24/03/2

UNIT-2: DIVIDE AND CONQUER
GREEDY METHOD

Divide And Conquer

- In divide and Conquer divide the problem into subproblem.
- Each sub-problem must be solved and Combine all the sub-problems into a single solution.
- Divide and Conquer strategy consists of three steps:
  1. Divide → Problem is divided into sub-problems.
  2. Conquer → If subproblem is large again divide the sub-problems and each problem can be solved independently).
  3. Combine → All the solutions are Combine together to get a Single Solution.

Control Abstraction

- Control Abstraction gives the flow of control how the problem can be solved.

```
1. Algorithm DAndC(P)
2. {
3. if Small(P) then S(P)
4. else
5. {
6. Divide P into Smaller instance P1 P2 ... Pk k≥1
7. Apply DandC to each of there problems
8. return Combine (DAndC(P1),DAndC(P2)....DAndC(Pn))
9. }
10. }
```

→ The time complexity of Divide and Conquer strategy is

$$T(n) = \begin{cases} g(n) & \text{, if } n \text{ is small} \\ T(n_1)+T(n_2)+T(n_3)+\ldots+T(n), & \text{if } n \text{ is large} \end{cases}$$

→ The same is equation is written in the form of recurrence relation.

$$T(n) = \begin{cases} 1 & \rightarrow n=1 \\ aT(n/b)+f(n) & \rightarrow n>1 \end{cases}$$

Ex:

$$T(n) = \begin{cases} 1 & , n=1 \\ aT(n/b)+f(n) & , n>1 \end{cases}$$

$T(n) = aT(n/b)+f(n)$

$a=2, b=2, f(n)=n$

$T(n) = 2T(n/2)+n \rightarrow (1)$

$F(n) = \theta(n^d) \rightarrow d=1 \geq 0$

Case 1: $T(n)=\theta(n^d)$ if $2<2^1 \rightarrow F$

Case 2: $a=b, 2=2 \rightarrow F$

then $\boxed{T(n)=\theta(n^d \log_2 n)}$

Applications

1. Binary Search
2. Quick Sort
3. Merge Sort
4. AND-OR-Graphs

(1) Binary Search:

- In Binary Search, the elements should be in Sorted arrays.
- Searching is the process of finding the location of the element.
- The key element is found in the list then search is successful. Otherwise, unsuccessful.
- In Binary Search, the key element is compared with the mid element.
- If the key element is greater than mid element then right sublist is considered. Otherwise, left sublist is considered.
- Mid = (low+high)/2

Where, low = First element index
High = index of last element

- There are two ways to implement binary search
(1) Recursive Binary Search
(2) Iterative Binary Search.

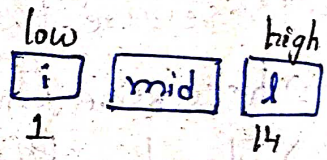
Algorithm for Recursive Binary Search

```
1. Algorithm BinSearch(a, i, l, x)
2. // Given an array a[i...l] of elements in non-
   decreasing order.
3. // x is the key element of the list
4. // x=a[j] return j Otherwise, return 0
5. {
6. if(l==i) then
7. {
```

8. if (x=a[i]) then return i
9. else return 0
10. }
11. else
12. { // p is divided into sub-problems
13. Mid = [(i+l)/2]
14. if (x == a[mid]) then return mid
15. else if (x<a[mid]) then
16. return Binsearch(a, i, mid-1, x)
17. else return Binsearch(a, mid+1, l, x)
18. }
19. }

Tracing

1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-6	0	7	9	23	54	82	101	112	125	131	142	151
[i]													[l]



Key element x = 151

6. if (14==1) → F
12. {
13. mid = [(1+14)/2] = 15/2 = 7
14. if (151==(a[7]=54)) → F
15. else if ((x=151)<(a[7]=54)) → F
17. else return Binsearch(a, mid+1, l, x)

Binsearch(a, i, l, x) → (a, 8, 14, 151)

6. if (14==8) → F
13. mid = [(8+14)/2] = 22/2 = 11
15. else if (151<(a[11]=125)) → F
17. return (a, mid+1, l, x)

(a, 12, 14, 151)

BinSearch(a, i, l, x)
BinSearch(a, 12, 14, 151)

6. if(14 == 12) → F
13. mid = ((12+14)/2) = 26/2 = 13
18. if (151 <= (a[13] = 142)) → F
17. return BinSearch(a, mid+1, l, x)
(a, 14, 14, 151)

BinSearch(a, i, l, x)
BinSearch(a, 14, 14, 151)

6. if (14 == 14) → T
8. if (151 == (a[14] = 151)) → True then return (i = 14)

o/p → 14th position the key element is found.

Algorithm for Iterative Binary Search

1. Algorithm. BinSearch (a, n, x)
2. // given an array[1...n] in non-decreasing order
3. // x is the key element of array list.
4. // if x is present return the position, otherwise return 0
5. {
6. low = 1, high = n
7. while(low <= high) do
8. {
9. mid = (low+high)/2
10. if (x < a[mid]) then high = mid - 1
11. else if (x > a[mid]) then low = mid + 1
12. else return mid.
13. }
14. return 0;
15. }

Tracing

1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-6	0	7	9	23	54	82	101	112	125	131	142	151

n = 14
X = 101

6. low = 1, high = n = 14
7. While (1 <= 14) → T
9. mid = (1+14)/2 = 15/2 = 7
10. if (101 < (a[7] = 54)) → F
11. if (101 > (a[7] = 54)) → T then low = mid+1 = 7+1 = 8

Again

7. While (low <= high) (8 <= 14) → T
9. mid = (low + high)/2 = (8+14)/2 = 22/2 = 11
10. if (X < a[mid]) (101 < (a[11] = 125)) → F then high = mid-1 = 11-1 = 10

Again

7. While (low <= high) (8 <= 14) → T
9. mid = (low + high)/2 = (8+10)/2 = 18/2 = 9
10. if (X < a[mid]) (101 < (a[9] = 101)) → F
11. if (X > a[mid]) (101 > 101) → F
12. else return (mid = 9)

∴ At 9th position key element is found

Time Complexity of Binary Search

Successful Search

1) The middle element is our key element then its takes One Comparision then the Best Case

Best Case → T(C) = O(1)

2) In the Worst Case, the key element is Compared with all the list of elements of length array N. So the time Complexity Worst Case → $O(\log_2 N)$

5) The element is located somewhere in between list of elements. so time complexity of

Average Case → $O(\log_2 N)$

Unsuccessful Search

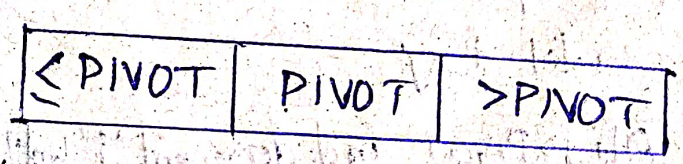
Best Case - $O(\log N)$

Average Case - $O(\log N)$

Worst Case - $O(\log N)$

(2) Quick Sort

1. As the name implies it is the fast Sorting technique.
2. This Algorithm was invented by CAR Hoare
3. Divide the list of elements into two sublists called partitioning.
4. Each sub-list is sorted independently and then merge to get the sorted sub-list.
5. The objective of quick sort is place the PIVOT element in the proper position that means the elements which are present at left side is less than the PIVOT and the elements which are present at Right side is greater than the PIVOT.



Procedure

1: Select first element of array as PIVOT

2: Initialize 'i' and 'j' or first and last elements of the array.

3: Increment 'i' until a[i] > PIVOT then stop.

4: Decrement 'j' until a[j] ≤ PIVOT then stop.

5: If i<j exchange a[i] and a[j]

6: Repeat Step-3,4,5 Until i>j or i and j crossed each other.

7: When i>j or i and j crossed each other then exchange 'j' with PIVOT element.

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
5	7	1	2	10	4	9	3	6	8

1:- 5 7 1 2 10 4 9 3 6 8

2:- PIVOT i j

3:- Increment 'i' Until A[i]≥PIVOT

5 7 1 2 10 4 9 3 6 8

PIVOT i j

4:- Decrement 'j' Until A[j]<PIVOT

5 7 1 2 10 4 9 3 6 8

PIVOT i j

5:- If i<j, [i=2]<[j=8] exchange A[i] and A[j]

5 3 1 2 10 4 9 7 6 8

PIVOT i j

6:- Repeat step-3,4, 5. Until i>j

3,4:- Increment 'i' Until A[i]>PIVOT and decrement 'j' Until A[j]<P

5 3 1 2 10 4 9 7 6 8

PIVOT i j

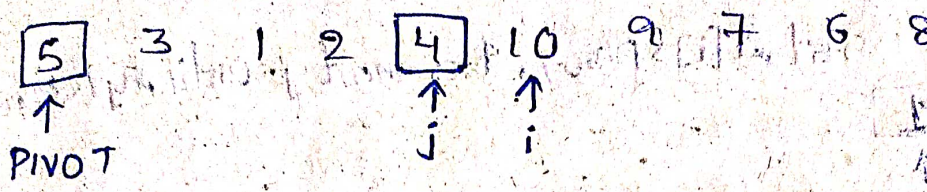
5:- Exchange A[i] and A[j] because [i=5]<[i=6]

5 3 1 2 4 10 9 7 6 8

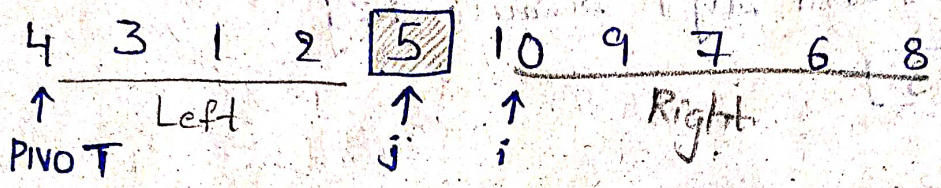
PIVOT i j

6:- Repeat Step 3,4,5 until i ≥ j

3,4:- Increment 'i' until A[i] > PIVOT and Decrement 'j' until A[j] < PIVOT.

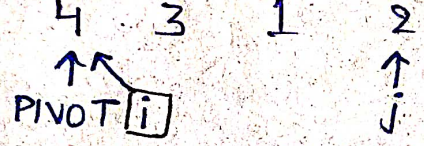


5:- Exchange A[j] and PIVOT element. Because, [i=6] > [i=5]

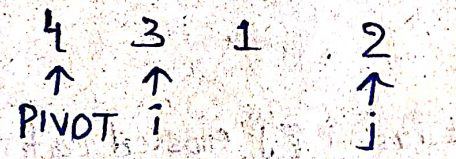


→ Consider Left Sub-list

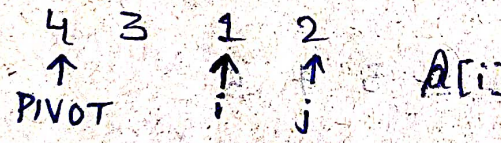
1&2:-



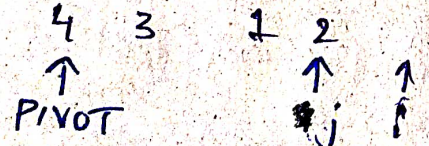
3:- Increment i until A[i] > PIVOT



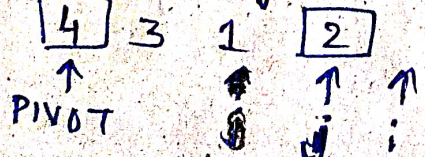
→ Again Increment i until A[i] > PIVOT



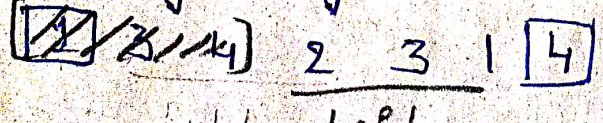
→ Again Increment i until A[i] > PIVOT



4:- Decrement j until A[j] < PIVOT



7:- Exchange A[j] and PIVOT, Because i ≥ j Crossed each other



→ Consider Left sub-list

1,2:- 2 3 1
PIVOT i j

3:- Increment i until A[i] > PIVOT, decrement j until A[j] < PIV

2 3 1
PIVOT i j

5:- Exchange A[i] and A[j] because i < j

2 1 | 3
Left

→ Consider Left sub-list

1,2:- 2 1
PIVOT i j

3:- Increment i until A[i] > PIVOT

2 1
PIVOT j i

7:- Exchange A[j] and PIVOT, because i & j crossed each other

1 | 2

→ Combine Left-Sublist Sort → 1 2 3 4 5

→ Consider Right-sublist

10 9 7 6 8
PIVOT i j

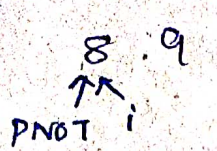
5:- Increment i until A[i] > PIVOT, decrement j until A[j] < PIVOT

