Unit 2. Divide

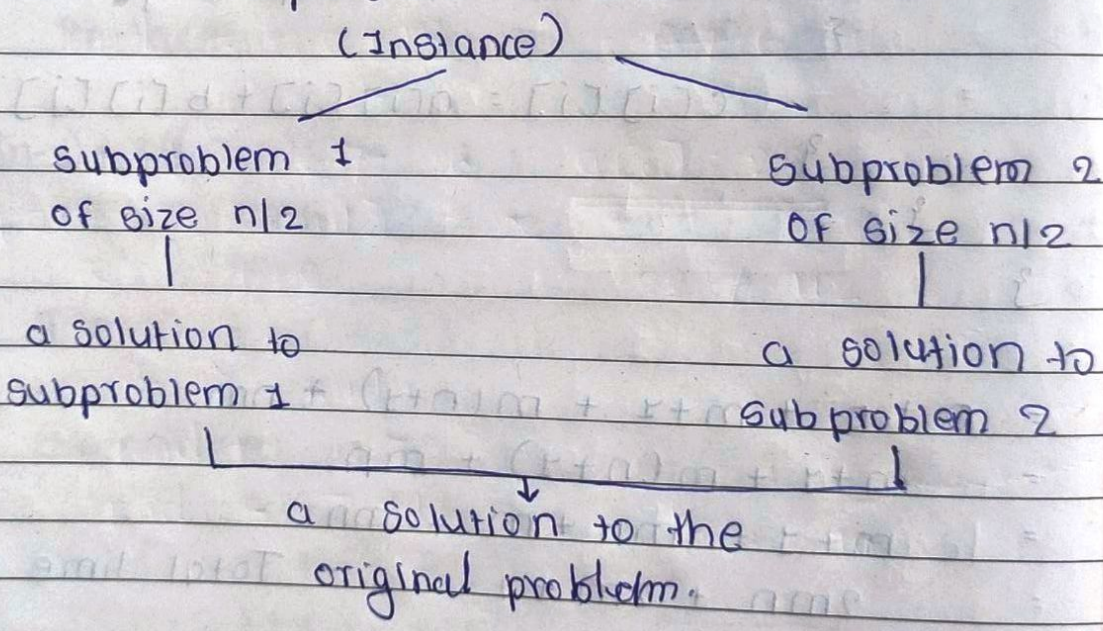
The Design and Analysis of Algorithm.

Divide & Conquare.

Whenever there is largest array then use.

The most well know algorithm design strategy

1. Divide the instance of problem into two or more smaller.



Problems:-

1) Sorting: mergesort & quicksort.
2) Binary tree travels.
3) Binary search (?)
4) Multiplication of large integers
5) Matrix multiplication
6) Closet-pair & convex-hull algorithms

```
1   Algorithm DAndC(e)
2   {
3       if Small(P) then return S(P);
4       else
5       {
6           divide P into smaller instance
7
8
9
10
```

Binary Search Algorithm
Reduce the time

```
1. Algorithm BinSrch(a, i, l, x)
2. // Give an array a[i : l] of elements in
       nondecreasing.
3. // oreder, 1 ≤ i ≤ l, determine whether x is is
       present, and.
4. // if so. return j such that x = a[j]; else
           return 0.
5.  {
6.      if (l = i) then // If Small(P)
7.      { else return 0;
8.          if (x = a[i]) then return i;
9.          else return 0;
10.     }
11.     else
12.     { // reduce P into smaller subproblem.
13.         mid := ⌊(i + l) / 2⌋;
14.         if (x = a[mid]) then return mid;
```

```
15.     else if(x<a[mid]) then
16.             return BinSrch(a,i,mid-1,x);
17.     else return BinSrch(a, mid+1, l, x);
18.     }
19. }
```

Q.

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

1) calculate mid:-

$$mid = \frac{0+12}{2} = 6$$

mid = 6 a[6] = 23

2) compare a[mid] = x

x = 112 23 ≠ 112

3) x > mid

mid + 1 = 7

$$mid = \frac{7+14}{2} = \frac{21}{2} = 10$$

mid = x

112 = 112

```
//max and min to the largest & smallest values in
// a[i:j] respectively.
{
  if(i==j) then max:= min := a[i]; // small(p)
  else if (i=j-1) then // Another case of small(p)
       {
         if (a[i] < a[j]) then
         {
            max := = a[j]; min := a[i];
         }
         else
         {
            max := a[i]; min := a[j];
         }
       }
       else
       {
         // if P is not small, divide P into subproblem
         // find where to split the set.

            mid := ⌊(i+j)/2⌋;

         // solve the subproblems.
            MaxMin(i, mid, max, min);

         if (max < max1) then max := max1;
         if (min > min1) then min := min1;
       }
}
```

0 1 2 3 4

a[5] = { 5, 10, 15, 20, 25 } key 25

Step 1:- a[mid] = $\frac{0+4}{2}$ = 2

a[mid] = 15

Step 2:- a[mid] < key

15 < 25

Step 3:- 25 is right side of mid.

mid is increment by 1

mid = 2

mid + 1 = 2 + 1 = 3

a[3] = 20

Step 4:- 20 < key

a[mid] = $\frac{3+4}{2}$

= $\frac{7}{2}$ = 3 Floor

a[3] = 20

Step 5:- 25 is greater than 20

mid increment by 1

3 + 1 = 4

a[4] = 25

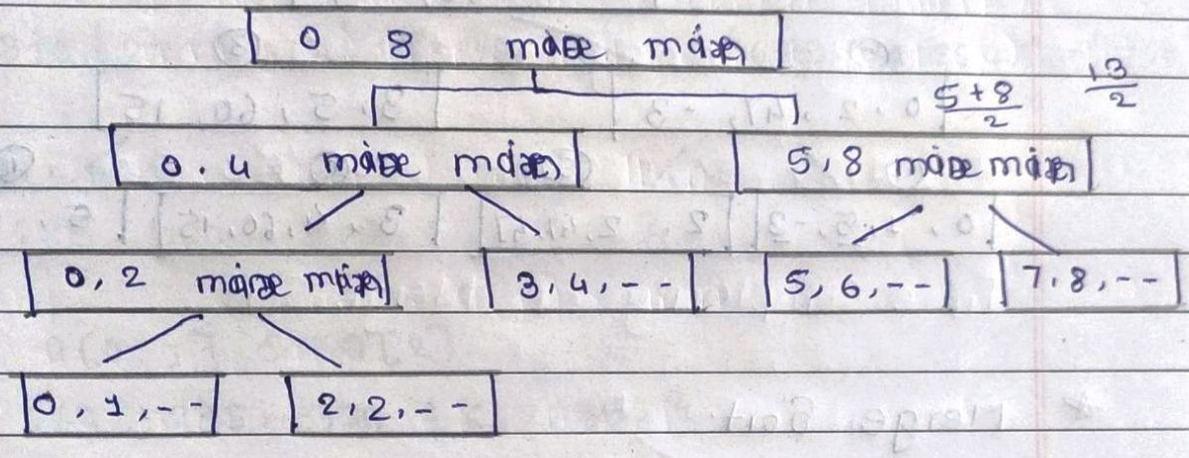
match is found.

