Unit - III

1. Solve Job sequencing with deadlines using Greedy method with the following instances n=5, deadline (2,1,2,1,3) and profits (100,19,27,15,13)

Job sequencing with deadlines:

Job sequencing with deadlines is a Greedy algorithm problem used in scheduling tasks to maximize total profit when each job has a deadline and profit associated with it

Given that

n=5

deadlines = (2,1,2,1,3)

profits = (100,19,27,15,13)

step1: List the jobs with their deadlines and profits

Jobs	J_1	J_2	J_3	J_4	J_5
profits	100	19	27	15	13
deadlines	2	1	2	1	3

step2: Sort the jobs in descending order of profit

Jobs	J_1	J_3	J_2	J_4	J_5
profits	100	27	19	15	13
deadlines	2	2	1	1	3

Step3: find the maximum deadlines

Here maximum deadline = 3

0 — 1 — 2 — 3

step4: Schedule the jobs greedily.

Here we assign each job to the latest available slot befor or equal to its deadline.

	Jobs considering	slot assignment	solution	profit
	J_1	[1,2]	J_1	100
	J_3	[0,1] [1,2]	J_3, J_1	100+27
X	J_2	[0 1] [1 2]	J_3, J_1	127
X	J_4	[0 1] [1 2]	J_3, J_1	127
	J_5	[0 1] [1 2] [2 3]	J_3, J_1, J_5	127+13 = 140

Here J_4 profit is 15 & deadline = 1.

slot 1 is full so cannot schedule J_4.

J_2 profit is 19 & deadline is 2.

slot 1 is also occupied so we can easily ignore it

step5: final schedule

0 $\underline{J_3}$ 1 $\underline{J_1}$ 2 $\underline{J_5}$ 3

Total profit = 100+27+13 = 140

4 Describe Strassen's matrix multiplication algorithm with example.

Strassen's matrix multiplication:

It is a divide and conquer algorithm that multiplies two matrices more efficiently than the standard method, reducing the time complexity from $O(n^3)$ to approximately $O(n^{2.81})$

By using 7 multiplications and some addition and substractions to calculate sub matrices instead of using standard 8 multiplications.

It is particularly advantageous for large matrices, though the standard method often better for smaller ones

Strassen's matrix multiplication follows 3 steps.

1. Divide:

The input matrices A and B are split into 4 equal sized sub matrices as below

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

2. Calculate 7 intermediate values:

Instead of 8 multiplications, strassen's algorithm calculates 7 specific intermediate values through additions and substractions of these submatrices

3. Combine:

These 7 intermediate values are combined in a specific combinations of additions and substractions to compute the four sub matrices of the result matrix C

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Algorithm:

Input : $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ $\quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$\in \mathbb{R}^{n \times n}$

if $n = 1$ then

$C = A \cdot B$

else

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q = B_{11}(A_{21} + A_{22})$

$R = A_{11}(B_{12} - B_{22})$

$S = A_{22}(B_{21} - B_{11})$

$T = (A_{11} + A_{12}) B_{22}$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$V = (A_{12} - A_{22})(B_{21} + B_{22})$

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

output :

$$A * B = C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$\in \mathbb{R}^{n \times n}$

Time Complexity:

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 7T(n/2)+n^2 & \text{if } n>1 \end{cases}$$

$a=2$

$b=7$

$\log_a b = \log_2 7 = 2.81$

$T.C = O(n^{2.81})$

Example:

$$A = \begin{bmatrix} 1 & 3 \\ 7 & 5 \end{bmatrix} \qquad B = \begin{bmatrix} 6 & 8 \\ 4 & 2 \end{bmatrix}$$

$$A*B = \begin{bmatrix} 1(6)+3(4) & 1(8)+3(2) \\ 7(6)+5(4) & 7(8)+5(2) \end{bmatrix}$$

$$= \begin{bmatrix} 6+12 & 8+6 \\ 42+20 & 56+10 \end{bmatrix} = \begin{bmatrix} 18 & 14 \\ 62 & 66 \end{bmatrix}$$