10 9 7 6 8
PIVOT j i

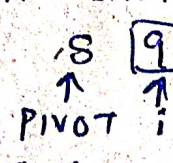
7:- i and j crossed each other Exchange A[j] and PIVOT

8 9 7 6 | 10
Left

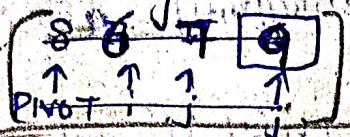
→ Consider Left-sublist



3:- Increment i until A[i]>PIVOT, decrement j until A[j]<PIVOT

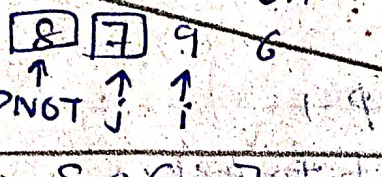


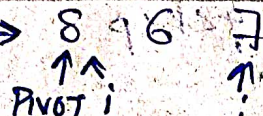
5:- Exchange a[i] and a[j], because i<j



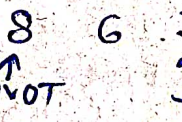
8 6 7 9
Left

Increment i until a[i]>PIVOT, Decrement j until a[j]<PIVOT



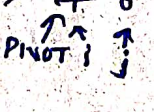


→ Increment i until a[i]>PIVOT

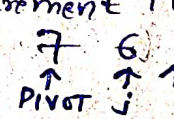


→ Exchange a[j] and PIVOT

7 6 8
Left



→ Increment i until a[i]>PIVOT



→ 6 7

a[j] and PIVOT

Algorithm Quicksort

```
1. Algorithm QuickSort(P, q)
2. // Sort the list of element in an array from a[P]..a[q]
3. // All the elements in an array are stored in the
   form of a[1...n].
4. {
5. if (P<q) then
6. {
7. // Divide P into two partitions
8. j = partition(a, P, q+1)
9. // j is the position of partitioning element
10. Solve each sub-problem independently.
11. QuickSort (P, j-1)
12. QuickSort (j+1, q)
```

```
13. }
14. }
```

Algorithm Partition

```
1. Algorithm partition(a,m,p):
2. // a is an array of the elements a[m],a[m+1]...a[p-
   are the elements in the list
3. // All the elements must be rearranged until we get
   the sorted list.
4. // a[q]=t for some q between m and p-1
5. // a[k]<=t for m<=k<=q and a[k]>=t for q<k<p
6. {
7. v=a[m]
8. i=m, j=p
9. repeat
10. {
11. repeat i=i+1
12. until (a[i]>=v)
13. repeat
14. j=j-1
15. until (a[j]<=v)
16. if(i<j) then Interchange(a,i,j)
17. } until (i>=j)
18. a[m]=a[j], a[j]=v, return j
19. }
```

Algorithm Interchange

```
1. Algorithm Interchange(a,i,j)
2. // Exchange a[i] and a[j]
3. {
```

4. p=a[i]

5. a[i]=a[j]

6. a[j]=p

7. }

Consider the list of element

1	2	3	4	5	6	7	8	9
65	70	75	80	85	60	55	50	45
↓								↓
P=1								q=9

5. if((P=1)<(q=9)) → T

6. //

7. {

8. j = partition(a, 1, 10) ← (a, m, p)

Partition(a, m, p)

2. //

3. //

6. {

7. v = a[m] = a[1] = 65

8. i = [m=1], ~~j=10(p)~~ j=p=10

9. repeat

i=2 until (a[i]>=v) ⇒ a[2]>=65 → T (70)

13. repeat

14. j=j-1 ⇒ j=10-1=9

15. Until (a[j]<=v) ⇒ a[9]<=65 → T (45)

16. if (i<j) then Interchange (a,i,j)

(2<9) → T then Interchange(a,2,9)

Interchange(a, i, j)

Interchange(a, 2, 9)

3. {

4. p = a[i] = a[2] = 70

5. a[i=2] = [a[j=9] = 45]
   a[2] = 45
6. a[j=9] = [p=70]
   a[9] = 70
7. }

List After Interchanging.

1	2	3	4	5	6	7	8	9
65	45	75	80	85	60	55	50	70
↓								↑
P=1								i=9

In partition

#1. repeat i=3 until (a[3] >= v) → T (75 >= 65)

repeat j=9-1=8 until (a[8] <= v) ⇒ (50 <= 65) → T

if (i<j) (3<8) then Interchange(a, 3, 8)

Interchange (a, 3, 8)

{

P = [a[3] = 75]

→ P = 75

a[3] = a[8]

→ a[3] = 50

a[8] = P

→ a[8] = 75

}

List After Interchanging

1	2	3	4	5	6	7	8	9
65	45	50	80	85	60	55	75	70
↓								↓
P								2

In partition

11. Repeat i=4 until (a[4]>=v) → T (80 >= 65)

Repeat j = 8-1 = 7 until (a[7] <= v) → (55 <= 65) → T

if (i<j)

(4<7) then Interchange(a, i, j) → Interchange(a, 4, 7)

Interchange(a,4,7)

{

p = a[4]

→ p = 80

a[4] = a[7]

→ a[4] = 55

a[7] = p

} → a[7] = 80

List After Interchanging

1	2	3	4	5	6	7	8	9
65	45	50	55	85	60	80	75	70
↓ p								↓ q

In partition

11. Repeat i=5 until (a[5]>=v) → T (85 >= 65)

Repeat j = 7-1 = 6 until (a[6] <= v) → (60 <= 65) → T

if (i<j)

(5<6) then Interchange(a, i, j) → Interchange(a, 5, 6)

Interchange(a,5,6)

{

p = a[5]

→ p = 85

a[5] = a[6]

→ a[5] = 60

a[6] = p

} → a[6] = 85

List after Interchanging

1	2	3	4	5	6	7	8	9
65	45	50	55	60	85	80	75	70
↑ P								↑ q

In Partition

1. Repeat i=6 until (a[6] >= v) → T (85 >= 65)

Repeat j=6-1=5 until (a[5] <= v) → F (60 <= 65)

(i >= j) → (6 >= 5) then [PIVOT → m=1, j=5, v=65]

a[m] = a[j], a[j] = v

a[1] = a[5] a[5] = 65 return j=>5

a[r] = 60

List

1	2	3	4	5	6	7	8	9
60	45	50	55	65	85	80	75	70
↓ P=1								↑ q=9

j=5

Quicksort(P, j-1)

↓

Quicksort (1, 4) → 60 45 50 55

Quicksort (j+1, q) → 85 80 75 70

Consider list →

1	2	3	4
60	45	50	55
↑ P			↑ q

Quicksort (p, q) (1, 4)

if (p < q) → T (1 < 4)

j = Partition (a, m, q+1) (a, 1, 5)

In partition (a, m, p) (a, 1, 5)

v = a[m] = a[i] = 60

1. repeat i=1, j=[p=5]

i=2 until (a[2] >= v) => (45 >= 60) → F

j=j-1

j=4 Repeat (a[4] <= v) => (55 <= 60) → T i=5 => (65 >= 60) → T

$(i>j) \rightarrow (5>4)$ $a[j=4] = v$

$a[m=1] = a[j=4]$ $\boxed{a[4] = 60}$ return $\boxed{j \Rightarrow 4}$

$\boxed{a[1] = 55}$

List

1	2	3	4
55	45	50	60
↓p		↓q	

Consider this list

Quick Sort (1, 3) (p, q)

if $(p < q) \rightarrow T$ (a, m, q+1)

j = partition $(a, 1, 4)$

In partition (a, 1, 4)

$v = a[1] = 55$ $i=1$, $j=4$

1. repeat $i=2$ until $(a[2] >= v) \Rightarrow (45 >= 55) \rightarrow F$

$i = 4$ until $(a[4] >= v) \Rightarrow (60 >= 55) \Leftrightarrow T$

$j = 4-1 = 3$ until $(a[3] <= v) \Rightarrow (50 <= 55) \rightarrow T$

$(i \geq j) \rightarrow (4 > 3)$ $a[j=3] = v$

$a[m=1] = a[j=3]$ $\boxed{a[3] = 55}$ return $\boxed{j \Rightarrow 3}$

$\boxed{a[1] = 50}$

List

1	2	3
50	45	55

Consider list

Combining sub-sorted list

$\boxed{45 \quad 50 \quad 55 \quad 60 \quad 65}$

Quick Sort (1, 2)

if $(p < q) \rightarrow T$ (a, m, q+1)

j = partition $(a, 1, 3)$

In partition (a, 1, 3)

$v = a[1] = 50$, $i=1$, $j=3$

1. repeat $i=2$ until $(a[2] >= v) \Rightarrow (45 >= 50) \rightarrow F$

$i=3$ until $(a[3] >= v) \Rightarrow (55 >= 50) \rightarrow T$ $(i \geq j) \Rightarrow (3 > 2)$

$j = 3-1 = 2$ until $(a[2] <= v) \Rightarrow (45 <= 50) \rightarrow T$

$a[i] = a[j=2]$ $a[j=2] = v$ List

$\boxed{a[1] = 45}$ $\boxed{a[2] = 50}$ $\boxed{45 \quad 50 \quad 55}$

Time Complexity of Quicksort

(1) Average Case

In Quick Sort we can divide the list into two parts so the recurrence relation for the quick sort is defined as

$$T(n) = 2T(n/2) + Cn \rightarrow (1)$$

$n = n/2$ in eq(1) :- $T(n/2) = 2T(n/4) + C(n/2) \rightarrow (2)$

$n = n/4$ in eq(1) :- $T(n/4) = 2T(n/8) + C(n/4) \rightarrow (3)$

Sub in eq(1) $\rightarrow$ $T(n) = 2T(n/2) + Cn$

$$= 2[2T(n/4) + C(n/2)] + n$$

$$= 2^2 T(n/4) + 2Cn$$

$$= 2^2[2T(n/8) + C(n/4)] + 2Cn$$

$$= 2^3 T(n/8) + 3Cn$$

$$T(n) = 2^k T(n/2^k) + kCn$$

$n = 2^k$

$k = \log_2 n$

$$= 2^k T(n/n) + \log_2 n \, Cn$$

$$= 2^k T(1) + \log_2 n \cdot Cn$$

$$= 2^k(1) + \log_2 n \, Cn$$

$$T(n) = n + Cn \log_2 n$$

$$\boxed{T(n) = n \log_2 n} \leftarrow$$

(2) Worst Case

The Worst case time Complexity in quick sort is defined in the form of $T(n) = T(n-1) + cn$

Because, the n element, the (n-1) comparisons are required

$$T(n) = T(n-1) + cn \rightarrow (1)$$

$n = n-1$ in eq(1) $\rightarrow T(n-1) = T(n-2) + c(n-1) \rightarrow (2)$

$n = n-2$ in eq(1) $\rightarrow T(n-2) = T(n-3) + c(n-2) \rightarrow (3)$

Substitute in eq(1)

$$T(n) = T(n-1) + cn$$
$$= T(n-2) + c(n-1) + cn$$
$$= T(n-3) + c(n-2) + c(n-1) + cn$$
$$= T(n-n) + c(n-2+n-1+n)$$
$$= T(0) + \frac{n(n+1)}{2}$$
$$= 0 + \frac{n^2}{2} + \frac{n}{2}$$

$$\boxed{T(n) = O(n^2)}$$

Merge Sort

Merge Sort also follows the Divide or Conquer Strategy.