Find maximum & Minimum

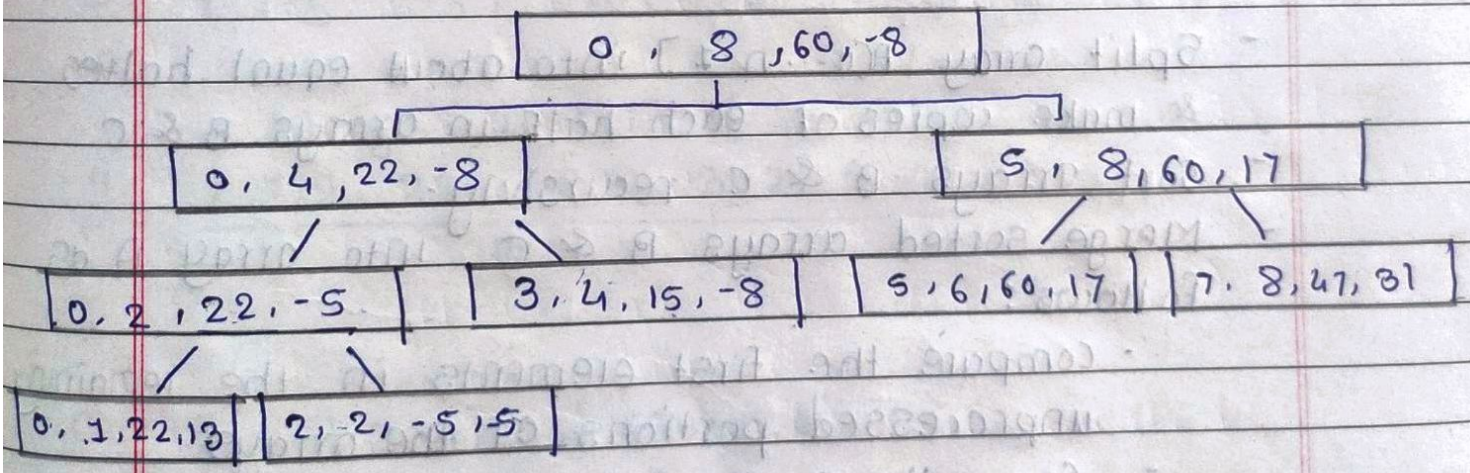
0	1	2	3	4	5	6	7	8
22	13	-5	-8	15	60	17	31	47

a[9] = { 22, 13, -5, -8, 15, 60, 17, 31, 47 }

mid = $\frac{i+1}{2}$ = $\frac{0+8}{2}$ = 4



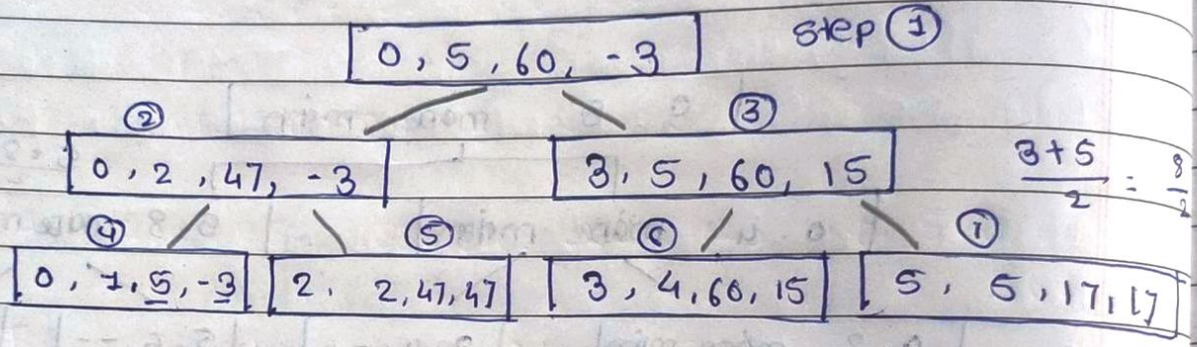
Bottom up approach



0 1 2 3 4 5

a[6] = {5, -3, 47, 15, 60, 17}

mid = $\frac{i+l}{2} = \frac{0+5}{2} = 2$



★ Merge Sort

1) Split array into equal halves

- Split array A[0...n-1] into about equal halves & make copies of each half in arrays B & C
- Sort arrays B & C recursively.
- Merge sorted arrays B & C into array A as follows:
  - Compare the first elements in the remaining unprocessed portions of the arrays
  - Copy the smaller of the two into A, while incrementing the index indicating the unprocessed.

Example :-

a[0:9] = { 310, 285, 179, 652, 351, 423, 861, 254, 450, 520 }

mid = $\frac{i+l}{2} = \frac{0+9}{2} = 4$

b[i] { 310 | 285 | 179 | 652 | 351 } c { 423, 861, 254, 450, 520 }

a[0] & a[1]

1 { 285, 310 | 179, 652, 351 }

a[0:1] & a[2]

2 { 179, 285, 310 | 652, 351 }

a[0:2] & a[3] & a[4]

3 { 179, 285, 310 | 351, 652 }

Merge a[0:2] & a[3:4]

4 { 179, 285, 310, 351, 652 }

C = { 423 | 861, | 254 | 450, 520 }

Merge a[5] & a[6]

5 { 423, 861 | 254 | 450 | 520 }

Merge a[5:6] & a[7]

6 { 254, 423, 861 | 450 | 520 }

Merge a[8] & a[9]

⑧ { 254, 423, 861 | 450, 520 }

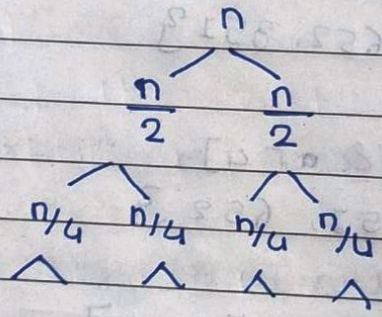
Merge a[5:7] & a[8:9]