$P = (1+3)(6+2) = 6(8) = 48$

$Q = 6(7+5) = 6(12) = 72$

$R = 1(8-2) = 6$

$S = 5(4-6) = 5(-2) = -10$

$T = (1+3)2 = 4(2) = 8$

$U = (7-1)(6+8) = 6(14) = 84$

$V = (3-5)(4+2) = -2(6) = -12$

$C_{11} = 48-10-8-12 = 18$

$C_{12} = 6+8 = 14$

$C_{21} = 72-10 = 62$

$C_{22} = 48+6-72+84 = 66$

$$C = \begin{bmatrix} 18 & 14 \\ 62 & 66 \end{bmatrix}$$

3. Write merge sort algorithm and trace algorithm for the elements: 12, 9, 3, 0, 5, 10, 7, 2, 6 and device time complexity of merge sort

Merge sort:

It is a popular sorting algorithm known for its efficiency and stability. It follows the divide and conquer approach.

It works by recursively dividing the input array into two halves, recursively sorting the two halves and finally merge together to obtain the sorted array.

Merge sort consists of three steps:

1. Divide
2. Conquer
3. Merge

1. Divide:

Divide the array recursively into two halfs until it can be no more divided.

2. Conquer:

Each sub array is sorted individually using the merge sort algorithm

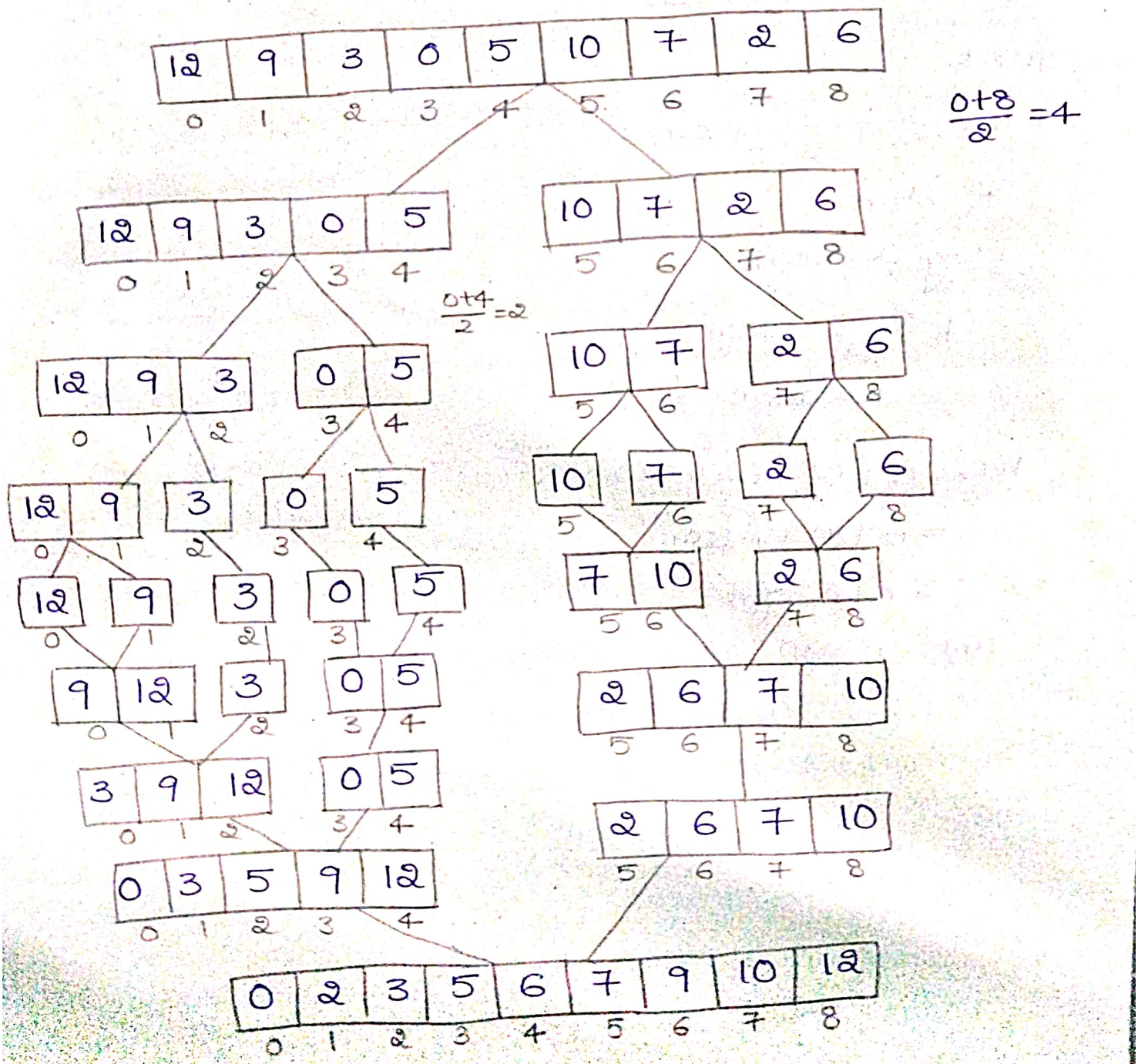
3. Merge:

The sorted subarrays are merge together in sorted order. This process continues until all elements from both subarrays have been merged

Algorithm:

```
mergesort(int arr, int l, int r
{
    if (l>r)
    {
        int m=(l+r)/2;
        mergesort(arr, l, m)
        mergesort(arr, m+1, r);
        mergesort(arr, l, m, r);
    }
}
```

elements : 12, 9, 3, 0, 5, 10, 7, 2, 6



Time complexity of merge sort:

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2T(\frac{n}{2})+n & \text{if } n>1 \end{cases}$$

By using substitution method

$$T(n) = 2T(\frac{n}{2})+n \qquad = 2^1 T(\frac{n}{2^1}) + 1 \cdot n$$

$$= 2\left[2T(\frac{n}{2}/2)+\frac{n}{2}\right]+n$$

$$= 4T(\frac{n}{4})+2n \qquad = 2^2 T(\frac{n}{2^2})+2n$$

$$= 4\left[2T(\frac{n}{4}/2)+\frac{n}{4}\right]+2n$$

$$= 8T(\frac{n}{8})+3n \qquad = 2^3 T(\frac{n}{2^3})+3 \cdot n$$

for k

$$T(n) = 2^k T(\frac{n}{2^k}) + k \cdot n$$

if $2^k = n$

$$\log 2^k = \log n$$

$$k \log_2 2 = \log n$$

$$k = \log n$$

$$T(n) = n \cdot T(\frac{n}{n}) + \log n \times n$$

$$= n\,T(1) + n\log n$$

$$= n + n\log n$$

$$T(n) = n\log n$$

2. Design an algorithm to sort the given list of elements using quick sort incorporating divide and conquer technique. Sort the following list using the same 14, 24, 10, 28, 74, 19, 36, 69

Quick sort:

It is a sorting algorithm based on the divide and conquer approach.

Here we chooses an element as a pivot and partition the given array around the selected pivot by placing the pivot in the correct position in the sorted array.

It works on the principle of divide and conquer, breaking down the problem into smaller sub problems.

There are mainly three steps in the algorithm.

1. choose a pivot
2. partition the array
3. Recursive call

1. choose a pivot:

select an element from the array as the pivot. The choice of pivot can vary (Eg: first, last, middle element)

2. partition the array:

Rearrange the array around the pivot. After partitioning, all elements smaller than the pivot will be on its left, and all elements greater than the pivot will be on its right. the pivot is then in its correct position, and we obtain the index of the pivot

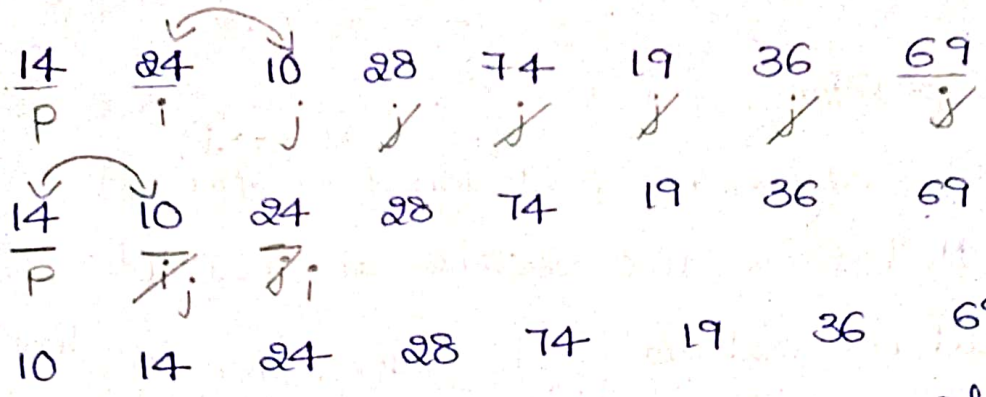
3. Recursive call:

Recursively apply the same process to the two partitioned sub-arrays

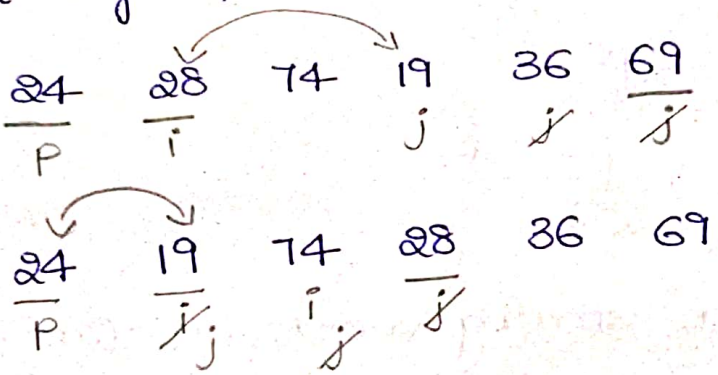
Algorithm:

```
int partition ( int arr[], int low, int high)
{
    int pivot = arr[low];
     int i = low+1
     int j = high
    while(i<j)
    {
      while( i<=high && arr[i]<=pivot)
           i++;
       while (j>=low && arr[j]>pivot)
           j--;
       if (i<j)
           swap (&arr[i], &arr[j])
    }
  swap(&arr[low],&arr[j]);
  return j;
}
void quicksort (int arr[], int low, int high)
{ if(low<high)
   {
     int pi = partition (arr, low, high);
      quicksort (arr, low, pi-1);
      quicksort (arr, pi+1, high);
   }
}
```

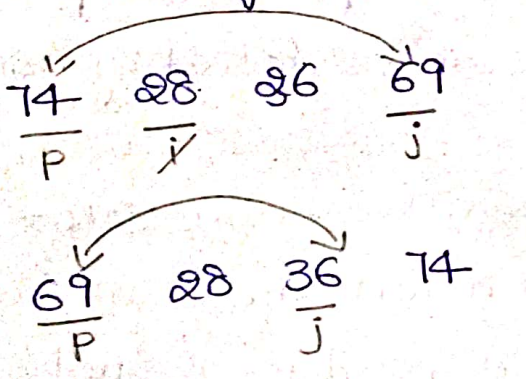
elements : 14, 24, 10, 28, 74, 19, 36, 69.



Here pivot is in the correct position. Then we take right right part.



Here pivot is at right position. Then we take rightpart



final sorted array is

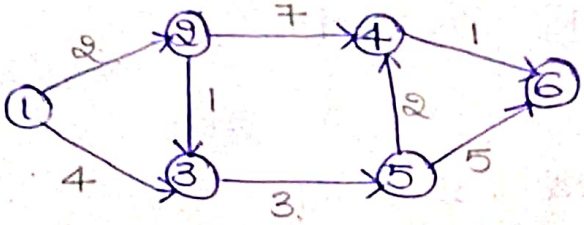
10 14 19 24 28 36 69 74

5. Describe Dijkstra's algorithm for shortest path with example.

Dijkstra's algorithm

→ The Dijkstra's algorithm is designed to find the shortest path between two vertices of a graph.

→ If a directed weighted graph is given then we have to find a shortest path from starting vertex to other vertices

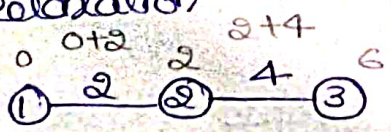


→ Here we taking 1 as starting vertex and find the shortest path to all other vertices

→ Here the problem is shortest path i.e minimization pro

Minimization problem is an optimization problem. Optimization problem can be solved by using Greedy approach

→ Here the Dijstra's algorithm always select a vertex of shortest path then it will update the shortest path to other vertices if possible. This updation is called as Relaxation



Relaxation formula:

If $d[u]+c[u,v] < d[v]$ then

$d[v] = d[u]+c[u,v]$

from the above example.

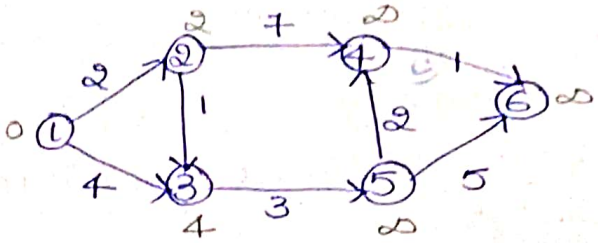
$2+4 \leq \infty$

$d[v] = 2+4 = 6$

step1: for any graph write the shortest path from the source vertex as '0' at 1

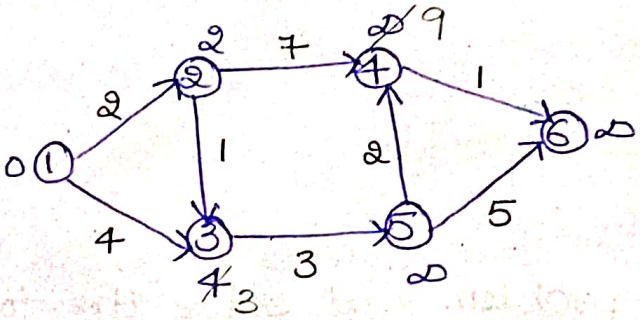
We write the shortest path for $1 \to 2 = 0+2 = 2$

$1 \to 3 = 0+4 = 4$, $1 \to 4 = \infty$, $1 \to 5 = \infty$, $1 \to 6 = \infty$



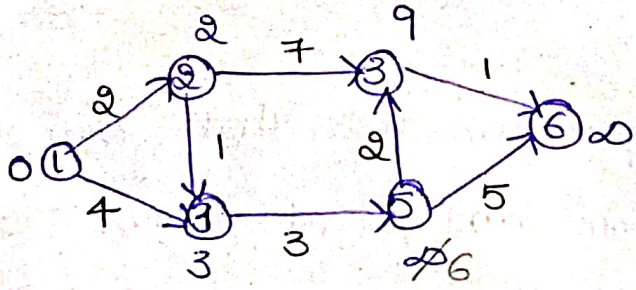
step2: select the shortest path out of 2, 4, ∞, ∞, ∞

Here 2 is the shortest path. so select vertex 2 and perform Relaxation operation.



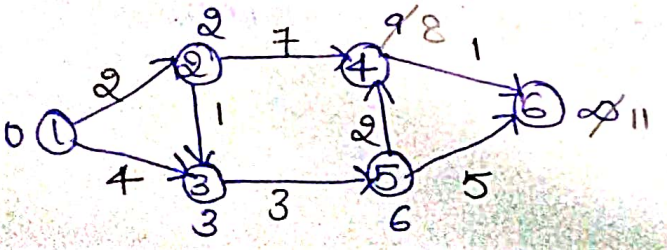
step 3: select the shortest path out of 3, 9, ∞, ∞

Here 3 is the shortest path. so select vertex 3 and perform Relaxation operation.



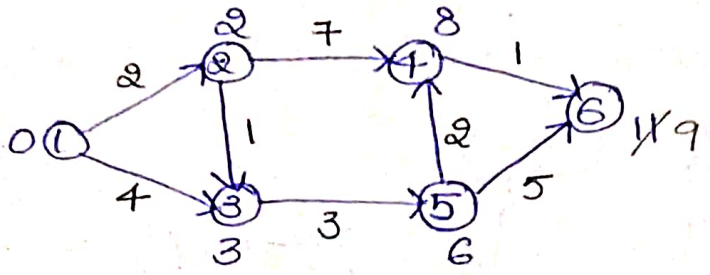
step4: select the shortest path out of 6, 9, ∞.

Here 6 is the shortest path. So select vertex 5 and perform Relaxation operation



Step 5: select the shortest path out of 8, 11.

Here 8 is the shortest path so select vertex 4 and perform Relaxation operation



The shortest path of the vertices are given below

V	d[V]
1	0
2	2
3	3
4	8
5	6
6	9

6. Define Knapsack problem and solve the following instances of knapsack problem using greedy method $n=3$, $(W_1, W_2, W_3) = (2, 3, 4)$, $(P_1, P_2, P_3) = (1, 2$ and $m=6$.

Knapsack problem:

The knapsack problem states that- given a set of items, holding weights and profit values, one must determine the subset of the items to be added in a knapsack such that, the total weight of the items must not exceed the limit of the knapsack and its total profit value is maximum

It is one of the most popular problems that take greedy approach to be solved. It is called as the fractional knapsack problem.

Algorithm:

1. Consider all the items with their weights and profits mentioned respectively.
2. Calculate P/W of all the items and sort the items in descending order based on their P/W values
3. Without exceeding the limit, add the items into the knapsack.
4. If the knapsack can still store some weight, but the weights of other items exceed the limit, the fractional part of the next time can be added
5. finally calculate profit and weight

Problem:

Given $n=3$
$m=6$

$(W_1, W_2, W_3) = (2,3,4)$

$(P_1, P_2, P_3) = (1,2,5)$

step1: consider all the items with their weights and profits mentioned respectively and calculate P/W values.

P	1	2	5
W	2	3	4
P/W	0.5	0.6	1.25

step2: sort the items in descending order based on their P/W values.

P	5	2	1
W	4	3	2
P/W	1.25	0.6	0.5

step3: Without exceeding the limit, add the items into the knapsack.

O_1	O_2	O_3
1	3/2	0

① consider heighest p/w value (1.25) the associa weight is 4. here our constraint is m=6.

$m = 6-4 = 2$

② consider second/next heighest p/w value (0.6) the associated weight is 3. here our constraint is m=2. But the associated weight is 3; it is exceeding the limit. Now we consider the fraction value 3/2 2/3

step4: calculate weight and profit

$$\sum x_i p_i = 1\times 5 + \frac{2}{3}\times 2 + 0\times 1$$

$$= 5 + \frac{4}{3}$$

$$= 5 + 1.3 = 6.3$$

$$\sum x_i w_i = 1\times 4 + \frac{2}{3}\times 3 + 0\times 2$$

$$= 4+2 = 6.$$

our constraint m=6 is reached