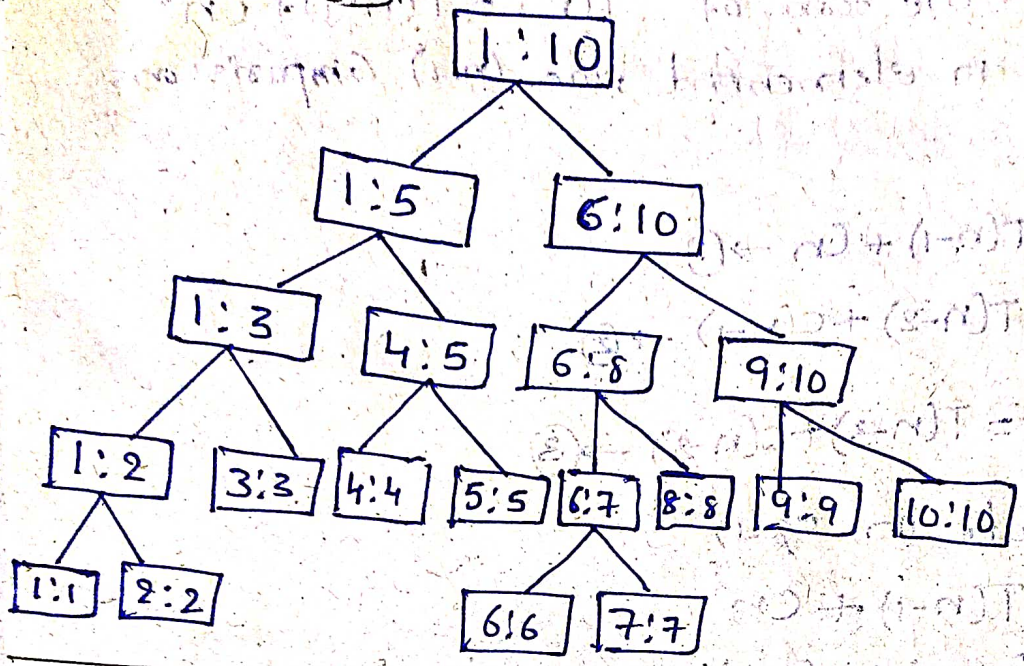
→ In DIVIDE the list of elements are divided until we get the single element in the list.

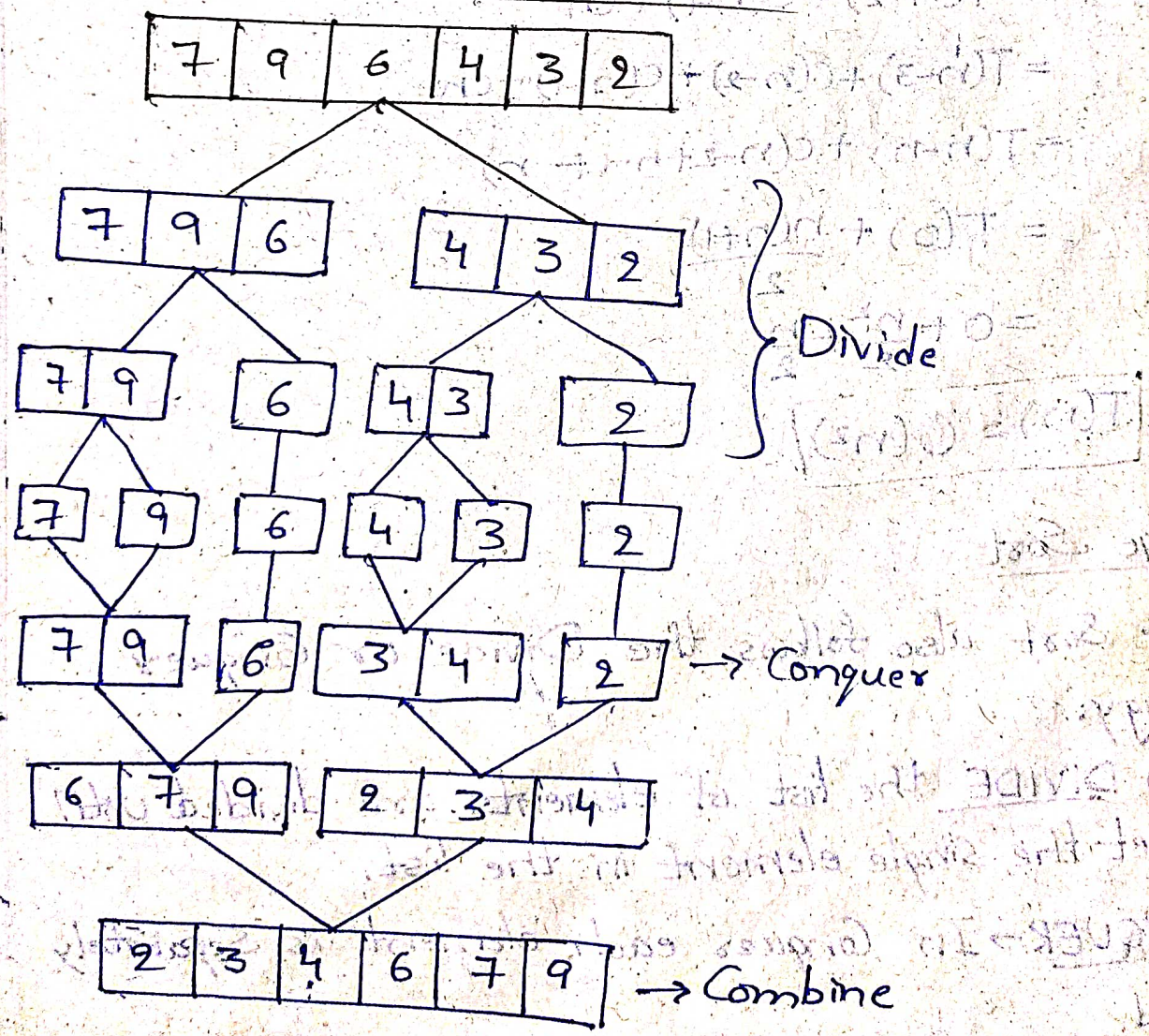
→ CONQUER → In Conquer each element is Separately Sorted.

COMBINE → In Combine all the list of elements are Combined together as a single list

The diagramatic Representation of Merge Sort as follows:

a[1:10] = [310, 285, 179, 652, 351, 423, 861, 254, 450, 520]





Algorithm MergeSort

```
1. Algorithm MergeSort(low, high)
2. // a[low: high] is a global array to be sorted
3. // small(P) is TRUE if only one element in the list
4. // All the elements are stored in an array
5. {
6. if (low < high) then
7. {
8. // Divide P into sub-problems
9. // Find the Middle element of the list mid = (low+high)/2
10. mid = (low+high)/2
11. // Solve the subproblems
12. MergeSort(low, mid)
13. MergeSort(mid+1, high)
14. // Combine the sub-problems
15. Merge(low, mid, high)
16. }
17. }
```

Example

```
MergeSort(1, 10)
{
  if(1<10) → T
  mid = (1+10)/2 = 5
  MergeSort(1, 5) → Fun-1
  MergeSort(6, 10) → Fun-2
  Merge(1, 5, 10) → Fun-3
}
```

Fun-1

MergeSort(1,5)
{
if(1<5) → T
mid = (1+5)/2 = 3
MergeSort(1, 3) → Fun-4
Mergesort(4, 5) → Fun-5
Merge(1,3,5) → Fun-6
}

Fun-2

Mergesort(6, 10)
{
if(6<10) → T
mid = (6+10)/2 = 8
MergeSort(6, 8) → Fun-7
MergeSort(9, 10) → Fun-8
Merge(6, 8, 10) → Fun-9
}

Fun-4

Mergesort(1, 3)
{
if(1<3) → T
mid = (1+3)/2 = 2
Mergesort(1, 2) → Fun-10
Mergesort(3, 3)
Merge(1, 2, 3) → Fun-11
}

Fun-5

Mergesort(4, 5)
{
if(4<5) → T
mid = (4+5)/2 = 4
Mergesort(4, 4)
Mergesort(5, 5)
Merge(4, 4, 5) → Fun-12
}

Fun-7

Mergesort(6, 8)
{
if(6<8) → T
mid = (6+8)/2 = 7
Mergesort(6, 7) → Fun-13
Mergesort(8, 8)
Merge(6, 7, 8) → Fun-14
}

Fun-8

Mergesort(9, 10)
{
if(9<10) → T
mid (9+10)/2 = 9
Mergesort(9, 9)
MergeSort(10, 10)
Merge(9, 9, 10) → Fun-15
}

Fun-10

Mergesort(1, 2)
{
if(1<2) → T
mid = (1+2)/2 = 1
Mergesort(1, 1)
Mergesort(2, 2)
Merge(1, 1, 2) → Fun-16
}

Fun-13

Mergesort(6, 7)
{
if(6<7) → T
mid = (6+7)/2 = 6
Mergesort(6, 6)
Mergesort(7, 7)
Merge(6, 6, 7) → Fun-17
}

Algorithm Merge

```
1. Algorithm Merge (low, mid, high)
2. // a[low: high] is global array Containing two sub-sets.
3. // a[low : mid] and a[mid+1 : high] is two sub-sets.
4. // The goal is to merge two sub-sets into a
   single sorted list.
5. // b[] is a Auxilary global Array.
6. {
7. h=low, i=low, j=mid+1
8. While((h<=mid) and (j<=high)) do
9. {
10. if (a[h]<=a[j]) then
11. {
12. b[i]=a[h], h=h+1
13. }
14. else
15. {
16. b[i]=a[j], j=j+1
17. }
18. i=i+1;
19. }
20. if (h>mid) then
21. for k=j to high do
22. {
23. b[i]=a[k], i=i+1
24. }
25. else
```

26. for k=h to mid do
27. {
28. b[i]=a[k], i=i+1
29. }
30. for k=low to high do
31. a[k]=b[k]
32. }

Tracing

a[1]	a[2]	a[3]	a[4]	a[5]
285	310	179	652	351

Fun-11 (low mid high)

Merge (1, 2, 3)

7. h=1, i=1, j=2+1=3
8. While ((1<=2) and (3<=3)) → T do (h<=mid, j<=high)
9. {
10. if ((a[h=1]=285)<=[a[j=3]=179]) → F
14. else
16. b[i] = a[h=3] ⇒ b[1]=179, j=3+1=4
18. i=1+1=2

Again Repeat While ((1<=2) and (4<=3)) → F do (h<=mid, j<=high)

10. if ((a[h=1]=285)<=[a[j=4]=652]) → T
20. if (h>mid) (1>2) → F
25. else
26. for k=h=1 to mid=2 do
27. {
28. b[i=2]=a[k=1] ⇒ b[2]=285, i=2+1=3

Again for loop

26. for k=2 to 2 do

28. b[i=3] = a[k=2] ⇒ **b[3] = 310**, i = 3+1 = 4

30. for k = low 1 to high 3 do

31. a[k] = b[k] = 179

k = 2 to 3 do

a[2] = b[2] = 285

a[3] = b[3] = 310

→ After Merge (1, 2, 3) the list of elements are

a[1]	a[2]	a[3]	a[4]	a[5]
179	285	310	652	351

Fun-12

Merge(4, 4, 5) (low, mid, high)