⑨ C= { 254, 423, 450, 520, 861 }

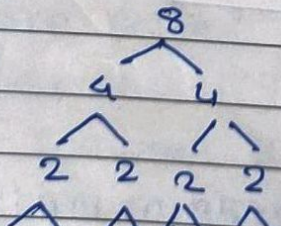
Marge array B & C

a= { 179, 254, 285, 310, 361, 423, 450, 520, 652, 861 }

Time Complexity of Merge Sort



Suppose n = 8

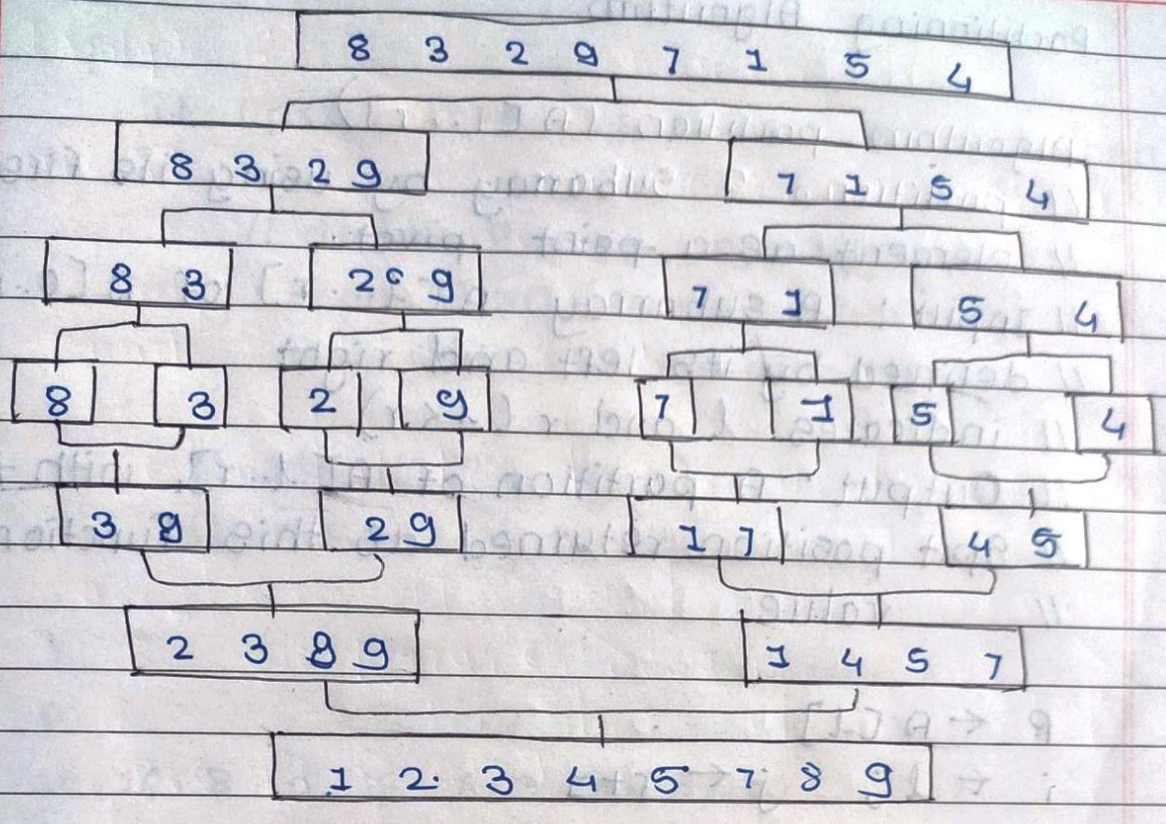


$= \log_2 2^8$

$= \log_2 n$

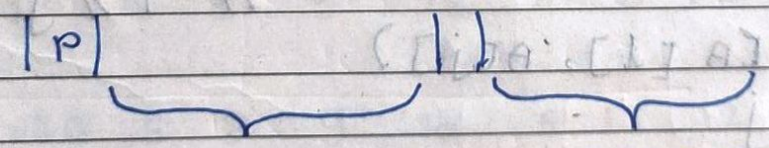
$= \log_2 n * n$

$= O(n \log n)$



★ Quick sort list divide depends upon pivot.

- Select pivot (partitioning element - here the first element
- Rearange the list to that



- left side is less than pivot element
- Right side element is greater than pivot ele.

Partitioning Algorithm

```
Algorithm partition (A[l..r])
// partitions a subarray by using its first
// element as a ~~point~~ pivot.
// Input : A subarray A[l..r] of A[0..n-1],
// defined by its left and right
// indicates l and r (l<r)
// Output : A partition of A[l..r], with the
// Spit position returned as this function's
//      value

p ← A[l]
i ← l; j ← r+1
repeat
    repeat i ← i+1 until A[i] >= p   or i>r
    repeat j ← j-1 until A[j] < p    or j=l
until i >= j
swap (A[i], A[j])  // undo last swap when
                   //  i >= j
swap [A[l], A[j])
return j
```

```
Algorithm QuickSort (p, q)
// Sort the elements a[p],..., a[q] which resides in
     the global
// array a[1:n] into asending order ; a[n+1] is
   considered to be defined & must be >= all the
   elements in a[1:n]
```

```
{
    if (p<q) then  // If there are more than 1 ele.
    {
        // divide p into two subproblems.
        j := partition (a, p, q, q+1);
            // j is the subproblem

        QuickSort (p, j-1);
        QuickSort (j+1, q);
```

}

p

9.*) 10, 16, 8, 12, 15, 6, 3, 9, 5, ∞
i j

Pivot = 10

10 16 8 12 15 6 3 9 5
i j

10 5 8 12 15 6 3 9 16
+ + i j

10 5 8 9 15 6 3 12 16
i i

10 5 8 9 3 6 15 12 16
ij

10 5 8 9 3 6 15 12 16
P

key = a[mid]
recur [q, mid-1] *)

