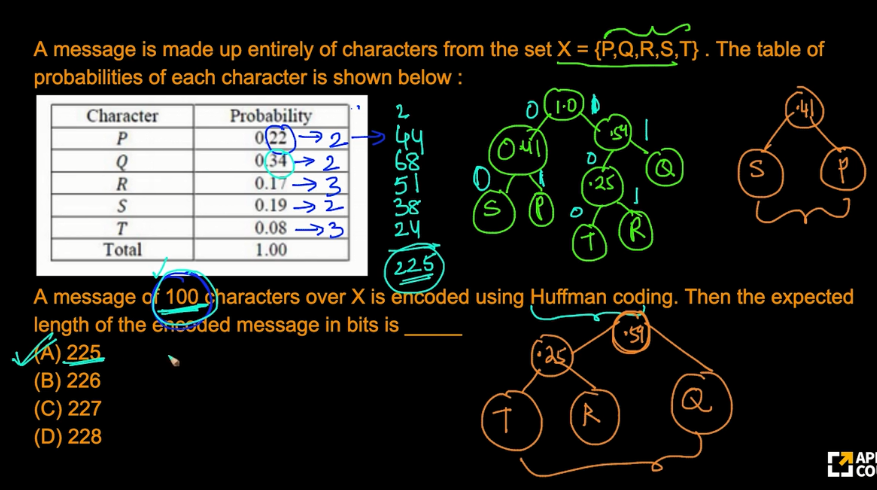
Here we are going from root node everytime and going to the leaf node.