7. h = low 4, i = low 4, j = mid+1 = 4+1 = 5

8. While ((h<=mid 4<=4) and (j<=high 5<=5)) → T do

9. {

10. if ((a[h=4] = 652) <= (a[j=5] = 351)) → F

14. else

16. b[i=4] = a[h=5] ⇒ **b[4] = 351**, j = 5+1 = 6

18. i = 4+1 = 5

Again Repeat While ((h<=mid 4<=4) and (j<=high 6<=5)) → F

20. if (h>mid 4 > 4) → F

25. else

26. for k = h 4 to mid 4 do

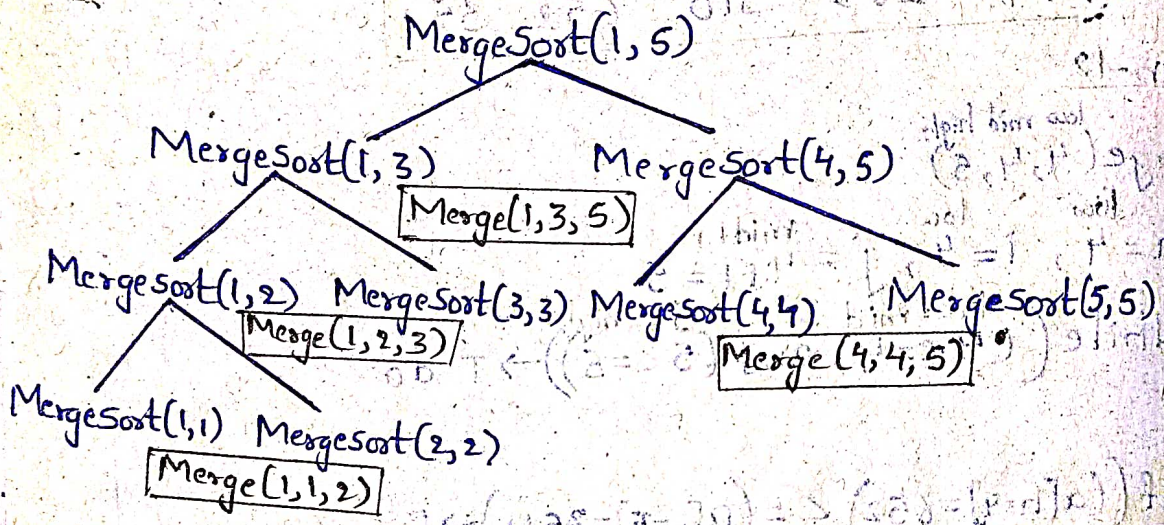
28. b[i=5] = a[k=4] ⇒ **b[5] = 652**, i = 5+1 = 6

30. for k=4 to 5 do
31. a[4]=b[4]=351
again k=5 to 5 do
31. a[5]=b[5]=652
32. }

→ After merge(4,4,5) the list of elements are

a[1]	a[2]	a[3]	a[4]	a[5]
179	285	310	351	652

Tree Representation of MergeSort



Time Complexity for Merge Sort

The Time Complexity for Merge Sort can be represented using the Recurrence Relation.

$$T(n)=\begin{cases} a & \text{Where } n=1 \\ 2T(n/2)+cn & \text{Where } n>1 \end{cases}$$

$$T(n)=2T(n/2)+cn \rightarrow (1)$$

$\underline{n=n/2 \text{ in eq(1)}} \Rightarrow T(n/2)=2T(n/4)+c(n/2) \rightarrow (2)$

$\underline{n=n/4 \text{ in eq(1)}} \Rightarrow T(n/4)=2T(n/8)+c(n/4) \rightarrow (3)$

sub eq(2), eq(3) in eq(1)

$$T(n) = 2T(n/2) + cn$$

$$= 2[2T(n/4) + c(n/2)] + cn$$

$$= 2^2 T(n/4) + 2cn$$

$$= 2^2[2T(n/8) + c(n/4)] + 2cn$$

$$T(n) = 2^3 T(n/2^3) + 3cn$$

$2^k = n$

$k = \log_2 n$

$$T(n) = 2^k T(n/2^k) + kcn$$

$$= n\,T(n/n) + \log_2 n\, cn$$

$$= n\,T(1) + \log_2 n\, cn$$

$$T(n) = na + \log_2 n\, cn$$

$$T(n) = na + n\log_2 n\, c$$

$$\boxed{T(n) = O(n\log_2 n)} \leftarrow \text{Time Complexity}$$

Best Case - $O(n\log_2 n)$

Average Case - $O(n\log_2 n)$

Worst Case - $O(n\log_2 n)$

Advantages of Merge Sort

→ Unnecessary Comparisions are avoided

→ Time will be save

Disadvantages of Merge Sort

→ It takes more memory

→ For each Recursive call we need to copy the elements from temporary array to original array because of this stack space is wasted.

AND/OR graphs

This graphs are useful for representing the solution of problems that can be decomposed into several sub-problems.

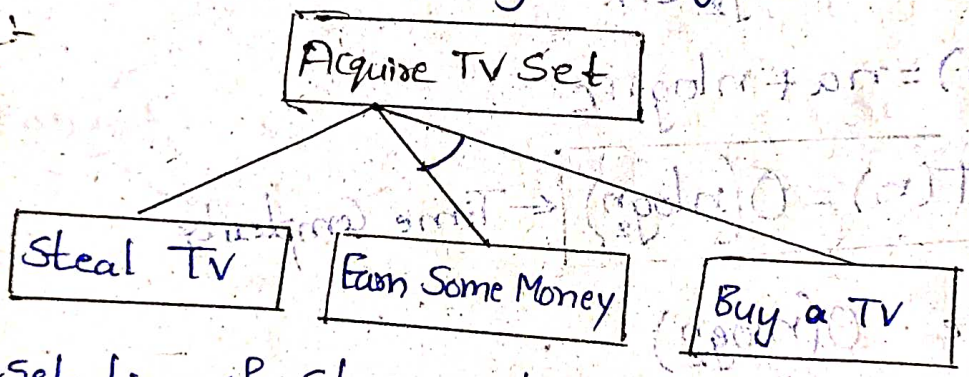
→ Each sub-problem is represented in the form of AND / OR.

→ AND node represents the decomposition of a problem and it is represented using "Arc" symbol in the graph.

→ OR node representing the decomposition of a problem between choices.

→ Nodes Representing using circles. Nodes with no children Represents using Rectangles.

Ex:-



→ Selection of Choice depends upon the cost function

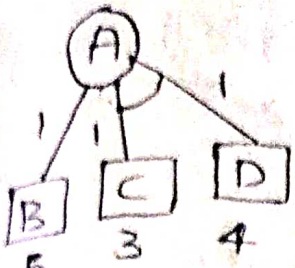
→ So AND-OR-graph consists of three variables [G, S, T]

G → It is Implicitly specified of AND-OR-graph

S → Starting node of AND-OR-graph

T → Terminal nodes for the graph then

h(n) → is the heuristic function which is used to calculate the cost of the node.



h(n) = 6 h(n) = 9

GREEDY METHODS

Introduction:-

For the given set of n inputs we need to find out the subset of solutions which satisfies the constraints. Any subset that satisfies the constraints is called as Feasible Solution.

→ Any Feasible Solution that maximises or minimises the given objective function is called as Optimal Solution.

→ The Main objective of Greedy method is to find the Optimal Solution.

Control Abstraction

```
1. Algorithm Greedy(a,n)
2. //a[1:n] contains the 'n' inputs
3. {
4. Solution=∅ // initialize the solution
5. for i=1 to n do
6. {
7. X= Select(a)
8. if Feasible(Solution,X) then
9. Solution=Union(Solution,X)
10. }
```

11. return Solution
12. }

Applications:-

1) Job Sequencing with dead lines
2) Knapsack problem
3) Minimum Cost Spanning tree
4) Single source Shortest path

(1) Job Sequencing with dead Lines

PROCEDURE:

→ There are 'n' Jobs are given There is a associated dead Line for each ang every Job which is denoted as d_i, where $d_i \geq 0$

→ And for each Job there is a profit that is denoted by P_i, where $P_i > 0$ The profit is earned if the Job is completed by its dead Line.

→ Only One Machine is available to process the Jobs

A Feasible Solution is Subset of Jobs such that each Job is Completed by its dead Line

→ The Optimal Solution is which Job Sequence Use the maximum profit.

Rules:-

- Each Job takes 1 unit of time
- If The Job starts before or atleast dead Line then the profit is earned, Otherwise no profit.
- Goal is to Schedules Jobs with Maximum Profit.

Ex:(1) Find the Optimal Solution for the following

	n	P_i	d_i
J_1 –	1	70	2
J_2 –	2	12	1
J_3 –	3	18	2
J_4 –	4	35	1

Step 1: Arrange all the jobs in ~~non~~-decreasing order

Job	J_1	J_4	J_3	J_2
Profit	70	35	18	12
deadline	2	1	2	1

Step 2:

J_4	J_1
0	1 2

Output Job slot	Assigned Slot	Selected Job	Action	Profit
$\emptyset$	$\emptyset$	J_1	Assign[1,2]	70
$\{J_1\}$	[1,2]	J_1, J_4	Assign[0,1]	105 {70+35}
$\{J_1, J_4\}$	[0,1] [1,2]	J_3	Reject	105
$\{J_1, J_4\}$	[0,1] [1,2]	J_2	Reject	105

$\{J_1, J_4\} = 105$

Second Method

1	2	3	4
35	70		
J_4	J_1		

$\{J_4, J_1\} = (70+35)$

$\{J_1, J_4\} = 105$

(2) Let $n=4$, $(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$ and Corresponding deadlines $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

Step 1: Arrange all the Jobs in decreasing Order

Jobs	J_1	J_4	J_3	J_2
Profit	100	27	15	10
deadline	2	1	2	1

Step 2:

J_4	J_1
0	1	2

Output Job slot	Assigned slot	Selected Job	Action	Profit
$\emptyset$	$\emptyset$	J_1	Assign[1,2]	100
$\{J_1\}$	[1,2]	J_1, J_4	Assign[0,1]	127 [100+27]
$\{J_1, J_4\}$	[0,1], [1,2]	J_3	Reject	127
$\{J_1, J_4\}$	[0,1] [1,2]	J_2	Reject	127

Output job slot $\{J_1, J_4\} = 127$

Second Method

1	2	3	4
J_4	J_1		

$\{J_4, J_1\} = 127$

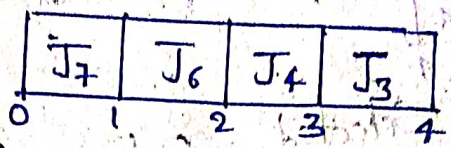
(3) Let $n=7$, $(P_1, P_2, P_3, P_4, P_5, P_6, P_7) = (3, 5, 20, 18, 1, 6, 30)$ the deadline are $(d_1, d_2, d_3, d_4, d_5, d_6, d_7) = (1, 3, 4, 3, 3, 2, 1)$ Find the Optimal solution

n	P_i	d_i
1	3	1
2	5	3
3	20	4
4	18	3
5	1	3
6	6	2
7	30	1

Step 1: Arrange all Jobs in decreasing order

Jobs	J_7	J_3	J_4	J_6	J_2	J_1	J_5
Profit	30	20	18	6	5	3	1
deadline	1	4	3	2	3	1	3

Step 2:



Output job slot	Assigned slot	Selected Job	Action	Profit
$\emptyset$	$\emptyset$	J_7	Assign[0,1]	30
$\{J_7\}$	[0,1]	J_7, J_3	Assign[3,4]	50 [30+20]
$\{J_7, J_3\}$	[3,4][0,1]	J_7, J_3, J_4	Assign[2,3]	68 [50+18]
$\{J_7, J_3, J_4\}$	[3,4][0,1][2,3]	J_7, J_3, J_4, J_6	Assign[1,2]	74 [68+6]
$\{J_7, J_3, J_4, J_6\}$	[3,4][0,1][2,3][1,2]	J_2	Reject	74
$\{J_7, J_3, J_4, J_6\}$	[3,4][0,1][2,3][1,2]	J_1	Reject	74
$\{J_7, J_3, J_4, J_6\}$	[3,4][0,1][2,3][1,2]	J_5	Reject	74

Output Job slot $\{J_7, J_3, J_4, J_6\} = 74$

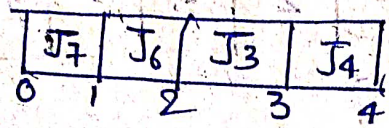
(4)

n	P_i	d_i
1	3	1
2	5	3
3	18	3
4	20	4
5	6	1
6	1	2
7	38	1

Step 1: Arrange in descending Order.