6 5 8 9 3 10 15 12 16

6 5 8 9 3
P i i j

6 8 8 9 5

6 5 3 9 8
i

[6] 5 8 9 3

Pivot = 6

6 5 8 9 3
P i j

6 5 8 9 3
P i j

6 5 3 9 8
P i j

3 5 6 9 8

9 8
P i j

8 9

15 12 16
P i j

15 12 16
P j i

Replace pivot with j

12 15 16

Quick Sort

1	2	3	4	5	6	7
7	3	10	12	11	6	4
p						

Adding ∞ at last of list

1	2	3	4	5	6	7	8
7	3	10	12	11	5	4	∞
p, i							j

Pivot = 7

i + 1 = 2 a[2] = 3

3 < 7

i + 1 = 3 a[3] = 10

10 > 7

j - 1 =

7	3	4	12	11	5	10
p	i		i	j	j	

7 3 4 11 12 5 10

7	3	4	5	11	12	10
			ij			

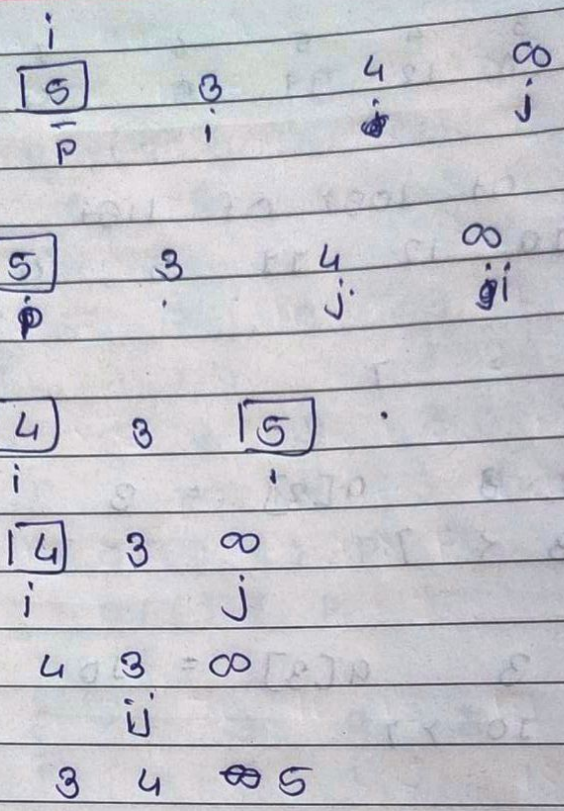
5 3 4 7 11 12 10

left side

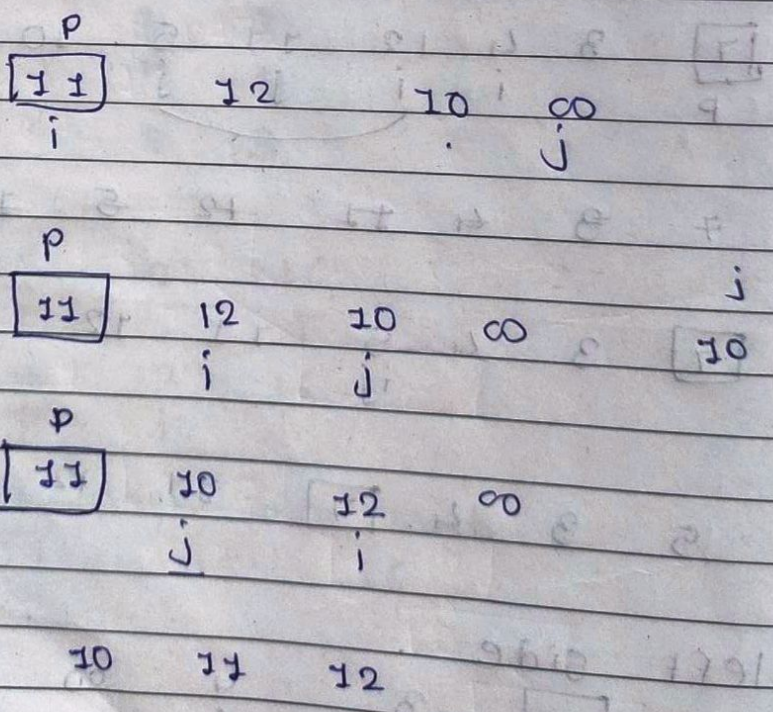
5	3	4	∞
p	i	j	~~j~~

5	3	4	∞
	j	i	j

∞ 3 4 5



Right side

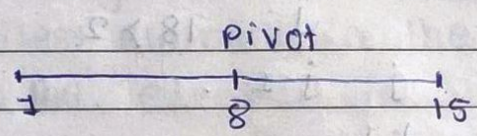


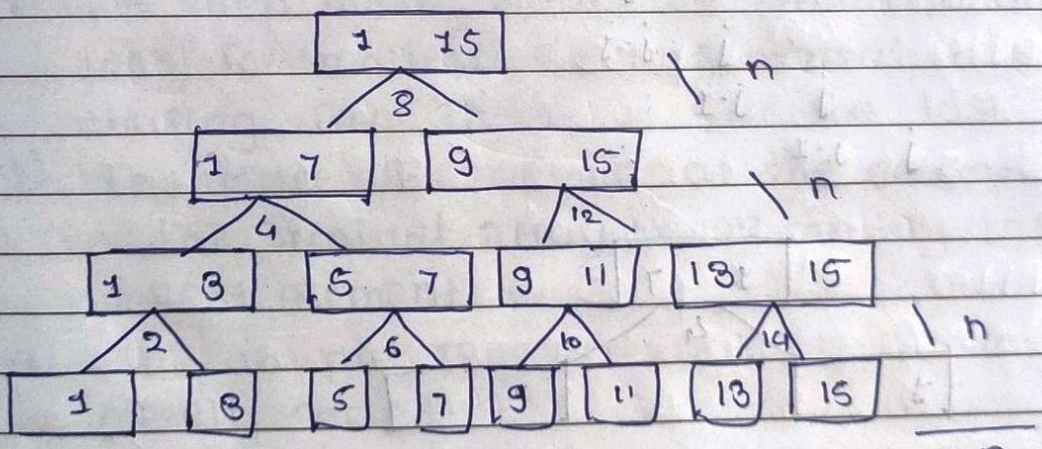
Sorted list 3 4 5 7 10 11 12

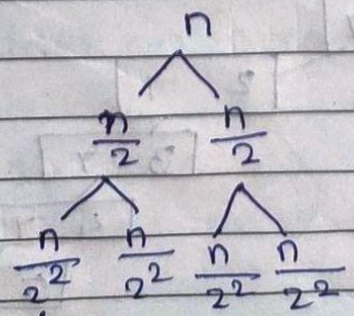
Analysis Of Quick Sort.

```
Algorithm QuickSort (l, h)
{
    if (l < h)
    {
        j = partition (l, h);
        QuickSort (l, j-1);
        QuickSort (j+1, h);
    }
}
```

List from 1 to 15







$\frac{n}{2^k} = 1$

$n = 2^k$ $= n \log_2 n$

$\log n * n$

$O(n \log n)$

best complexity.

occure when list is already in sorted form.