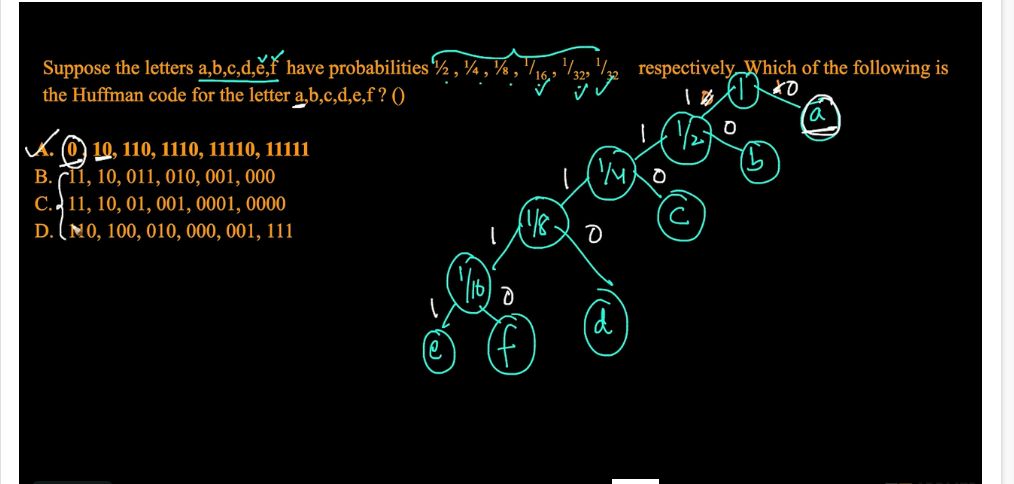




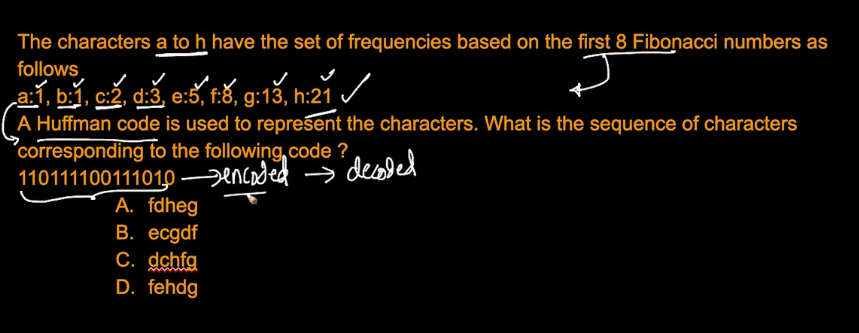


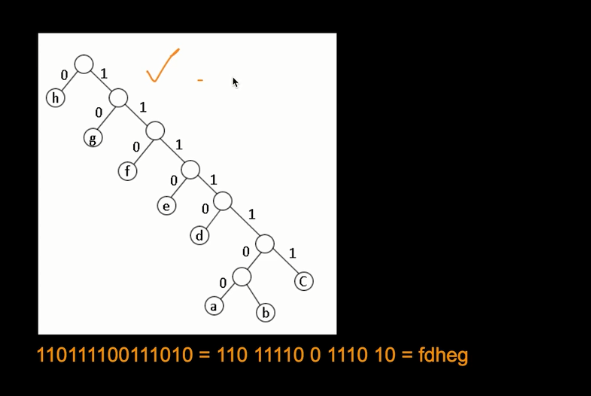


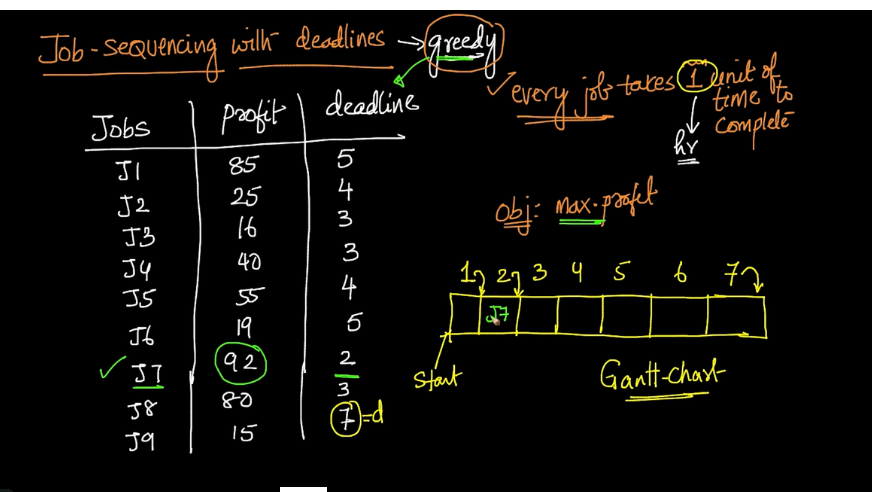
here on the top, 2 ,2,3 are the bits. probabilty means number of character present out of 100 characters. So here, 0.22 means 22\*2=44 like this we have calculated.



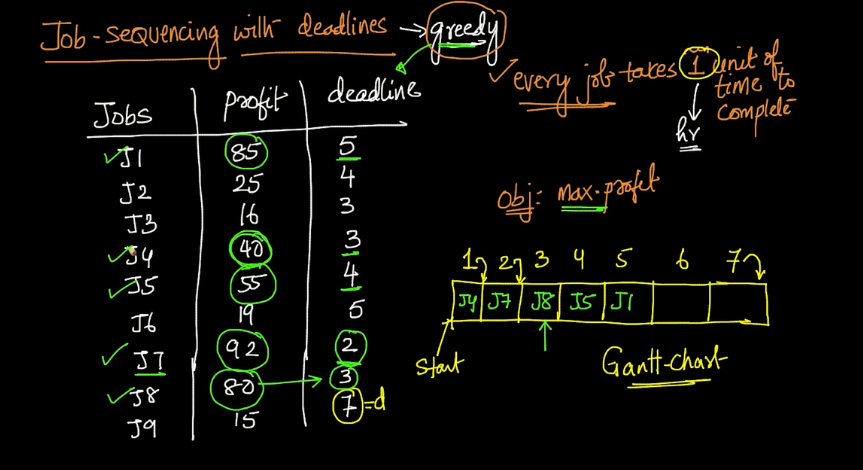
Here "a" should be represented by 0 or 1. It should be represeted as 1 character.