Jobs	J_7	J_4	J_3	J_5	J_2	J_1	J_6
Profit	38	20	18	6	5	3	1
deadline	1	4	3	1	3	1	2

Step 2:



Output Job slot	Assigned slot	Selected Job	Action	Profit
$\emptyset$	$\emptyset$	J_7	Assign[0,1]	38
$\{J_7\}$	[0,1]	J_7, J_4	Assign[3,4]	58 [38+20]
$\{J_7, J_4\}$	[0,1][3,4]	J_7, J_4, J_3	Assign[2,3]	76 [58+18]
$\{J_7, J_4, J_3\}$	[0,1][3,4][2,3]	J_5	Reject	76
$\{J_7, J_4, J_3\}$	[0,1][3,4][2,3]	J_2	Reject	76
$\{J_7, J_4, J_3\}$	[0,1][3,4][2,3]	J_1	Reject	76
$\{J_7, J_4, J_3\}$	[0,1][3,4][2,3]	J_7, J_4, J_3, J_6	Assign[1,2]	77 [76+1]

Output Job slot $\{J_7, J_4, J_3, J_6\} = 77$

D -> deadline
P -> Profit
n -> Number of Jobs
dmax -> Maximum deadline

Algorithm Job Sequence

```
Algorithm JS(D, P, n, dmax)
{
  maxprofit = 0
  for i=1 to n do
  {
    k=min(dmax, D[i])
    while(k>=1) do
    {
      if(timeslot[k] is false) then
      {
        timeslot[k]=J[i]
```

```
maxprofit = maxprofit + p[i]
break;
k = k+1;
}
}
```

Ex:

	1	2	3	4	5	
	J_1	J_2	J_3	J_4	J_5	→ J
	2	1	3	2	1	→ D
	60	100	20	40	20	→ P

n = 5

→ Arrange in descending Order

	J_2	J_1	J_4	J_3	J_5	
	100	60	40	20	20	→ P
	1	2	2	3	1	→ D

Timeslot	1	2	3
status	False	False	False

True

Tracing :-

Iteration - 1

dmax = 3 , n = 5

maxprofit = 0

for i = 1 to 5 do

{

k = min(3, (D[1] = 1)) = min(3, 1)

k = 1

While (1 >= 1) do

{

if (timeslot[1]) is false then (false)

{

timeslot[1] = J[1] = J_2

maxprofit = 0 + p[1] = 100

k = 2

}

}

}

Again

```
while(2>=1)
{
  if(timeslot[2]) is false then   (false)
  {
    timeslot[2] = J[2] = J1
    maxprofit = 100 + 60 = 160
    k = 3
  }
}
```

Again

```
while(3>=1)
{
  if(timeslot[3]) is false then   (false)
  {
    timeslot[3] = J[3] =
```

2) Knapsack Problem:-

Introduction:

- There are 'n' objects and a knapsack (Bag) and every object having weight called 'w_i' and knapsack has the capacity of 'm'.
- If a fraction 'x_i' is in $0 \le x_i \le 1$ of object 'i' is placed into the knapsack then the profit of '$P_i x_i$' is earned.
- The objective is to obtain a filing of the knapsack that maximizes the total profit earn.
- The problem can be stated as

1. Maximize $\sum_{1 \le i \le n} P_i x_i$, $1 \le i \le n \rightarrow (1)$
2. Subject to $\sum_{1 \le i \le n} w_i x_i \le m \rightarrow (2)$
3. and $0 \le x_i \le 1$, $1 \le i \le n \rightarrow (3)$

NOTE:- The profits and weights are positive numbers

Ex:① Find the Optimal Solution for the following knapsack problem where $n=3$, $m=20$ and there are three profits $(P_1, P_2, P_3) = (25, 24, 15)$ and Weights are $(w_1, w_2, w_3) = (18, 15, 10)$

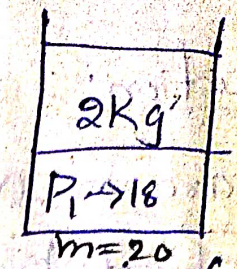
n → total number of objects
m → knapsack capacity

Case 1: Maximum Profit:-

→ In this case which objects having maximum profit is placed into the knapsack.

→ The object 'X_1' having more profit; so X_1 is Completely Placed into the knapsack.

Therefore, $\boxed{X_1 = 1}$



→ After placing 'X_1' the remaining Space is (20-18=2) '2 units is left.

→ Next highest Profit is 'X_2', Then

$$X_2 = \frac{\text{Left out of Space}}{\text{Weight of item to be placed}}$$

$$\boxed{X_2 = \frac{2}{15}}$$

→ There is no place in the knapsack. So, $\boxed{X_3 = 0}$

1. $\sum_{i \le i \le n} P_i X_i = P_1X_1 + P_2X_2 + P_3X_3$

$= 25 \times 1 + 24 \times \frac{2}{15} + 15 \times 0$

$= 25 + \frac{48}{15}$

$\boxed{\sum P_i X_i = 28.2}$

2. $\sum_{i \le i \le n} w_i x_i = w_1x_1 + w_2x_2 + w_3x_3$

$= 18 \times 1 + 15 \times \frac{2}{15} + 10 \times 0$

$= 18 + 2 + 0$

$\boxed{\sum w_i x_i = 20}$

3. $\underline{0 \le x_i \le 1}$

$0 \le x_1 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$0 \le x_2 \le 1 \Rightarrow 0 \le \frac{2}{15} \le 1 \rightarrow T$

$0 \le x_3 \le 1 \Rightarrow 0 \le 0 \le 1 \rightarrow T$

$1 \le i \le n$ $[n=3, i=(1,2,3...,n)]$

$1 \le 1 \le 3 \rightarrow T$

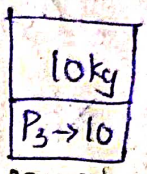
$1 \le 2 \le 3 \rightarrow T$

$1 \le 3 \le 3 \rightarrow T$

Case 2: Minimum Weight :-

→ The object 'X_3' having less profit, so X_3 is completely placed into knapsack.

Therefore, $\boxed{X_3 = 1}$



→ After placing 'X_3' the remaining space is (20-10 = 10) 10 units is left.

→ Next lowest profit is 'X_2', then

$$X_2 = \frac{10}{15}$$

$$\boxed{X_2 = \frac{2}{3}}$$

→ There is no place in knapsack, so $\boxed{X_1 = 0}$

1. $\sum P_i X_i = P_1 X_1 + P_2 X_2 + P_3 X_3$

$1 \le i \le n$

$= 25 \times 0 + 24 \times \frac{2}{3} + 15 \times 1$

$= 16 + 15$

$\boxed{\sum P_i X_i = 31}$

2. $\sum w_i X_i = w_1 X_1 + w_2 X_2 + w_3 X_3$

$1 \le i \le n$

$= 18 \times 0 + 15 \times \frac{2}{3} + 10 \times 1$

$= 0 + 10 + 10$

$\boxed{\sum w_i X_i = 20}$

3. $0 \le x_i \le 1$

$0 \le x_1 \le 1 \Rightarrow 0 \le 0 \le 1 \rightarrow T$

$0 \le x_2 \le 1 \Rightarrow 0 \le \frac{2}{3} \le 1 \rightarrow T$

$0 \le x_3 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$1 \le i \le n$

$1 \le 1 \le 3 \rightarrow T$

$1 \le 2 \le 3 \rightarrow T$

$1 \le 3 \le 3 \rightarrow T$

Case 3: $x_i = \frac{P_i}{w_i}$ → Profit per Unit weight

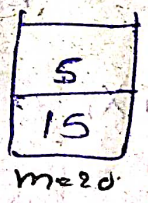
$x_1 = \frac{P_1}{w_1} = \frac{25}{18} = 1.38 \rightarrow 1.4$ (approx)

$x_2 = \frac{P_2}{w_2} = \frac{24}{15} = 1.6$

$x_3 = \frac{P_3}{w_3} = \frac{15}{10} = 1.5$

→ 'x_2' has more profit per weight

$\boxed{x_2 = 1}$



→ After placing 'x_2' remaining space (20-15=5) 5 units left

$x_3 = \frac{5}{10} = \frac{1}{2}$

$\boxed{x_3 = 0.5}$

→ $\boxed{x_1 = 0}$

1. $\sum P_i x_i = P_1 x_1 + P_2 x_2 + P_3 x_3$

$0 \le i \le n$

$= 25 \times 0 + 24 \times (1) + 15 \times (0.5)$

$= 24 + 7.5$

$\boxed{\sum P_i x_i = 31.5}$

2. $\sum w_i x_i = w_1x_1 + w_2x_2 + w_3x_3$

$0 \le i \le n$ $= 18 \times 0 + 15 \times 1 + 10 \times \frac{1}{2}$

$= 15 + 5$

$\boxed{\sum w_i x_i = 20}$

3. $0 \le x_i \le 1$

$0 \le x_1 \le 1 \Rightarrow 0 \le 0 \le 1 \rightarrow T$

$0 \le x_2 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$0 \le x_3 \le 1 \Rightarrow 0 \le 0.5 \le 1 \rightarrow T$

$1 \le i \le n$

$1 \le 1 \le 3 \rightarrow T$

$1 \le 2 \le 3 \rightarrow T$

$1 \le 3 \le 3 \rightarrow T$

Total Possible Solution

Case	$\sum P_i X_i$	$\sum w_i x_i$
1	28.2	20
2	31	20
3	31.5	20

1, 2, 3 are called as feasible Solutions

3 - Optimal solution.

2) Find the Optimal solution for the following data

$\{(60,10)\ (100,20)\ (120,30)\}$, $w = 50$ (or) m

$(P_1, P_2, P_3) = (60, 100, 120)$

$(w_1, w_2, w_3) = (10, 20, 30)$

Case 1: Maximum Profit

$x_3 = 1$

$x_2 = 1$

$x_1 = 0$

0 kg
20
(50-30)=20
30

m=50

1. $\sum_{0 \le i \le 1} P_i x_i = P_1 x_1 + P_2 x_2 + P_3 x_3$

$= 60(0) + 100(1) + 120 \times 1$

$\sum P_i x_i = 100 + 120 = 220$

2. $\sum_{0 \le i \le 1} w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3$

$= 10(0) + 20(1) + 30 \times 1$

$\sum w_i x_i = 30 + 20 = 50$

3. $0 \le x_i \le 1$

$1 \le i \le n$

$0 \le x_1 \le 1 \rightarrow 0 \le 0 \le 1 \rightarrow T$

$1 \le 1 \le 3$

$0 \le x_2 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$1 \le 2 \le 3$

$0 \le x_3 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$1 \le 3 \le 3$

Case 2: Minimum Profit Weight

$x_1 = 1$

$x_2 = 1$

$x_3 = \frac{20}{30} = \frac{2}{3}$

20 kg
20
40 kg
10

m=50

1. $\sum_{0 \le i \le n} P_i x_i = P_1 x_1 + P_2 x_2 + P_3 x_3$

$= 60 \times 1 + 100 \times 1 + 120 \times \frac{2}{3}$

$= 160 + 80$

$\sum P_i x_i = 240$

2. $\sum_{0 \le i \le n} w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3$

$= 10 \times 1 + 20 \times 1 + 30 \times \frac{2}{3}$

$= 10 + 20 + 20$

$\sum w_i x_i = 50$

3. $0 \le x_i \le 1$ $\quad 1 \le i \le n$

$0 \le x_1 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$ $\quad 1 \le 1 \le 3$

$0 \le x_2 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$ $\quad 1 \le 2 \le 3$

$0 \le x_3 \le 1 \Rightarrow 0 \le \frac{2}{3} \le 1 \rightarrow T$ $\quad 1 \le 3 \le 3$

Case 3: Profit per unit weight $x_i = \frac{P_i}{w_i}$

$x_1 = \frac{P_1}{w_1} = \frac{60}{10} = 6$

$x_2 = \frac{P_2}{w_2} = \frac{100}{20} = 5$

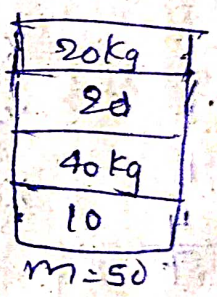
$x_3 = \frac{P_3}{w_3} = \frac{120}{30} = 4$