★ Worst case Time complexity of Quick sort

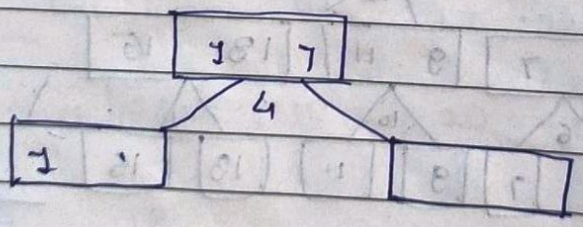
[2] 4, 8, 10, 16, 18, 19, ∞
P j

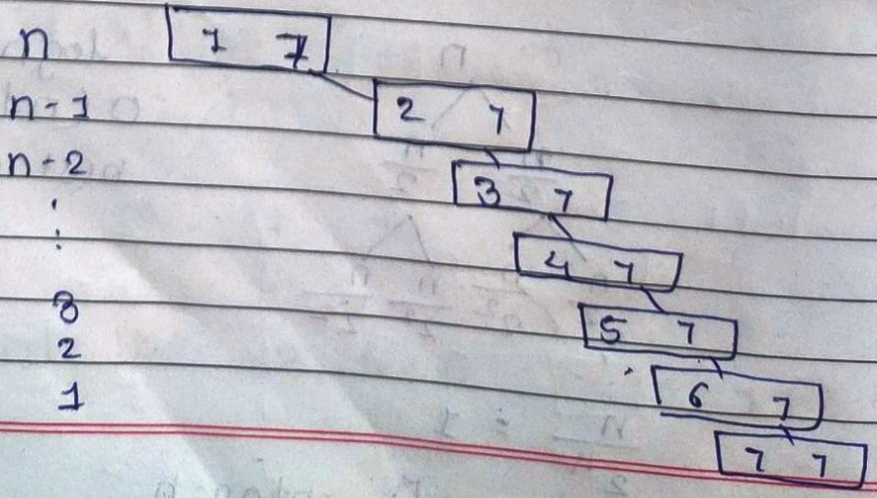
[2] 4 8 10 16 18 19 ∞
P i j

4 > 2 take i as it is check j is less than pivot.

19 > 2 j > P
j - 1

[2] 4 8 10 16 18 19 ∞
j 18 > 2
j j-1
j j-1
j j-1
j j-1
j j-1





$$1 + 2 + 3 \ldots n$$
$$= \frac{n(n+1)}{2}$$
$$= \frac{n^2 + n}{2}$$
$$= n^2$$
$$= O(n^2)$$

Selection Sort Algorithm :-

1) Starting from the first element we search the smallest element in the array and replace it with the element in the first position.
2) We then move on to the second position and look for smallest element present in the subarray starting from index 1, till the last index.
3) We replace the element at the second position in the original array or we can say the second smallest element
4) This is pre repeated, until the array is completely sorted.

Let's consider an array with values:

{ 3 6, 1, 8, 4, 5 }

Below we have a pictorial

original array

3 6 1 8 4 5

After 1st pass

1 6 3 8 4 5

After 2nd pass

1 3 6 8 4 5

After 3rd pass

1 3 4 8 6 5

After 4th pass

1 3 4 6 8 5

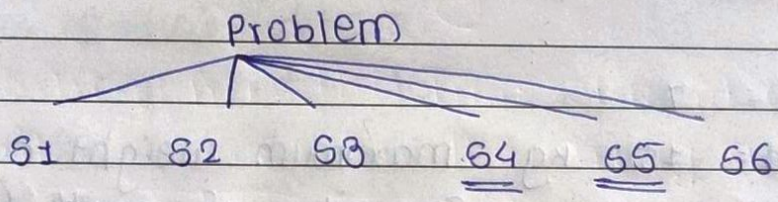
After 5th pass

1 3 4

Gready Method :- Use to create & Design algorithm.

- Based on principle of optimization.
- problem demands result in either minimum or maximum form.
- Optimal soln should be one 6.
- Optimal soln from feasible solution.

P1- A -> B



Feasible Solution - S4, S5.
Optimal solution -> S4

Gready method is based on principle of optimization, this method is used to solve minimization and maximization problem.

Optimization -

A problem which demands a solution either in maximum and minimum is called as optimization.

Feasible Solution- Contraints

Solution with satisfies the ~~constant~~ of the given problem are knows feasible soln.

Optimal Soln

Optimal solution it should be one from feasible solution which satisfy the objective of the problem.

Knapsack Problem :-

Object	0	1	2	3	4	5	6	7
profits	10	10	3	15	7	6	18	3
Weight	W	2	3	5	7	1	4	1

n = 7

m = 15 kg maximum weight carring Capacity of sack. (m)

$x = \frac{P}{W}$

15 kg

x =	(5	10	3	1	6	4.5	3)
	x_1	x_2	x_3	x_4	x_5	x_6	x_7
	1	2/3	1	0	1	1	1

x = (1 2/3 1 0 1 1 1)

m = 16 kg

15 - 1 = 14
14 - 2 = 12
12 - 4 = 8
8 - 5 = 3
3 - 1 = 2
2 - 2 = 0

Date

$\Sigma x_i w_i$: $(1 \times 2) + (2/3 \times 3) + (1 \times 5) + 0 + (1 \times 1) + (1 \times 4) + (1 \times 1)$

$= 2 + 2 + 5 + 1 + 4 + 1$

$= 15$

$\Sigma x_i p_i$: $(1 \times 10) + (2/3 \times 5) + (15 \times 1) + 0 + (6 \times 1) + (1 \times 18) + (3 \times 1)$

$= 10 + 32 + 15 + 6 + 18 + 3$

$= 54.6$

i) $\Sigma x_i w_i \leq 15$

i.e $15 = 15$

ii) Max $\Sigma p_i x_i$.