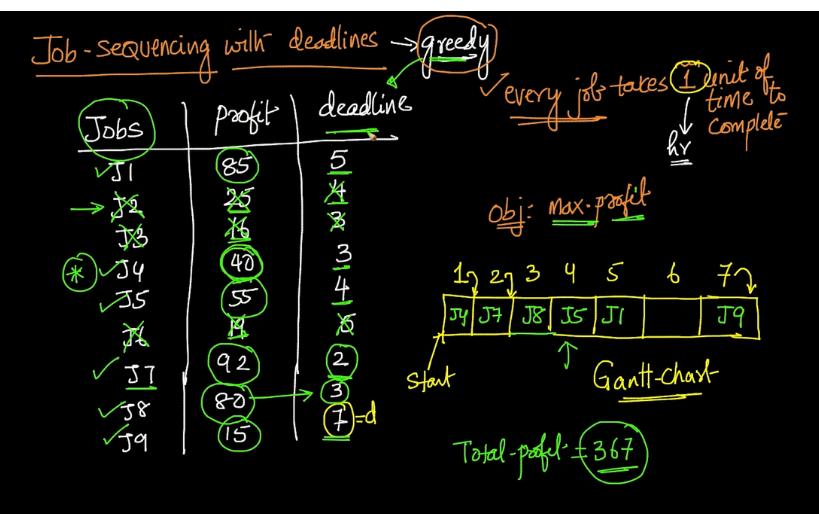


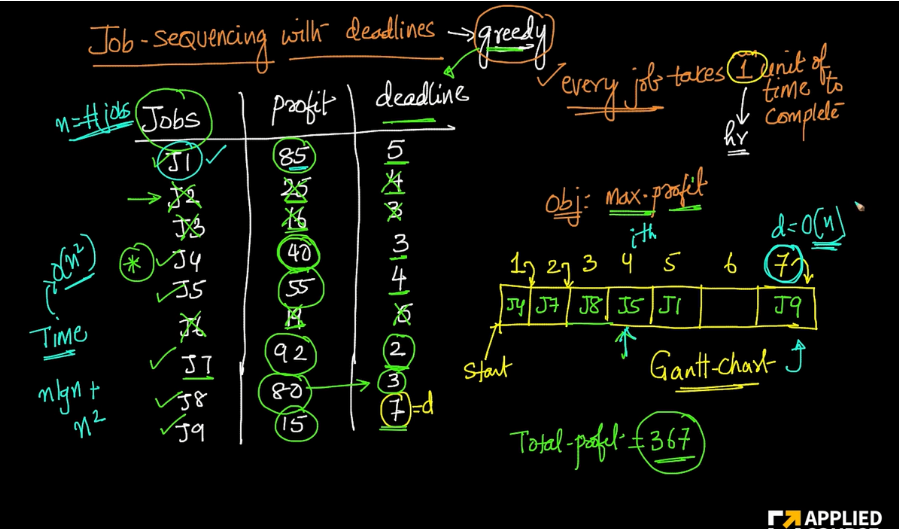


Here J7 is 2 hours means we need to finish the job before 2 hours. Here we are picking the jobs that has maximum profit.



for J4, we see that the cell J8 and J7 is already filled. So we will put it in 1st slot.





There is optimal substructure, becuase we are taking the jobs,putting it into the chart and removing the job after that. After removing i need to find out the optimal job from the rest of the sequence.

# A class to store job details. Each job has an identifier,

# a deadline, and profit associated with it.

class Job:

    def \_\_init\_\_(self, taskID, deadline, profit):

        self.taskID = taskID

        self.deadline = deadline

        self.profit = profit

# Function to schedule jobs to maximize profit

def scheduleJobs(jobs, T):

    # stores the maximum profit that can be earned by scheduling jobs

    profit = 0

    # list to store used and unused slots info

    slot = [-1] \* T

    # arrange the jobs in decreasing order of their profits

    jobs.sort(key=lambda x: x.profit, reverse=True)

    # consider each job in decreasing order of their profits

    for job in jobs:

        # search for the next free slot and map the task to that slot

        for j in reversed(range(job.deadline)):

            if j < T and slot[j] == -1:

                slot[j] = job.taskID

                profit += job.profit

                break

    # print the scheduled jobs

    print("The scheduled jobs are", list(filter(lambda x: x != -1, slot)))

    # print total profit that can be earned

    print("The total profit earned is", profit)

if \_\_name\_\_ == '\_\_main\_\_':

    # List of given jobs. Each job has an identifier, a deadline, and

    # profit associated with it

    jobs = [

        Job(1, 9, 15), Job(2, 2, 2),

        Job(3, 5, 18), Job(4, 7, 1),

        Job(5, 4, 25), Job(6, 2, 20),

        Job(7, 5, 8), Job(8, 7, 10),

        Job(9, 4, 12), Job(10, 3, 5)

    ]

    # stores the maximum deadline that can be associated with a job

    T = 15

    # schedule jobs and calculate the maximum profit

    scheduleJobs(jobs, T)