→ x_1 has Max profit

$x_1 = 1$

$x_2 = 1$

$x_3 = \frac{20}{30} = \frac{2}{3}$



1. $\sum_{0 \le i \le n} P_i x_i = P_1 x_1 + P_2 x_2 + P_3 x_3$

$= 60 \times 1 + 100 \times 1 + 120 \times \frac{2}{3}$

$\sum P_i x_i = 240$

2. $\sum w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3$

$= 10 \times 1 + 20 \times 1 + 30 \times \frac{2}{3}$

$\sum w_i x_i = 50$

3. $0 \le x_i \le 1$ $\quad 1 \le i \le n$

$0 \le x_1 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$ $\quad 1 \le 1 \le 3$

$0 \le x_2 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$ $\quad 1 \le 2 \le 3$

$0 \le x_3 \le 1 \Rightarrow 0 \le \frac{2}{3} \le 1 \rightarrow T$ $\quad 1 \le 3 \le 3$

Total Possible Solution

Case	$\sum P_i x_i$	$\sum w_i x_i$
1	220	50
2	240	50
3	240	50

1, 2, 3 are ~~feasible~~ Solutions

2, 3 are Optimal solutions

(3) Consider $n=7$, $m=15$ and the profits are $(10, 5, 15, 7, 6, 18, 3)$ ($x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7$), Weights are $(2, 3, 5, 7, 1, 4, 1)$ ($x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7$). Find the feasible solution and Optimal solution

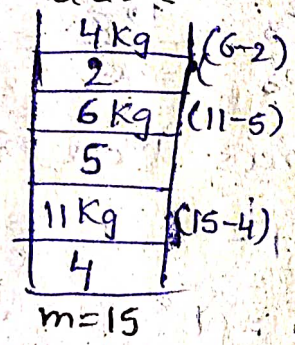
Case 1: Maximum Profit

$x_6 = 1$ $\quad x_5 = 0$

$x_3 = 1$ $\quad x_7 = 0$

$x_1 = 1$ $\quad x_2 = 0$

$x_4 = \frac{4}{7}$



1. $\sum_{0 \le i \le 1} P_i x_i = P_1x_1 + P_2x_2 + P_3x_3 + P_4x_4 + P_5x_5 + P_6x_6 + P_7x_7$

$= 10\times1 + 5\times0 + 15\times1 + 7\times\frac{4}{7} + 6\times0 + 18\times1 + 3\times0$

$= 10 + 15 + 4 + 18 + 0$

$\sum P_i x_i = 47$

2. $\sum_{0 \le i \le n} w_i x_i = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + w_7x_7$

$= 2\times1 + 3\times0 + 5\times1 + 7\times\frac{4}{7} + 1\times0 + 4\times1 + 1\times0$

$= 2 + 5 + 4 + 4$

$\sum w_i x_i = 15$

3. $0 \le x_i \le 1$

$0 \le x_1 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$ $\quad 0 \le x_5 \le 1 \Rightarrow 0 \le 0 \le 1 \rightarrow T$

$0 \le x_2 \le 1 \Rightarrow 0 \le 0 \le 1 \rightarrow T$ $\quad 0 \le x_6 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$0 \le x_3 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$ $\quad 0 \le x_7 \le 1 \Rightarrow 0 \le 0 \le 1 \rightarrow T$

$0 \le x_4 \le 1 \Rightarrow 0 \le \frac{4}{7} \le 1 \rightarrow T$

$1 \le i \le n$

$1 \le 1 \le 7$ $\quad 1 \le 4 \le 7$

$1 \le 2 \le 7$ $\quad 1 \le 5 \le 7$ $\quad 1 \le 7 \le 7$

$1 \le 3 \le 7$ $\quad 1 \le 6 \le 7$

Case 2: Minimum Weight

$x_7 = 1$

$x_2 = 1$ $\quad x_3 = \frac{1}{5}$

$x_5 = 1$ $\quad x_6 = 0$

$x_4 = 1$

$x_1 = 1$

1kg	(3-2)
2	
3kg	(10-7)
7	
10kg	(11-1)
1	
11kg	(14-3)
3	
14kg	(15-1)
1	

$m = 15$

1. $\sum_{0 \le i \le 1} P_i x_i = P_1x_1 + P_2x_2 + P_3x_3 + P_4x_4 + P_5x_5 + P_6x_6 + P_7x_7$

$= 10 \times 1 + 5 \times 1 + 15 \times \frac{1}{5} + 7 \times 1 + 6 \times 1 + 18 \times 0 + 3 \times 1$

$= 10 + 5 + 3 + 7 + 6 + 3$

$\sum P_i x_i = 24$

2. $\sum_{0 \le i \le n} w_i x_i = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + w_7x_7$

$= 2 \times 1 + 3 \times 1 + 5 \times \frac{1}{5} + 7 \times 1 + 1 \times 1 + 4 \times 0 + 1 \times 1$

$= 2 + 3 + 1 + 7 + 1 + 1$

$\sum w_i x_i = 15$

3. $0 \le x_i \le 1$

$0 \le x_1 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$0 \le x_2 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$0 \le x_3 \le 1 \Rightarrow 0 \le \frac{1}{5} \le 1 \rightarrow T$

$0 \le x_4 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$0 \le x_5 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$0 \le x_6 \le 1 \Rightarrow 0 \le 0 \le 1 \rightarrow T$

$0 \le x_7 \le 1 \Rightarrow 0 \le 1 \le 1 \rightarrow T$

$1 \le i \le n$

$1 \le 1 \le 7$ $\quad 1 \le 5 \le 7$

$1 \le 2 \le 7$ $\quad 1 \le 6 \le 7$

$1 \le 3 \le 7$ $\quad 1 \le 7 \le 7$

$1 \le 4 \le 7$

Case 3:- Profit per Unit Weight $x_i = \frac{P_i}{w_i}$

$X_1 = \frac{P_1}{w_1} = \frac{10}{2} = 5$

$X_2 = \frac{P_2}{w_2} = \frac{5}{3} = 1.7$

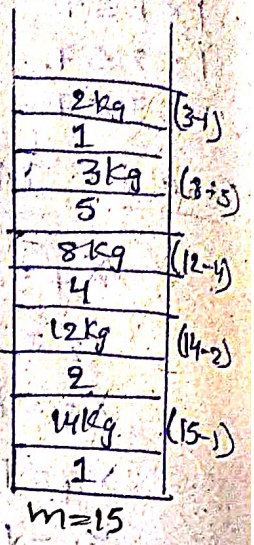
$X_3 = \frac{P_3}{w_3} = \frac{15}{5} = 3$

$X_4 = \frac{P_4}{w_4} = \frac{7}{7} = 1$

$X_5 = \frac{P_5}{w_5} = \frac{6}{1} = 6$

$X_6 = \frac{P_6}{w_6} = \frac{18}{4} = 4.5$

$X_7 = \frac{P_7}{w_7} = \frac{3}{1} = 3$



$X_5 = 1$ $X_3 = 1$ $X_2 = \frac{2}{3}$

$X_1 = 1$ $X_7 = 1$ $X_4 = 0$

$X_6 = 1$

1. $\sum_{0 \le i \le n} P_i x_i = P_1x_1 + P_2x_2 + P_3x_3 + P_4x_4 + P_5x_5 + P_6x_6 + P_7x_7$

$= 10 \times 1 + 5 \times \frac{2}{3} + 15 \times 1 + 7 \times 0 + 6 \times 1 + 18 \times 1 + 3 \times 1 =$

$= 55.3$

2. $\sum_{0 \le i \le n} w_i x_i = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + w_7x_7$

$= 2 \times 1 + 3 \times \frac{2}{3} + 5 \times 1 + 7 \times 0 + 1 \times 1 + 4 \times 1 + 1 \times 1$

$= 15$

3. $0 \le x_i \le 1$ $\quad 1 \le i \le n$

$0 \le x_1 \le 1 \Rightarrow 0 \le 1 \le 1$ $\quad 0 \le (x_5 = 1) \le 1$ $\quad 1 \le 1 \le 7$ $\quad 1 \le 5 \le 7$

$0 \le x_2 \le 1 \Rightarrow 0 \le \frac{2}{3} \le 1$ $\quad 0 \le (x_6 = 1) \le 1$ $\quad 1 \le 2 \le 7$ $\quad 1 \le 6 \le 7$

$0 \le x_3 \le 1 \Rightarrow 0 \le 1 \le 1$ $\quad 0 \le (x_7 = 1) \le 1$ $\quad 1 \le 3 \le 7$ $\quad 1 \le 7 \le 7$

$0 \le x_4 \le 1 \Rightarrow 0 \le 0 \le 1$ $\quad 1 \le 4 \le 7$

Total Possible Solution

Case	$\sum P_i x_i$	$\sum w_i x_i$
1	47	15
2	24	15
3	55.3	15

1, 2, 3 are called feasible solutions

Case-3 is called optimal solution.

Algorithm Knapsack

```
1. Algorithm GreedyKnapsack(m,n)
2. //p[1:n] and w[1:n] contains profits and weights
   respectively.
3. // n is the total number of objects such that
     P[i]/w[i] ≥ P[i+1]/w[i+1]
4. //m is the knapsack capacity and x[1:n] is the
   solution vector
5. {
6. for i=1 to n do x[i]=0.0
7. U=m
8. for i=1 to n do
9. {
10. if(w[i]>U) then break
11. x[i]=1.0, U=U-w[i]
12. }
13. if(i≤n) then x[i]=U/w[i]
14. }
```

Tracing

P_i	25	24	15
w_i	8	15	10

m=20

$P_1/w_1 = 1.4 \rightarrow X_3$

$P_2/w_2 = 1.6 \rightarrow X_1$

$P_3/w_3 = 1.5 \rightarrow X_2$

	w_i	P_i
$X_1 \rightarrow$	15	24
$X_2 \rightarrow$	10	25
$X_3 \rightarrow$	8	25

6. for i = 1 to 3 do
   x[1] = 0.0
   x[2] = 0.0
   x[3] = 0.0
7. U = 20
8. for i = 1 to 3 do
10. If (w[1] > 20)
        15 > 20 → F
11. x[1] = 1.0 , U = 20 - 15 = 5

Again

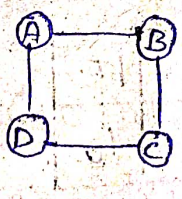
8. for i = 2 to 3 do
10. if (w[2] > 5)
        10 > 5 → T
13. if (2 ≤ 3) => x[2] = 5/10 = 1/2
14. x[3] = 0

(3) Minimum Cost Spanning Tree

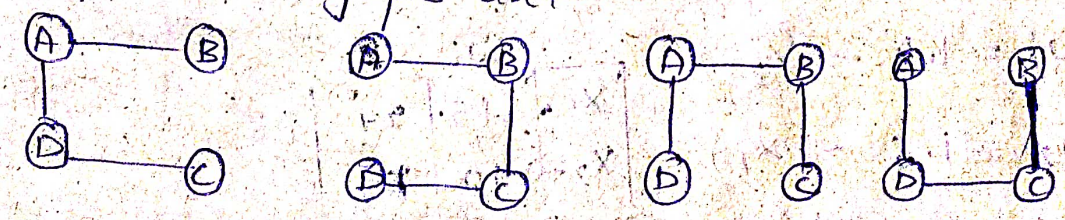
Let G = (V, E) is a graph a sub-graph of G = (V', E') is a Spanning tree.

Definition: A spanning Tree is a tree which doesn't Contains loops.

Ex: Consider the graph



The possible sub-graphs are:



Applications

#Spanning trees are used to obtain independent set of equations.

#Spanning trees are used in routing algorithms.

#Spanning trees are used in Network design.

#An edge of the spanning tree is called Branch and an edge in the graph that is not in the spanning tree is called chrod.

⇒It spans the graph and includes every vertex of the graph.

⇒The total Weight of all the edges should be minimum then it is called as Minimum Cost Spanning tree.