Example -

Object	O	1	2	3	4	5
Profit	P	15	7	8	5	3
Weight	W	2	3	4	2	1

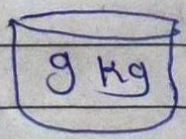
$n = 5$ $\quad m = 9$

(P/W) x : (7.5 $\quad$ 2.33 $\quad$ 2 $\quad$ 2.5 $\quad$ 3)

$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5$

1 $\quad$ 1 $\quad$ 1/4 $\quad$ 1 $\quad$ 1)

x - (1 $\quad$ 1 $\quad$ 1/4 $\quad$ 1 $\quad$ 1)



$9 - 2 = 7$ $\quad$ $6 - 2 = 4$

$7 - 1 = 6$ $\quad$ $4 - 3 = 1$

$\Sigma x_i w_i = (1 \times 2) + (1 \times 3) + (1/4 \times 8) + (2 \times 1) + (1 \times 1)$

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

$= 9$

$\Sigma x_i p_i = (1 \times 15) + (1 \times 7) + (1/4 \times 8) + (5 \times 1) + (2 \times 3)$

$= 15 + 7 + 2 + 5 + 3$

$= 32$

$\Sigma x_i w_i \leq 9$ and

Max $\Sigma x_i p_i$

9.3)

Object	O	1	2	3	4	5	6	7
Profits	P	13	8	9	5	3	4	2
Weight	W	2	4	3	3	2	1	2

$n = 7 \qquad m = 13$

$(P/W) = (6.5 \quad 2 \quad 3 \quad 1.6 \quad 1.5 \quad 4 \quad 1)$

$x \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7$

$1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0$

13 Kg $\quad 13 - 2 = 11 \quad 11 - 1 = 10 \quad 10 - 3 = 7$

$7 - 4 = 3 \quad 3 - 3 = 0$

$x = (1 \; 1 \; 1 \; 1 \; 0 \; 1 \; 0)$

$\Sigma x_i w_i = (1\times2) + (1\times4) + (1\times3) + (1\times3) + 0 + (1\times1) + 0 = 13$

$\Sigma x_i p_i = (1\times13) + (1\times8) + (1\times9) + (1\times5) + 0 + (1\times4) + 0$

$= 13 + 8 + 9 + 5 + 4$

$= 39.$

Job Sequencing with deadline -

Ex $n = 5$ number of jobs.

Jobs $J = J_1\ J_2\ J_3\ J_4\ J_5$

Profit	$P =$ 20	15	10	5	1
Deadlines	2	2	1	3	3

(uniprocessor)

1) 1 machine. only 1 machine you given.
2) No premption. (1 unit of time) each job require
3) Max profit.

→

	J2	J1	J4	
0 —	1 —	2 —	3	Here we have 3
9	10	11	12	deadline

Profit J2 = 15 J1 = 20 J4 = 5

15 + 20 + 5 = 40

J2 → J1 → J4 =

Job considered	slot assigned	soln	Profit
—	—	—	0
J2	[0,1]	J2	15
J1	[0,1] [1,2]	J1, J2	10+15
J3	[0,1] [1,2]	J1, J2	20+15
J4	[0,1] [1,2] [2,3]	J1, J2, J4	20+15+5
J5	[0,1] [1,2] [2,3]	J1, J2, J4	40

Example :-

n = 7

Job	J	J1	J2	J3	J4	J5	J6	J7
Profit	P	35	30	25	20	15	12	5
deadline	d	3	4	4	2	3	1	2

Machine
9 am

0 —J4— 1 —J3— 2 —J1— 3 —J2— 4

J4 → J3 → J1 → J2

Profit

J4 = 20

J3 = 25

J2 = 30

J1 = 35

= 20 + 25 + 30 + 35

= 110

Job considered	Slot assigned	soln	Profit
J4	[0,1]	[J4	20
J3	[0,1] [1,2]	J3, J4	25+20
J1	[0,1] [1,2] [2,3]	J2, J3, J4	30+25+20
J2	[0,1] [1,2] [2,5] [3,4]	J1, J2, J3 J4	35+30+25+20

Example

n = 7

Job J	J1	J2	J3	J4	J5	J6	J7
P	3	5	20	18	01	6	30
d	1	3	4	3	2	1	2

Machine gan: 0 J6 1 J7 2 J4 3 J3 4

J6 -> J7 -> J4 -> J3

Profit

J6 = 6
J7 = 30
J4 = 18
J3 = 20

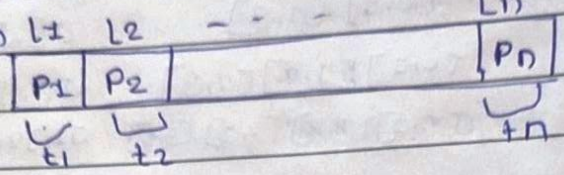
6 + 30 + 18 + 20 = 74

Job considered	Slot assigned	soln	Profit
J3	~~[0,1] [1,2] [2,3] [3,4]~~	~~J6 J7 J4, J3~~	20
J4	[2,3] [3,4]	J3, J4	20+
J7			

Optimal Storage of Tapes.

deadline store in tapes in sequential manner.

L - Length of Program
P - Program.



t = time require to retrive.

$P_2 = t_1 + t_2$

$P_n = t_1 + t_2 \dots t_n$

Total time $= \sum_{j=1}^{n} t_j$

Min retrival time

$$MRT = \frac{1}{n} \sum_{j=1}^{n} t_j = \frac{\text{Addition total time}}{n}$$

$d(I)$ $\boxed{I = P_1, P_2 \dots P_n}$

$d(I)$ ~~MRT~~ depends upon the data is represent in which sequance manner on tapes.

MRT depend upon $d(I)$

$d(I)$ · optimal ordering.

there are $n!$ way to store program in tapes

No. of program arranged on tapes : $n!$

★ Example.

We have 3 program P1 P2 P3

length of program 5 10 3

I	d(I) P1	P2	P3	MRT	
1 2 3	5	5+10	5+10+3	38/3	12.6
1 3 2	5	5+3	5+3+10	31/3	10.33
2 1 3	10	10+5	10+5+3	48/3	14.3
2 3 1	10	10+3	10+3+5	41/3	13.6
3 1 2	3	3+5	3+5+10	29/3	9.6
3 2 1	3	3+10	3+10+5	34/3	11.3

9.6	optimal MRT

3 1 2
3 5 10

Drawbacks of

1) For large value of n time will be very high to find (MRT) all possible permutations, for n number of program it is a time consuming process.

Solution -

Solution to this problem is a greedy method, in this case greedy method say's that if we arrange / store program in increasing order of there length then we will get optimal MRT within a very less amount of time. Minretrival time.

Example

n = 4

	P1	P2	P3	P4
length	12	5	8	4

	P1	P2	P3	P4
4*3*2	12	5	8	4

J	d(J)
1 2 3 4	12+12+5+12+5+8+12+5+8+4 : 83
1 3 2 4	12+
1 4 3 2	12+12+4+12+4+8+12+4+8+5 : 71
2 1 4 3	
2 3 4 1	5+5+12+5+12+4+5+12+4+8
2 4 1 3	5+5+8+5+8+4+5+8+4+12
3 1 2 4	
3 2 4 1	8+8+5+8+5+12+8+5+12+4
3 4 1 2	8+8+4+8+4+12+8+4+12+5
4 1 2 3	4+4+12+4+12+5+4+12+5+8
4 2 3 1	
4 3 2 1	4+4+8+4+8+5+4+8+5+12

n

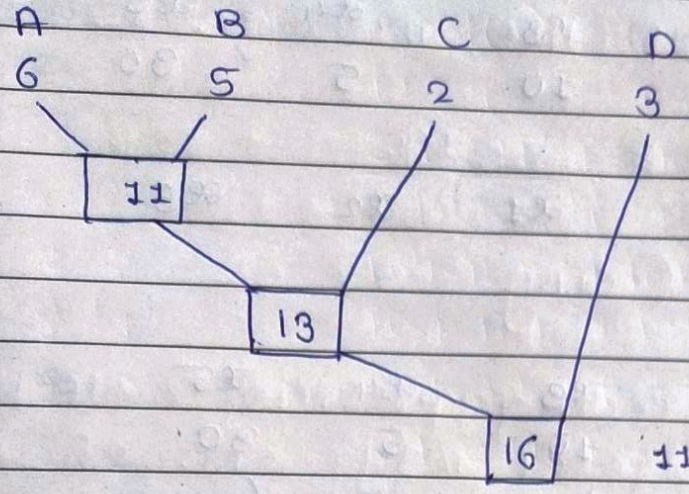
Optimal Merge Pattern :-

Increasing order

A	B	C	C
3	5	9	9
8	9	11	5
12	11	9	8
20	16		9
4	4		11
n	m		12
			16
			20

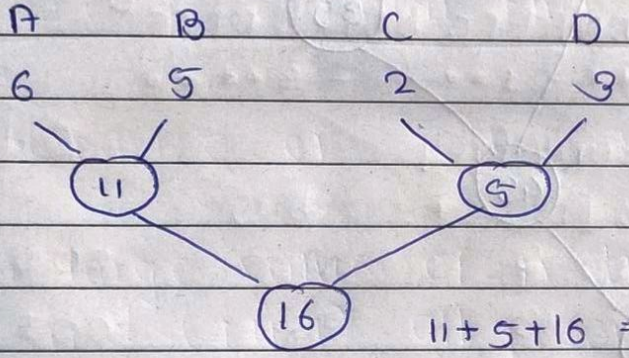
8 element

Total element → 8 n+m



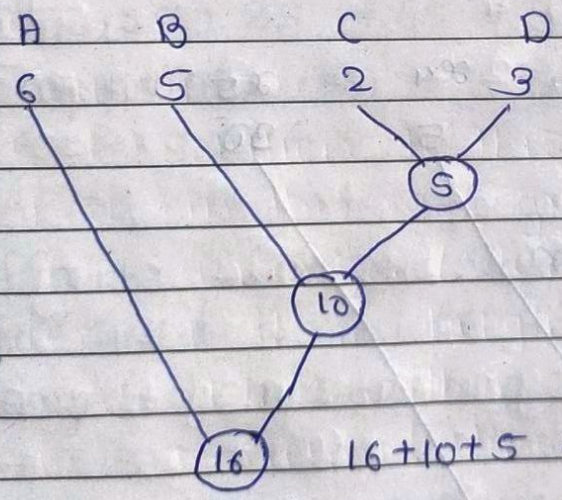
Merge patter ①

$11+13+16 = 40$



Merge Pattern ②

$11+5+16 = 32$



Merge Pattern ③

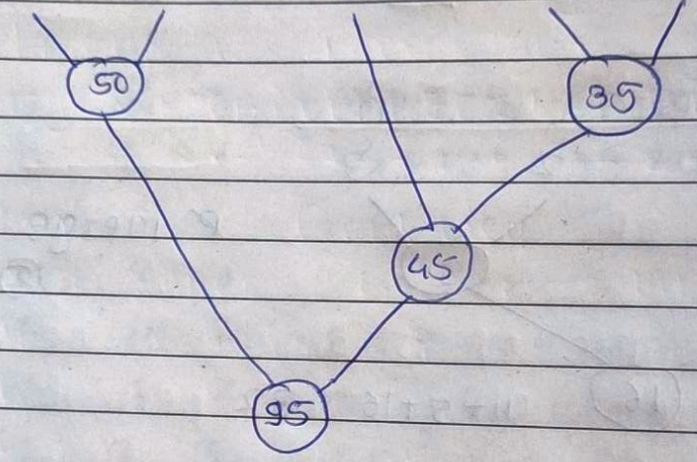
– Optimal Merge Pattern because we get smallest

$16+10+5 = 31$

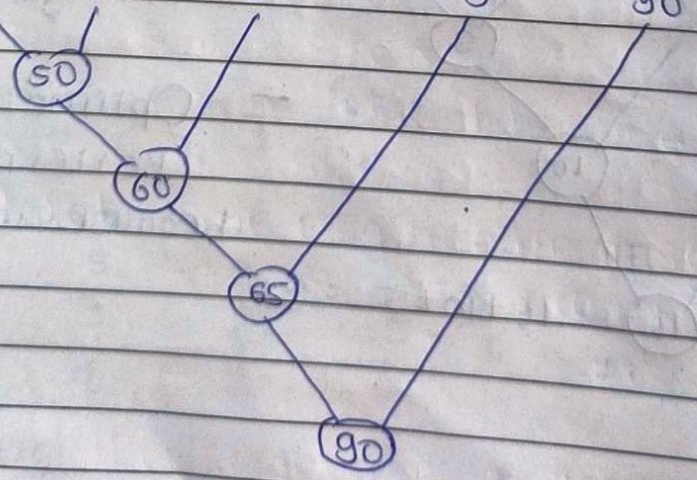
x_1	x_2	x_3	x_4	x_5
20	30	10	5	30

x_4	x_3	x_1	x_2	x_5

x_1	x_2	x_3	x_4	x_5
20	30	10	5	30



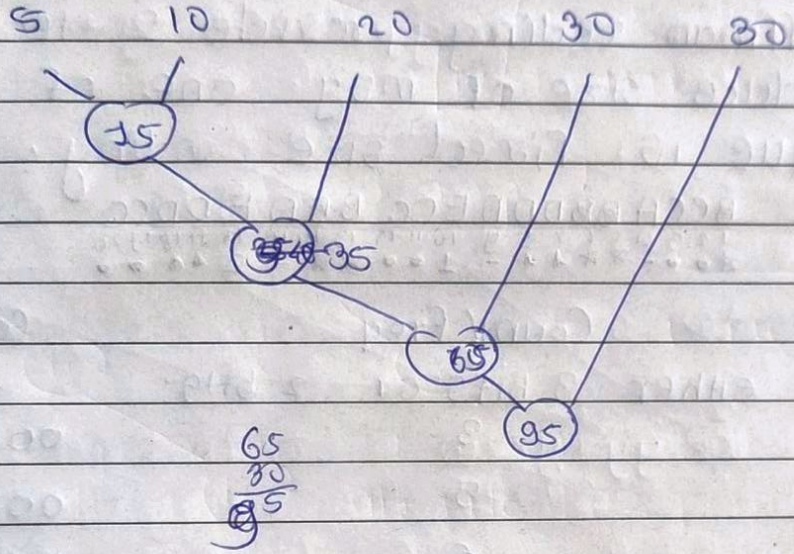
x_1	x_2	x_3	x_4	x_5
20	30	10	5	30



65
35
90

65

x_1	x_2	x_3	x_4	x_5
20	30	10	5	30



65
30
5

$15 + 35 + 65 + 95 = 205$

optimal cost = 205

* Huffman Coding It is a copression techinca is used to reduce the size of msg or data or info.

Message : BCCABBDDAECCBBAEDDCC

5 Alphabate are used repeatedly in msg.

	Ascii Value	Binary
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101

used to reduce cost of ~~m~~ data transmission.

Page No.
Date

Total size of the message is 160 bits.

$20 \times 8 = 160$

Fixed size Coding :-

Haffman coding provide 2 technique to reduce size of msg. one of this technique is fixed size coding.

Msg : BCCABBDDABECCBBBAEDDCC

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
★	●	●	-	★	★	▲	▲	-	1	●	●	★	★	-	2	▲	▲	●	●

Char	Count/ freq	Code	Size
A	3	000	20 × 3
B	5	001	60
C	6	010	
D	4	011	
E	2	100	

code is fixed either 3 bits or 2 bits.

$5 \times 8 = 40$

15 bits.

Size : 20×3

$= 60$ bits size carry this

total size msg : size msg + size table

$= 60 + 55 \leftarrow (40+15)$

$= 115$ bits.

When the msg use 1 letter then 1 ~~bit~~

4 — 2

8 — 3 bit binary value

Date

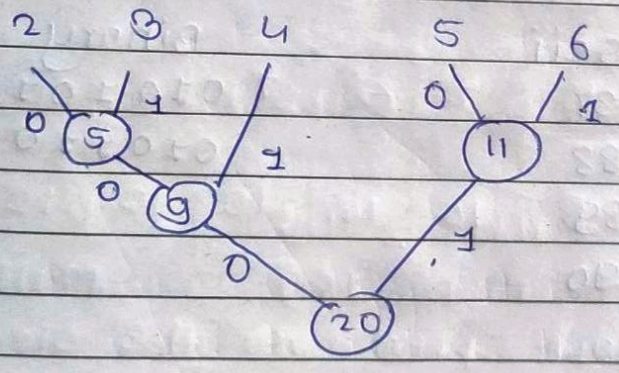
Variable size coding.

Message : BCCABBDDAECCBBAEDDCC

Char	Count
A	3
B	5
C	6
D	4
E	2

Arranging in increasing order.

2	3	4	5	6
E	A	D	B	C



Cost :-
5 + 9 + 11 + 20
= 45 bits -

Assume left side : 0
Right side : 1

Char	count	code	count code
A	3	100	3×3 = 9
B	5	01	5×2 = 10
C	6	11	6×2 = 12
D	4	10	4×2 = 8
E	2	000	2×3 = 6
			45 bits

① Cost size of Msg = 45 bits

② Size of table : size char + size code
= 40 + 12
= 57 52

③ Total size cost msg : size msg + size table
= 45 + 52
= 97

Messs
xzzyyxyzzw
1 2 3 4 5 6 7 8 9 10

Char	Ascii	Binary
w	87	01010111
x	88	01011000
y	89	
z	90	

Total size of the message is
10 x 8 = 80

Fixed Size coding.

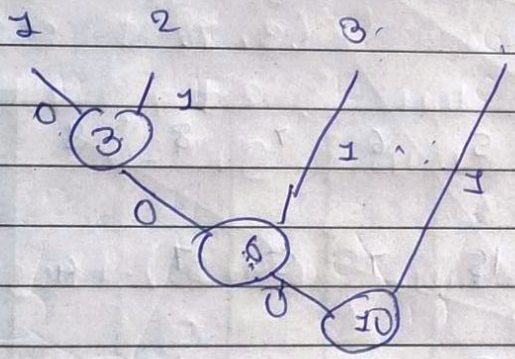
char	Count	code
w	1	00
x	2	01
y	3	10
z	4	11

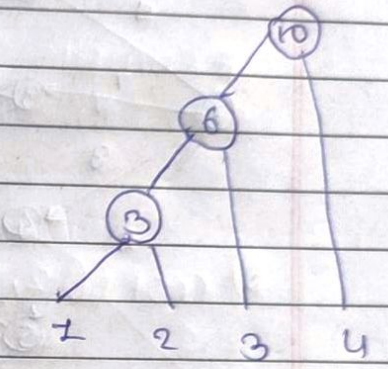
Size = 10 × 2
= 20

Total size msg = size msg + size table
= 20 + 40
= 60

Variable size coding

w	1
x	2
y	3
z	4





3+6+10 = 19

Assume left side : 0 Right side = 1

Char	count	code	
w	1	000	1 × 3 = 3
x	2	100	2 × 3 = 6
y	3	10	3 × 2 = 6
z	4	1	4 × 1 = 4
			19

4 × 8 = 32

cost size of msg : 19 bits

size of table = size char + size code
= 32 + 9 = 41

Page No.
Date

total cost size msg = size msg + size
= 19 + 41
= 60