There are two Algorithms to implement Minimum Cost Spanning tree:-

(1) Prim's Algorithm

(2) Kruskal's Algorithm

(1) Prim's Algorithm

step 1: Start with first Vertex

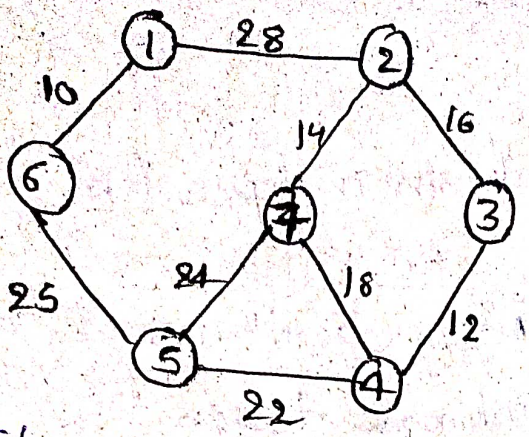
step 2: Select Next adjacent Vertex whose cost is minimum.

step 3: Repeat this process Until all the vertices should be visited.

step 4: If any edge form the cycle discord that edge

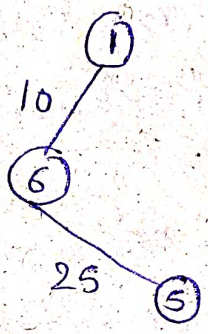
step 5: Calculate the total Weight of all the edges.

Ex: Find the Cost of Spanning tree Using Prim's Algorithm.

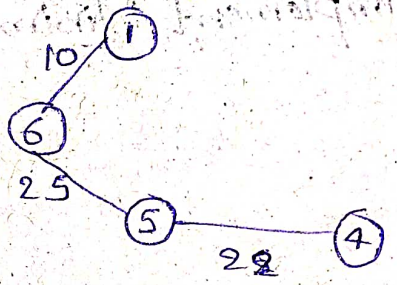


Step 1: Select 1-6

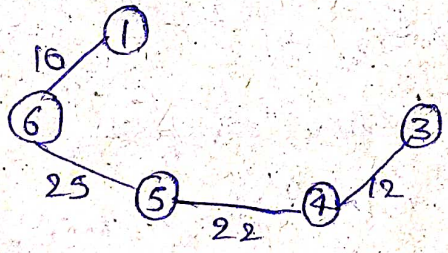
Step 2: Select 6-5



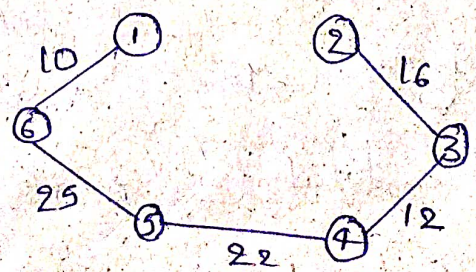
Step 3: 7 & 4 are adjacent of 5 then select 5-4 (cost 22)



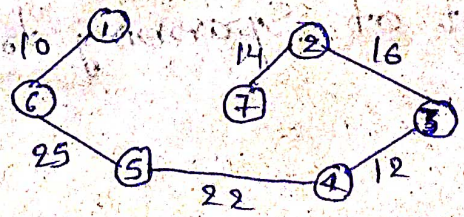
Step 4: 7 & 3 are adjacent for 4 then select 4-3 (cost 12)



Step 5:- Select 2



Step 6:- Select 7



Total Cost = 10+25+22+12+16+14 = 99 units

(2) Kruskal's Algorithm :-

step 1: Arrange all the edges according to the weight or cost.

step 2: Select edge whose cost is minimum

step 3: Repeat these process for all the edges.

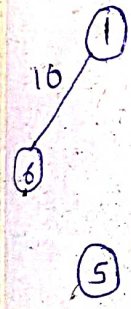
step 4: If any edge form the cycle discard that edge.

step 5: Calculate the total cost of spanning tree

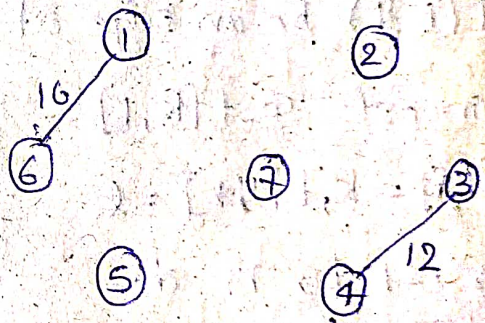
Ex:-

Edge	1-6	3-4	2-7	2-3	4-7	4-5	5-7	5-6	1-2
Cost	10	12	14	16	18	22	24	25	28

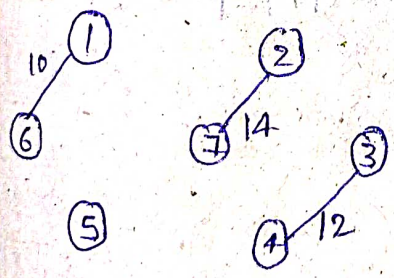
step 1:



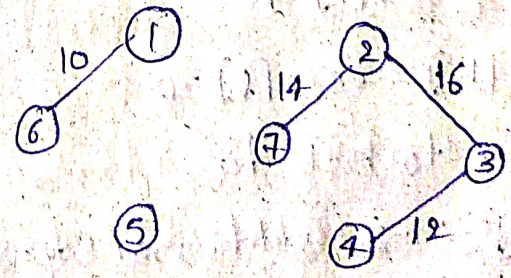
step 2:-



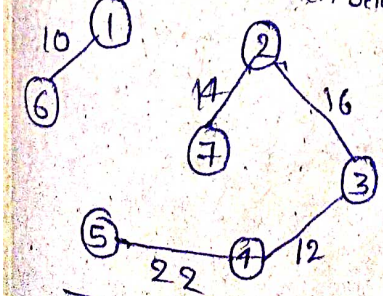
step 3:-



step 4:-

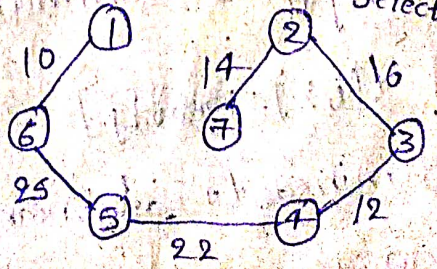


Step 5:-(4-7) → forms cycle. then select (4-5)



Step 6:-(5-7) → forms cycle then select (5-6)

(1-2) → forms cycle



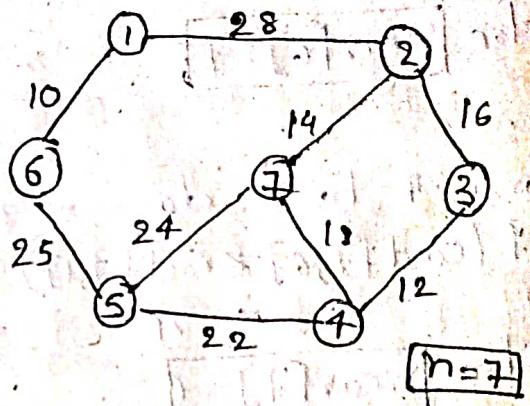
Total Cost = 10+12+14+16+22+25 = 99 units

Algorithm Prim's

```
1. Algorithm Prim(G, cost, n, t)
2. // E is the set of Edges in G
3. // cost[1:n, 1:n] is the adjacency matrix of n
   vertex such that cos[i:j]
4. // cos[i:j] is either positive Real number or infinity
   if no edge (i,j) exits
5. // A Minimum Spanning tree is computed
6. // All the values are stored in an array t[1:n-1, 1:2]
   t[i,1] t[1,2] is an edge that returns the minimum
7. // Cost spanning tree and the final cost is return.
8. {
9. Let (k,l) be an edge of minimum cost in E
10. mincost = cost[k,1]
11. t[1,1] = k, t[1,2] = l
12. for i=1 to n do
13. if (cost[i,l] < cost[i,k]) then near[i]=l
14. else near[i]=k
15. near[k] = near[l] = 0
16. for i=2 to n-1 do
17. // Find n-2 additional edges for t
18. Let j be an index such that near[j] ≠ 0 and
19. cost[j, near[j]] is minimum
20. t[i,1] = j  t[i,2] = near[j]
21. mincost = mincost + cost[j, near[j]]
22. near[j] = 0
```

23. for k=1 to n do
24. if ((near[k]≠0) and cost[k,near[k]]) > cost[k,j])
25. then near[k]=j
26. }
27. return mincost
28. }

Tracing



9. (1,6) is the edge with min cost
10. mincost = cost[1,6] = 10
11. t[1,1] = 1, t[1,2] = 6
12. for i=1 to 7 do
13. if (cost[i,1] < cost[i,k]) ⇒ (cost[1,6] < cost[1,1])
then near[i=1] = 6 ⇒ near[1] = 6
10 < ∞ → T

Again
12. for i=2 to 7 do
13. if (cost[2,6] < cost[2,1])
∞ < 28 → F
14. else near[i=2] = 1

Again
12. for i=3 to 7 do
13. if (cost[3,6] < cost[3,1])
∞ < ∞ → F
14. else near[3] = 1

Again
12. for i=4 to 7 do
13. if (cost[4,6] < cost[4,1])
∞ < ∞ → F
14. else near[4] = 1

Again

12. for i = 5 to 7 do
13. if (cost[5, 6] < cost[5, 1]) then near[5] = 6
    25 < ∞ → T

Again

12. for i = 6 to 7 do
13. if (cost[6, 6] < cost[6, 1]
    ∞ < 10 → F
14. else near[6] = 1

Again

12. for i = 7 to 7 do
13. if (cost[7, 6] < cost[7, 1])
    ∞ < ∞ → F
14. else near[7] = 1

15. near[1] = near[6] = 0
16. for i = 2 to 6 (n-1 = (7-1)) do
17. // find n-2 edges
18. Let j be an index such that near[j] ≠ 0
19. Cost[2, near[2]] ⇒ Cost[2, 1] = 28
    Cost[3, near[3]] ⇒ Cost[3, 1] = ∞
    Cost[4, near[4]] ⇒ Cost[4, 1] = ∞
    Cost[5, near[5]] ⇒ Cost[5, 6] = 25
    ~~Cost[6, near[6]] ⇒ Cost[6, x] ≠ 10~~
    Cost[7, near[7]] ⇒ Cost[7, 1] = ∞

    j = 5

20. t[i, 1] = j 5 , t[i, 2] = near[j] = 6
21. mincost = 10 + Cost[j, near[j]] = 10 + 25 = 35
22. near[5] = 0
23. for k = 1 to n 7 do

24. if ([near[1] ≠ 0]
0 ≠ 0 → F

Again
23. for i = 2 to 7 do
24. if ([near[2] ≠ 0) and (cost[2,1] > cost[2,5])
1 ≠ 0 25 > ∞ → F

Again
23. for i = 3 to 7 do
24. if ([near[3] ≠ 0) and (cost[3,1] > cost[3,5])
1 ≠ 0 ∞ > ∞ → F

Again
23. for i = 4 to 7 do
24. if ([near[4] ≠ 0) and (cost[4,1] > cost[4,5])
1 ≠ 0 ∞ > 22 → T
25. then near[4] = 5

Again
23. for i = 5 to 7 do
24. if ([near[5] ≠ 0) and (cost[5,1] > cost[5,5])
6 ≠ 0 ∞ > ∞ → F

Again
23. for i = 6 to 7 do
24. if ([near[6] ≠ 0)
0 ≠ 0 → F

Again
23. for i = 7 to 7 do
24. if ([near[7] ≠ 0) and (cost[7,1] > cost[7,5])
1 ≠ 0 ∞ > 24 → T
25. then near[7] = 5

Updated values
near[1] = 0, near[2] = 1, near[3] = 1, near[4] = 5,
near[5] = 0, near[6] = 0, near[7] = 5

Algorithm Kruskal's

```
1. Algorithm Kruskal(E, cost, n, t)
2. // E is the set of Edges in G. G has 'N' vertices
3. // cost(u,v) is the cost of edge u,v
4. // t is the set of Edges in the minimum
   Cost spanning tree and the final cost is returned.
5. {
6.  // Construct a Heap out of the Edges Cost using
      heapify.
7.  for i=1 to n do parent[i] = -1
8.  // Each vertex is in different set
9.  i = 0  mincost = 0.0
10. While((i<n-1) and (heap NOT Empty)) do
11. {
12.   Delete a minimum Cost edge (u, v) from the heap
13.   Re-heapify using Adjust
14.   j = find(u), k = find(v).
15.   if j != k then
16.   {
17.     i = i+1
18.     t[i,1] = u, t[i,2] = v
19.     mincost = mincost + cost(u,v)
20.     Union(j, k)
21.   }
22. }
23. if(i != n-1) then write no spanning tree
24. else return mincost
25. }
```

Algorithm Find

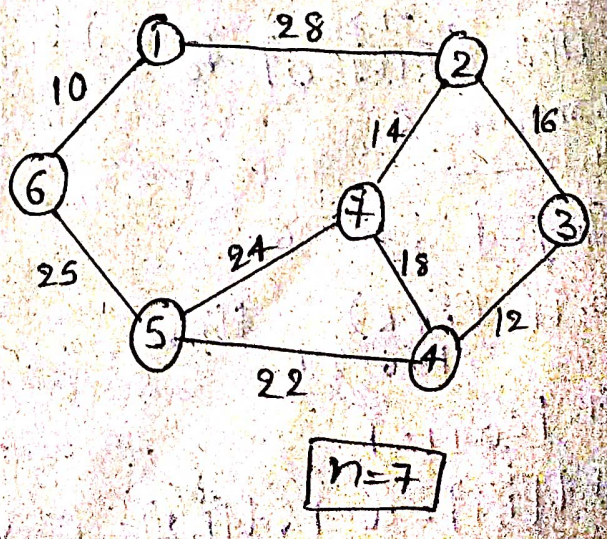
```
1. Algorithm Find(i)
2. {
3. while (P[i]>=0) do
4. return i;
5. }
```

Algorithm Union

```
1. Algorithm Union(i,j)
2. {
3. temp= P[i]+P[j]
4. if (P[i]>P[j])
5. {
6. P[i]=j
7. P[j]= temp
8. }
9. else
10. {
11. P[j]=i
12. P[i]=temp
13. }
```

Tracing

4, 5	22
4, 7	18
2, 3	16
2, 7	14
3, 4	12
1, 6	10 → Delete



7. for i=1 to 7 do
Parent[1] = -1
Parent[2] = -1
Parent[3] = -1
Parent[4] = -1
Parent[5] = -1
Parent[6] = -1
Parent[7] = -1

9. i=0, mincost=0.0

10. While (i<n-1 and (heap not empty)) do
$0<6 \rightarrow T$ (heap not empty: T)

11. {

12. Delete(u, v) → Delete(1, 6)

14. j = Find(u) , k = Find(v) → j = Find(1), k = Find(6)

---

1. Algorithm Find(i)
2. {
3. While (P[i] ≥ 0) → F (P[i] = -1 ≥ 0)
4. return i;

---

14. j = 1 [Find(1)] , k = Find(6)

---

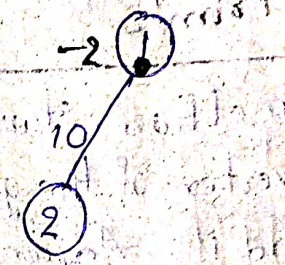
1. Algorithm Find(6)
2. {
3. While (P[6] ≥ 0) do (P[6] = -1 ≥ 0)
4. return 6;

---

14. j = 1 , k = 6
15. if (j ≠ k) then
$1 \neq 6 \rightarrow T$
17. i = 0+1 = 1
18. t[i,1] = u = 1 , t[i,2] = 6 = v
19. mincost = 0.0 + cost(1,6) = 0.0 + 10
= 10

2. Union (i, 6)

1. Algorithm Union(1, 6)
2. {
3. temp = p[i] + p[j] = p[1] + p[6]
   temp = -1 + (-1) = -2
4. if (p[1] > p[6])
   -1 > -1 → F
6. else
5. p[6] = 1 ; p[1] = -2 (temp)



23. if (i ≠ n+1) → no spanning tree.
    1 ≠ 6 → T

Time Complexity for Prim's

1. First for loop → for i = 1 to n repeat (n+1) times
2. for i = 2 to (n-1) repeat (n-1) times
3. for k = 1 to n repeat (n+1) times
4. So, the time complexity T(n) = (n+1)(n-1) + (n+1)

$$T(n) = n^2 - 1 + n + 1$$

$$T(n) = n^2 + n$$

$$\boxed{T(n) = O(n^2)}$$

Time Complexity for Kruskal's

- As the graph is connected we have $E \geq V - 1$ so, it can be observed that all the edges takes $O(E \log V)$ time. the for loop iterates 'E' number of times and each iteration involves a constant number of

accessing the vertices.
- Each access is performed $\theta(V)$ time so the total Running time of Algorithm is expressed as

$$T(n) = \theta(E \log V)$$

Prim's	Kruskal's
1. This method starts with single vertex of the graph and add its Adjacent vertices whose cost is minimum.	1. The Kruskal's algorithm also starts with each and every vertex of the graph and adds the smallest edge that joins the two trees to get the minimum cost spanning tree.
2. Prim's Considers all the vertices and it is suitable for few edges are present in the graph.	2. Kruskal's algorithm is suitable where all the edges and vertices taken into consideration.
3. Prim's algorithm is Considered as more space efficient. Since, Only one tree is taken and the edges that Connect to the vertices in tree are examine	3. Kruskal's algorithm is Considered as a time efficient. Because, Ordering of edges as per their weights is easy to traverse and traversing is done in faster manner.
4. Prim's algorithm is more time consuming.	4. Kruskal's algorithm is more space consuming

Single Source Shortest Path

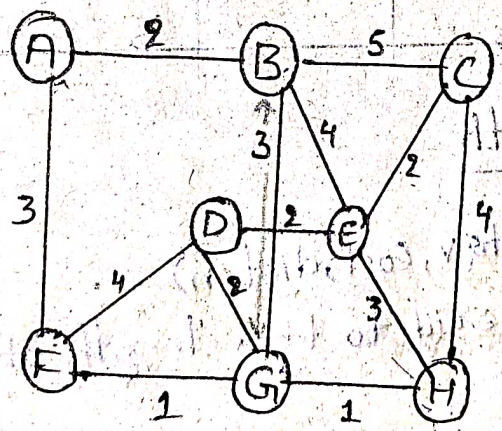
Generally, graph contains Vertices and Edges. Vertices Representing Cities Whereas, Edges Representing distance between two cities or time to reach the city.

→ The Length of the path is defined as sum of the weights of the edges on the path.

→ The starting vertex of the graph is called as source and the last vertex of the graph is called as destination.

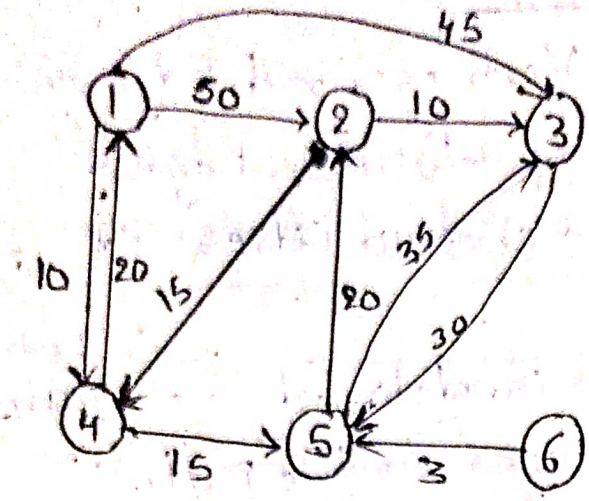
→ All Weights are positive

Ex: Find the shortest path from single source to all the vertices in the graph.



Source	Destination	Path	Length	No. of edges
A	B	[A, B]	2	1
A	F	[A, F]	3	1
A	G	[A, F, G]	4	2
A	D	[A, F, G, D]	6	3
A	H	[A, F, G, H]	5	3
A	E	[A, B, E]	6	2
A	C	[A, B, C]	7	2

(2) Find the shortest for directed graph



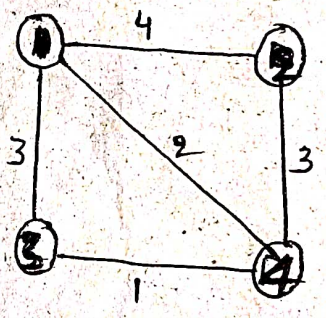
Source	Destination	Path	Length	No. of edges
1	2	[1,4,5,2]	45	3
1	4	[1,4]	10	1
1	3	[1,3]	45	1
1	5	[1,4,5]	25	2
1	6			

Algorithm Shortest Path

```
1. Algorithm ShortestPath(v, cost, dist, n)
2. //dist[j], 1≤j≤n is said to the length of the
   shortest path.
3. //v is set of vertices in the given graph
4. //dist[V] is said to "zero"
5. //cost[1:n, 1:n] is the adjacency matrix of graph.
6. {
7. for i=1 to n do
8. //initialize S
9. S[i]=false, dist[i]=cost[v,i]
10. }
11. S[v]=true, dist[v]=0.0
```

12. for num=2 to n-1 do
13. {
14. // determine (n-1) paths from V
15. // choose u from among these vertices
16. Not in S i.e, dist[u] is minimum
17. S[u] = true
18. for (each w adjacent to u with S[w]=false) do
19. // Update distance
20. if (dist[w] > dist[u] + cost[u,w]) then
21. dist[w] = dist[u] + cost[u,w]
22. }
23. }

Tracing.



Source vertex (V) = 1

n = 4 (Number of vertices)

dist[1] = 0

7. for i=1 to 4 do

9. S[1] = false, dist[1] = cost[1,1] = 0

S[2] = false, dist[2] = cost[1,2] = 4

S[3] = false, dist[3] = cost[1,3] = 3

S[4] = false, dist[4] = cost[1,4] = 2.

10. }

11. S[1] = true, dist[1] = 0.0

12. for num=2 to n-1 = 3 do

15. u=4 because dist[4]=2 is minimum

17. S[4] = true

18. for (each w adjacent to u then with s[w] = false) do

S[2] = s[3] = false

20. if (dist[2] > dist[4] + cost[4, 2])

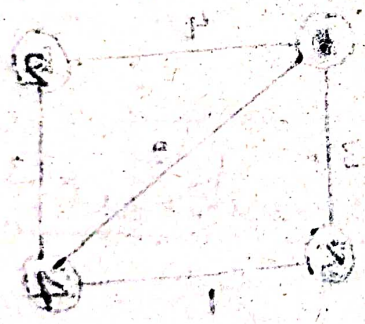
4 > 2 + 3

4 > 5 → F

if (dist[3] > dist[4] + cost[4, 3])

3 > 2 + 1

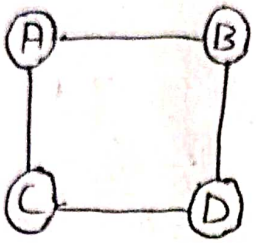
3 > 3 → F



UNIT III GRAPHS

Introduction

- Graph is a collection of vertices and edges.
- In other terms, $G = (V, E)$ where,
  $V \rightarrow$ Set of Vertices, $E \rightarrow$ Set of Edges

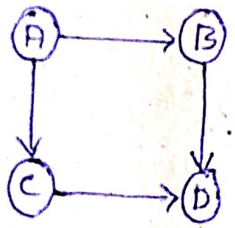


Here, A, B, C, D are called Vertices

AB, AC, CD, BD are called Edges

Directed Graph:- There is an arrow mark on the edges is called as directed graph.

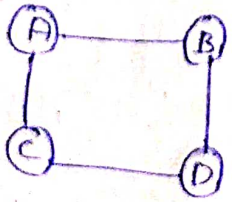
Ex:



Undirected Graph:- There is no arrow mark on the edges is called as Undirected Graph.

→ In an Undirected Graph traversing is possible in both directions.

Ex:



Weighted Graph:- Weights are associated on the edges is called as Weighted Graph.

Ex: