UNIT-II

Divide and Conquer :- General methods, applications - Binary search, Quick Sort, Merge sort, finding the maximum and minimum.

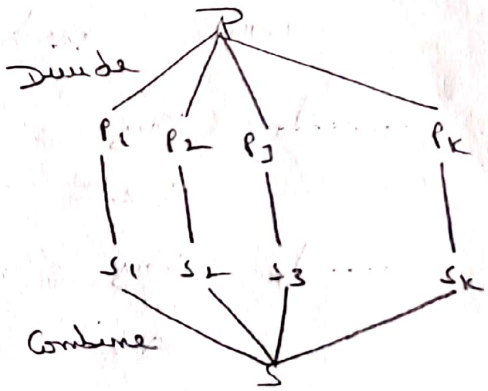
Divide and Conquer (General Methods)

→ If the given problem is a small problem then return a solution.
→ If the given problem is a large then apply divide and conquer method.
→ If the given problem is large then it can be divided into possible number of sub problems.
→ For each sub problem we have to find out solutions.
→ Finally all the solutions are combined.

Algorithm (or) Control abstraction

```
Algorithm DAC(P)
{
  if Small(P) then
    return S(P);
  else
  {
    divide P into P1, P2, ..... , Pk;
    Apply DAC(P1), DAC(P2), ..... , DAC(Pk);
    Combine (DAC(P1), DAC(P2), ..... , DAC(Pk))
  }
}
```

Prob. Representation



Note :- All the DAC algorithms are recursive hence we can find out the time complexity using Recurrence Relation.

Binary search :-

→ In the binary search method the elements are sorted order i.e ascending order

→ If we want search for the element say "x"

→ then we first divided the list at middle so that two sublists are created.

→ If x greater than middle then right sub list Consider and x is searched in the right sub list otherwise x is searched in the left sub list.

Ex :- the given list of elements are 3, 6, 8, 12, 14, 17, 25, 29, 31, 36, 42, 47, 53, 55, 62 and key element is 42. then find a key element 42 is available (or) not

Soln :- Consider the given list of elements are

3	6	8	12	14	17	25	29	31	36	42	47	53	55	62
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Let us assume l = 1 ; h = 15 ; n = 15

If always low less than are equal to high.

Now $mid = \frac{l+h}{2} = \frac{1+15}{2} = 8$

If (key == mid) then

42 == 29 false

If (key < mid) then

42 < 29 false

If (key > mid) then

42 > 29 True

then low = mid + 1 = 8 + 1 = 9

Now l = 9 ; h = 15

$\therefore mid = \frac{l+h}{2} = \frac{9+15}{2} = 12$

If (key == mid) then

42 == 47 false

if (key < mid) then

42 < 47 ..... true

then high = mid - 1 = 12 - 1 = 11

now l = 9 ; h = 11

∴ mid = $\frac{l+h}{2}$ = $\frac{9+11}{2}$ = $\frac{20}{2}$ = 10

if (key == mid) then

42 == 36 false

if (key < mid) then

42 < 36 false

if (key > mid) then

42 > 36 true

then low = mid + 1 = 10 + 1 = 11

now l = 11 ; h = 11

∴ mid = $\frac{l+h}{2}$ = $\frac{11+11}{2}$ = 11

if (key == mid) then

42 == 42 true

Hence element 4 found.

Ex:- 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99.

key element is 40.

Sol:-

11	22	30	33	40	44	55	60	66	77	80	88	99
1	2	3	4	5	6	7	8	9	10	11	12	13

Let us assume l = 1 ; h = 13 ; n = 13

If always low less than are equals to high

now mid = $\frac{l+h}{2}$ = $\frac{1+13}{2}$ = 7

if (key == mid) then

40 == 55 false

if (key < mid) then

40 < 55 true

then high = mid - 1 = 7 - 1 = 6

now l = 1 ; h = 6

$\therefore$ mid = $\frac{l+h}{2} = \frac{1+6}{2} = 3$

if (key == mid) then

40 == 30 false

if (key < mid) then

40 < 30 false

if (key > mid) then

40 > 30 true

then low = mid + 1 = 3 + 1 = 4

now l = 4 ; h = 6

$\therefore$ mid = $\frac{l+h}{2} = \frac{4+6}{2} = 5$

if (key == mid) then

40 == 40 true

Hence the element is found.

Ex:- 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99.

key element is 62.

Soln:-

11	22	30	33	40	44	55	60	66	77	80	88	99
1	2	3	4	5	6	7	8	9	10	11	12	13

Let us Assume l = 1 ; h = 13 ; n = 13

If always low lesser than or equal to high.

now mid = $\frac{l+h}{2} = \frac{1+13}{2} = 7$.

if (key == mid) then

62 == 55 false

if (key < mid) then

62 < 55 false

if (key > mid) then

62 > 55 . true

then low = mid + 1 = 7 + 1 = 8

now l = 8; h = 17

∴ mid = $\frac{l+h}{2}$ = $\frac{8+17}{2}$ = 10

if (key == mid) then

62 == 77 false

if (key < mid) then

62 < 77 true

then high = mid - 1 = 10 - 1 = 9

now l = 8; h = 9

∴ mid = $\frac{l+h}{2}$ = $\frac{8+9}{2}$ = 8

if (key == mid) then

62 == 60 false

if (key < mid) then

62 < 60 false

if (key > mid) then

62 > 60 true

then low = mid + 1 = 8 + 1 = 9

now low = 9 ; h = 9

∴ mid = $\frac{l+h}{2}$ = $\frac{9+9}{2}$ = 9

if (key == mid) then

62 == 66 false

if (key < mid) then

62 < 66 true

then high = mid - 1 = 9 - 1 = 8

now l = 9 ; h = 8 ⇒ l > h

⇒ the element is not present.

Iterative Binary search Algorithm

```
int BinSearch (A, n, key)
{
    l = 1; h = n;
    while (l ≤ h)
    {
        mid = (l+h)/2;
        if (key == A[mid])
            return mid;
        if (key < A[mid])
            h = mid - 1;
        else
            l = mid + 1;
    }
    return 0;
}
```

the Time Complexity of Binary Search is

	Best	average	worst
successful	O(1)	O(log n)	O(log n)
unsuccessful	O(log n)	O(log n)	O(log n)

Recursive Binary search algorithm

T(n) — Algorithm RBinsearch (l, h, key)

```
{
    if (l == h)
    {
        if (A[l] == key)
            return l;
        else
            return 0;
    }
    else
    {
        mid = (l+h)/2;
        if (key == A[mid])
            return mid
        if (key < A[mid])
            return RBinsearch (l, mid-1, key);
        else
            return RBinsearch (mid+1, h, key);
    }
}
```

1 — (base case)

1 — (equality check)

T(n/2) — (recursive calls)

Recurrence Relation

$$T(n) = \begin{cases} 1 & ; n = 1 \\ T(n/2) + 1 & ; n > 1 \end{cases}$$

Consider $T(n) = T(n/2) + 1$ ——— (1)

Replace n by n/2 in equation (1), we get

$T(n/2) = T(n/2^2) + 1$ ——— (2)

Substitute eqn (2) in eqn (1), we get

$T(n) = [T(n/2^2) + 1] + 1$

$T(n) = T(n/2^2) + 2$ ——— (3)

Replace n by $n/2^2$ in eqn (1), we get

$T(n/2^2) = T(n/2^3) + 1$ ——— (4)

Substitute eqn (4) in eqn (3), we get

$T(n) = [T(n/2^3) + 1] + 2$

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

$\vdots$

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

the Algorithm will terminate if $\frac{n}{2^k} = 1$

$\Rightarrow n = 2^k$

$\Rightarrow \log_2 n = k$

$\therefore k = \log_2 n$

Hence $T(n) = T(1) + \log_2 n$

$= 1 + \log_2 n$

$\Rightarrow$ the time complexity as $O(\log_2 n)$.

Finding Maximum and Minimum element

→ Identify low index and high index

→ calculate mid value based on low and high indexes. i.e mid = $\lfloor \frac{l+h}{2} \rfloor$

→ Based on the mid element the given list can be divided into two sub lists.

1) left sub list
2) Right sub list.

→ Consider left sub list, identify low, high indexes and calculate mid element.

→ until the left sub list contain one (or) two elements in the list.

→ Consider Right sub list, identify low, high indexes and calculate mid element.

→ until the right sub list contain one (or) two elements in the list.

→ Compare left and Right sub lists identify maximum element and minimum element.

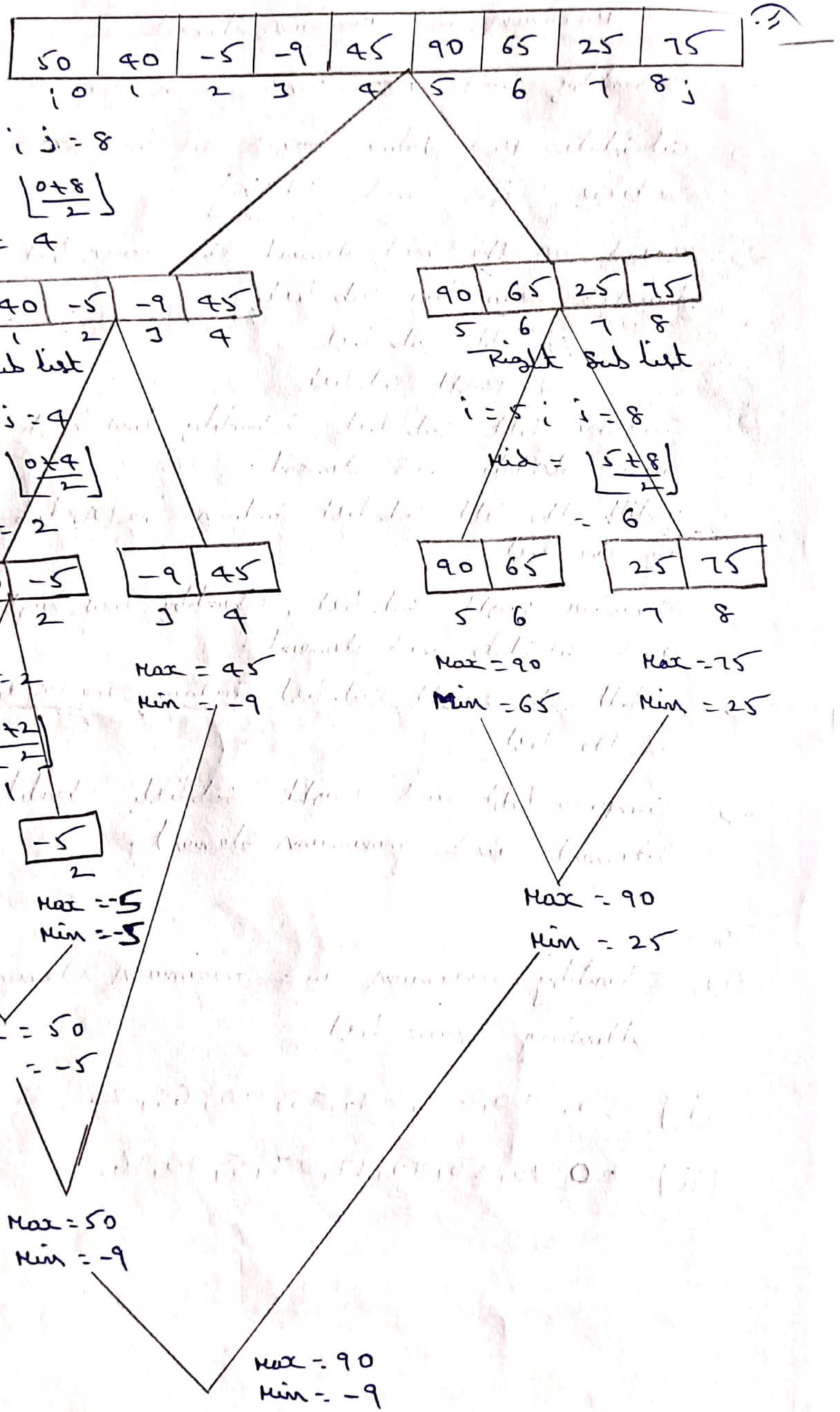
Ex:-

① Identify maximum and minimum element from the following given list

(i) 50, 40, -5, -9, 45, 90, 65, 25, 75.

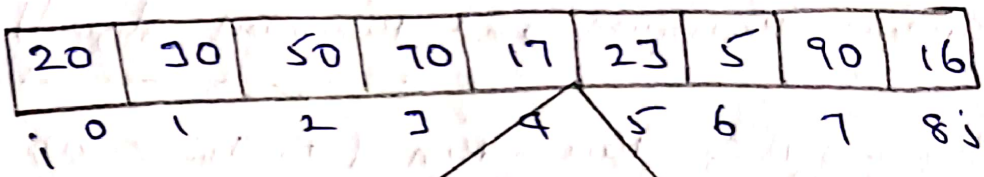
(ii) 20, 30, 50, 70, 17, 23, 5, 90, 16.

i)



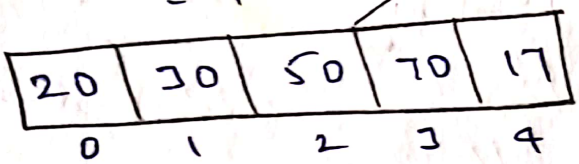
Here the maximum number is 90 and minimum number is -9

ii)



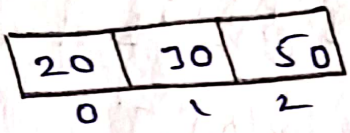
$i=0$, $j=8$

$Mid = \left\lfloor \frac{0+8}{2} \right\rfloor = 4$



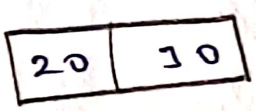
$i=0$, $j=4$

$Mid = \left\lfloor \frac{0+4}{2} \right\rfloor = 2$

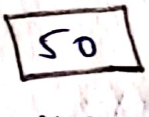


$i=0$, $j=2$

$Mid = \left\lfloor \frac{0+2}{2} \right\rfloor = 1$

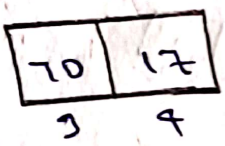


Max = 30
Min = 20



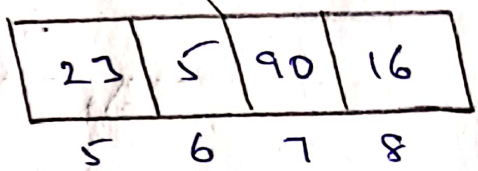
Max = 50
Min = 50

Max = 50
Min = 20



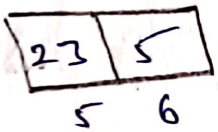
Max = 70
Min = 17

Max = 70
Min = 17

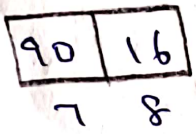


$i=5$, $j=8$

$Mid = \left\lfloor \frac{5+8}{2} \right\rfloor = 6$



Max = 23
Min = 5



Max = 90
Min = 16

Max = 90
Min = 5

Max = 90
Min = 5

Hence the maximum number is 90
minimum number is 5

Recursive Maximum and Minimum Algorithm

$T(n)$ —— DAC Max-Min (A, i, j, max, mini)

{

0 —— if (i == j)

max = mini = A[i];

10 → 0 Comparison

i j

else

if (i == j-1)

i j

10 | 20 → 1 Comparison

0 1

1 —— {

if (A[i] < A[j])

max = A[j]; mini = A[i];

else

max = A[i]; mini = A[j];

}

else

{ mid = $\frac{i+j}{2}$;

$T(n/2)$ —— DAC Max-Min (A, i, mid, max, mini)

$T(n/2)$ —— DAC Max-Min (A, mid+1, j, max, mini)

1 —— {

if (max$_1$ < max$_2$)

Max = max$_2$;

else

Max = max$_2$;

1 —— {

if (min$_1$ < min$_2$)

Mini = min$_1$;

else

Mini = min$_2$;

}

}

Recurrence Relation :-

$$T(n) = \begin{cases} 0 & ; n = 1 \\ 1 & ; n = 2 \\ T(n/2) + T(n/2) + 2 & ; n > 2 \end{cases}$$

Now $T(n) = 2T(n/2) + 2$ ——— (1)

Replace n by $n/2$ in eqn (1), we get

$T(n/2) = 2T(n/2^2) + 2$ ——— (2)

Substituting eqn (2) in eqn (1), we get

$T(n) = 2\left[2T(n/2^2) + 2\right] + 2$

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

Replace n by $n/2^2$ in eqn (1), we get

$T(n/2^2) = 2T(n/2^3) + 2$ ——— (4)

Substituting eqn (4) in eqn (3), we get

$T(n) = 2^2\left[2T(n/2^3) + 2\right] + 2^2 + 2$

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

$\vdots$

$T(n) = 2^k T(n/2^k) + \underbrace{2^k + 2^{k-1} + \cdots + 2^2 + 2}_{\text{G.P}}$

the Algorithm will terminate $\frac{n}{2^k} = 2 \Rightarrow \frac{n}{2} = 2^k$

$\therefore T(n) = \frac{n}{2} T(2) + \frac{2(2^k - 1)}{2-1}$

$= \frac{n}{2}(1) + 2^{k+1} - 2$

$= \frac{n}{2} + 2^k \cdot 2 - 2$

$= \frac{n}{2} + n - 2$

$= \frac{3n}{2} - 2$

Hence time complexity of $O(n)$.

Merge Sort :-

→ In merge sort, based on the middle element, the list can be divided into left sublist and right sublist.

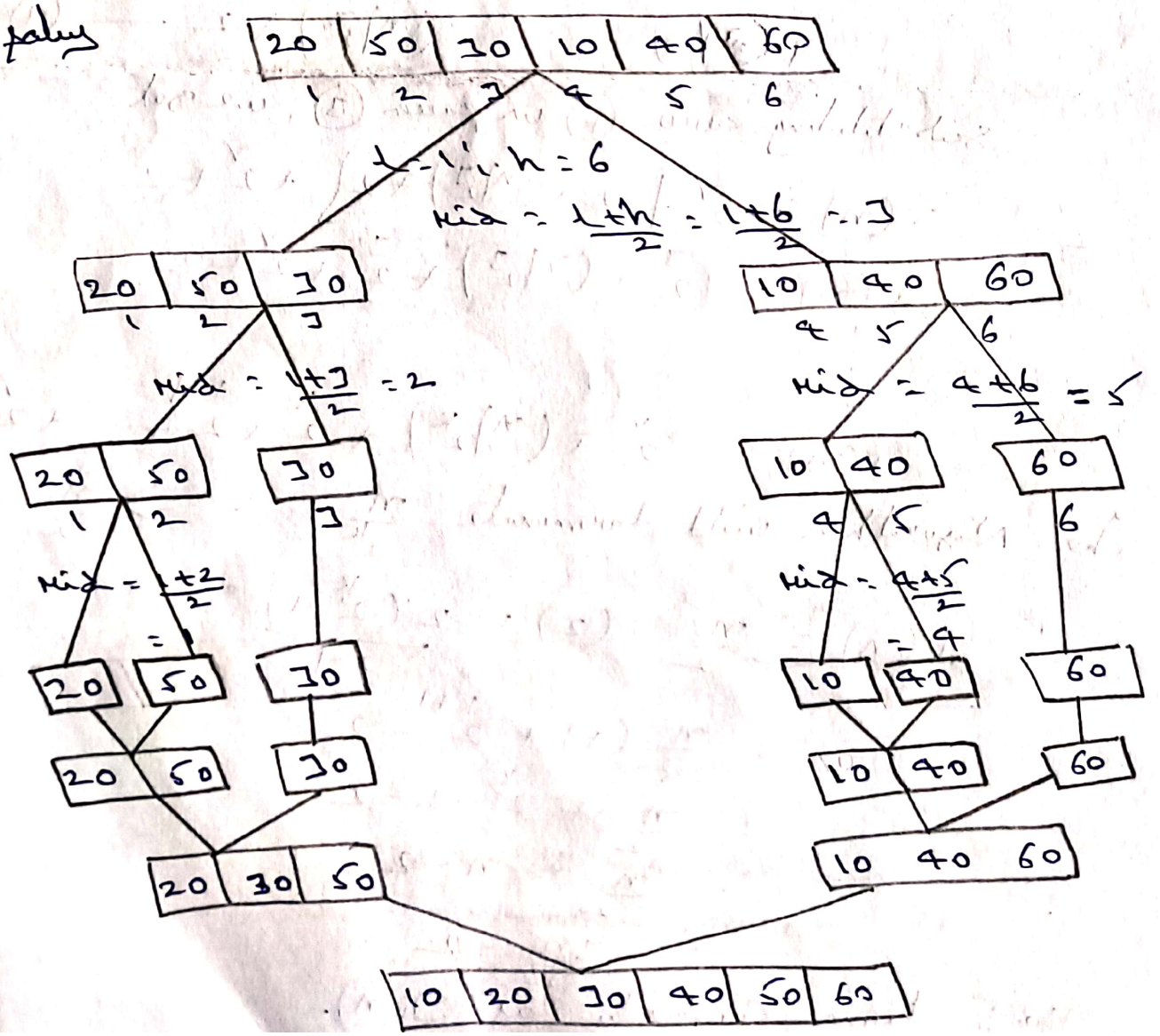
→ Again consider left sublist find out mid value again it can be divide into left sublist 1 and Right Sublist 1, untill we get 1 element in the list. Then Combine all the elements.

→ In the same way right sublist also divide the list untill we get 1 element in the list.

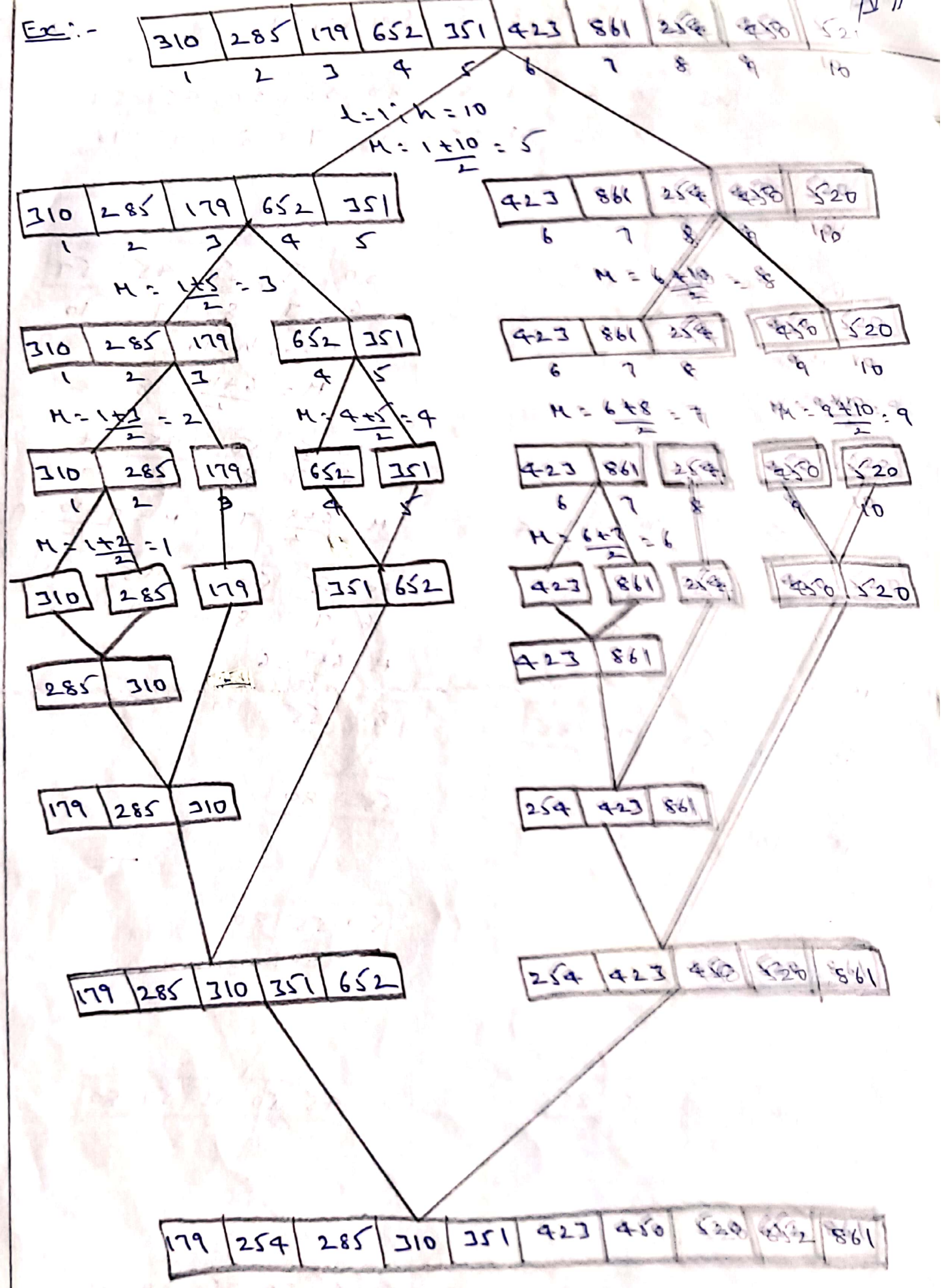
→ Finally Combined left and right sort list

Ex:- 20 50 30 10 40 60

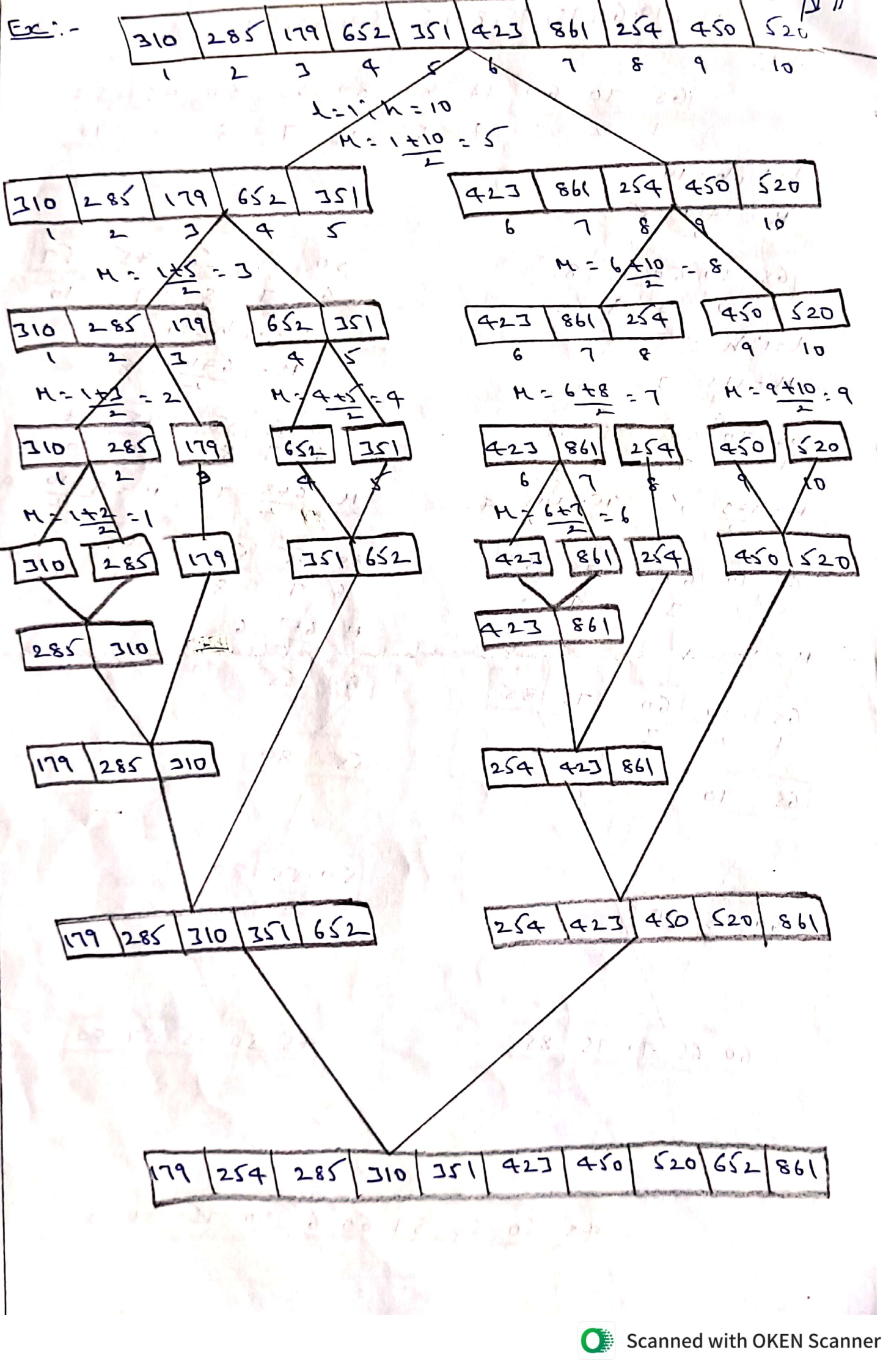
Solu



Ex:-


310 285 179 652 351 423 861 254 450 520
1 2 3 4 5 6 7 8 9 10
l = 1, h = 10
M = 1+10/2 = 5
310 285 179 652 351
1 2 3 4 5
M = 1+5/2 = 3
310 285 179
1 2 3
M = 1+3/2 = 2
310 285
1 2
M = 1+2/2 = 1
310
285 310
179 285 310
652 351
M = 4+5/2 = 4
423 861 254 450 520
6 7 8 9 10
M = 6+10/2 = 8
423 861 254
6 7 8
M = 6+8/2 = 7
423 861
6 7
M = 6+7/2 = 6
423 861
254 423 861
450 520
M = 9+10/2 = 9
179 254 285 310 351 423 450 520 652 861

Ex:-


310 285 179 652 351 423 861 254 450 520
1 2 3 4 5 6 7 8 9 10
l=1, h=10
M = 1+10/2 = 5
310 285 179 652 351
1 2 3 4 5
423 861 254 450 520
6 7 8 9 10
M = 1+5/2 = 3
M = 6+10/2 = 8
310 285 179
1 2 3
652 351
4 5
423 861 254
6 7 8
450 520
9 10
M = 1+3/2 = 2
M = 4+5/2 = 4
M = 6+8/2 = 7
M = 9+10/2 = 9
310 285
1 2
179
3
652
4
351
5
423 861
6 7
254
8
450 520
9 10
M = 1+2/2 = 1
M = 6+7/2 = 6
310
285 310
179 285 310
351 652
423 861
254 423 861
179 285 310 351 652
254 423 450 520 861
179 254 285 310 351 423 450 520 652 861

Recursive Merge Sort Algorithm

```
T(n) — Algorithm Merge sort (low, high)
{
    if (low < high)
    {
        mid = ⌊(low + high)/2⌋ ;
T(n/2)  —— Merge sort (low, mid) ;
T(n/2)  —— Merge sort (mid+1, high) ;
n       —— Merge (low, mid, high) ;
    }
}
```

Recurrence Relation :-

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

Now we have $T(n) = 2T(n/2) + n$ ——— (1)

Replace n by $n/2$ in equation (1), we get

$$T(n/2) = 2T(n/2^2) + \frac{n}{2} \quad \text{——— (2)}$$

Substituting equation (2) in equation (1), we get

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

$$T(n) = 2^2 T(n/2^2) + 2n \quad \text{——— (3)}$$

Replace n by $n/2^2$ in equation (1), we get

$$T(n/2^2) = 2T(n/2^3) + n/2^2 \quad \text{——— (4)}$$

Substituting equation (4) in equation (3), we get

$$T(n) = 2^2\left[2T(n/2^3) + \frac{n}{2^2}\right] + 2n$$

$$T(n) = 2T\left(\frac{n}{2}\right) + 1n$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

the Algorithm will terminate if $\frac{n}{2^k} = 1$

$$n = 2^k$$

$$k = \log_2 n$$

$$\therefore T(n) = nT(1) + \log_2 n \cdot n$$

$$= n(1) + n\log_2 n$$

$$= n + n\log_2 n$$

Hence time complexity of $O(n\log_2 n)$.

Quick sort :-

→ In quick sort method we are going to identify 'i (low)', 'j (high)' and Pivot.

→ Always Pivot = a[i];

→ 'i' has to be incremented when a[i] ≤ Pivot

→ 'j' has to be decremented when a[j] > Pivot

→ otherwise no change to be made, if i and j has stop then swap a[i] and a[j]

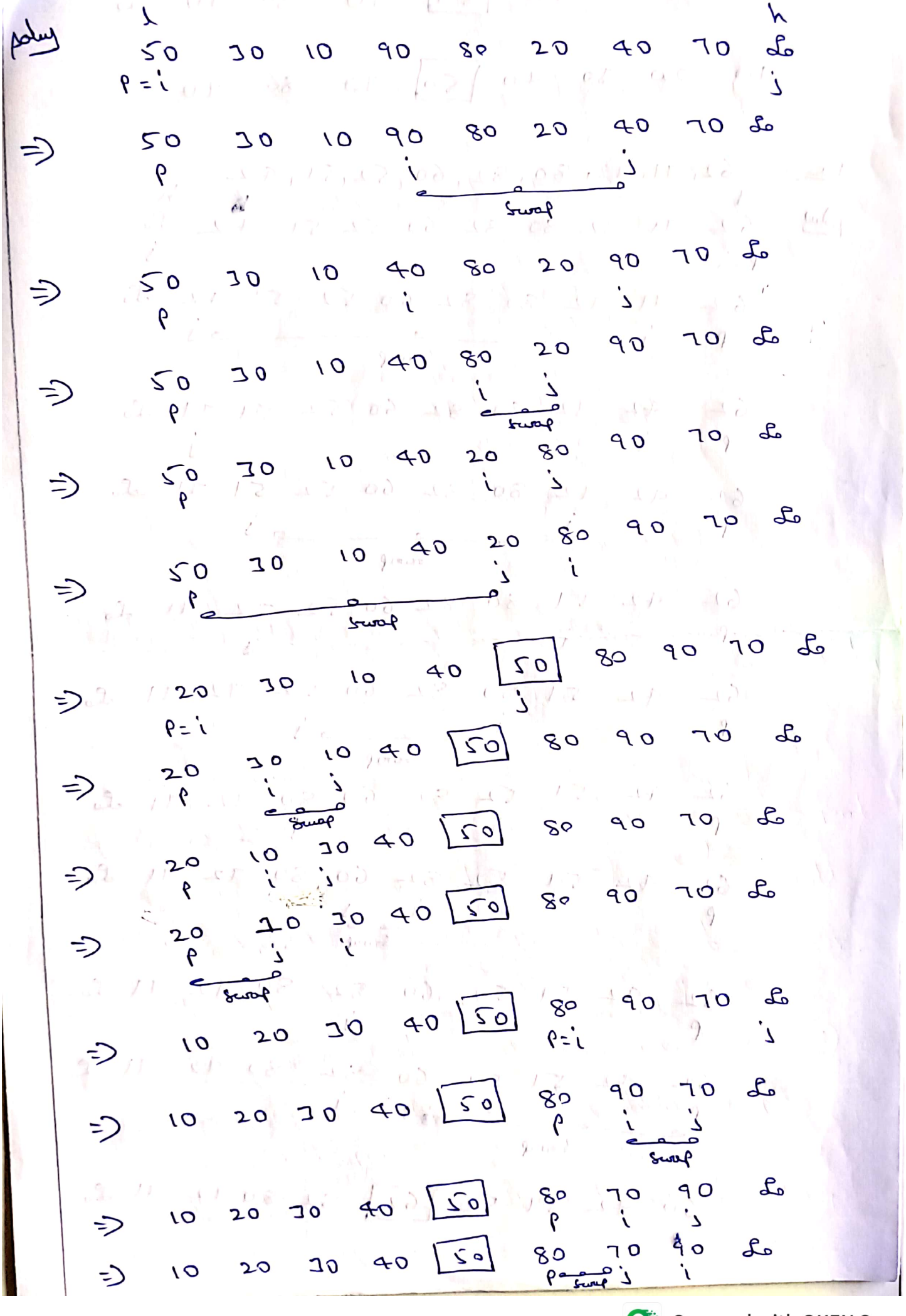
→ then again Continue the process

→ If 'i' & 'j' Crossed each other then swap a[j] and Pivot Also when i, j are in same place.

→ finally, we can generate the list in sorted order. i.e Ascending order.

Ex:- 50, 30, 10, 90, 80, 20, 40, 70

Sol:



⇒ 10 20 30 40 [50] 70 80 90 ∅

⇒ 10 20 30 40 [50] 70 80 90.

Ex:- 62, 71, 72, 80, 82, 60, 52, 51, 42

Sol:- 62 71 72 80 82 60 52 51 42 ∅

p=i j

62 71 72 80 82 60 52 51 42 ∅

p i j

swap

62 42 72 80 82 60 52 51 71 ∅

p i j

62 42 72 80 82 60 52 51 71 ∅

p i j

swap

62 42 51 80 82 60 52 72 71 ∅

p i j

62 42 51 80 82 60 52 72 71 ∅

p i j

swap

62 42 51 52 82 60 80 72 71 ∅

p i j

62 42 51 52 82 60 80 72 71 ∅

p i j

swap

62 42 51 52 60 82 80 72 71 ∅

p i j

62 42 51 52 60 82 80 72 71 ∅

p j i

swap

60 42 51 52 [62] 82 80 72 71 ∅

60 42 51 52
p=i j

60 42 51 52 .....
p j i
swap

52 42 51 60

52 42 51
p=i j

52 42 51 .....
p j i
swap

51 42 52

51 42
p=i j

51 42 .....
p j i
swap

42 51

⇒ 42 51 52 60

82 80 72 71
p=i j

82 80 72 71 .....
p j i
swap

71 80 72 82

71 80 72
p=i j

71 80 72
p=j i

71 80 72

80 72
p=i j

80 72 .....
p j i
swap

72 80

71 72, 80, 82

∴ 42 51 52 60 71 72 80 82

Hence the elements are sorted.

Ex:- 65, 70, 75, 80, 85, 60, 55, 50, 45.

Quick sort Algorithm :-

```
Algorithm Quick sort (A,l,h) ——— T(n)
{
    if (l<h) then
    {
        j = Partition (A,l,h); ——— n
        Quick sort (l, j-1);
        Quick sort (j+1, h);
    }
}

Partition (A,l,h)
{
    pivot = A[l]
    i = l; j = h
    while (i<j)
    {
        while (A[i] <= pivot)
        { i++
        }
        while (A[j] > pivot)
        { j--
        }
        if (i<j) then
        { swap (A[i], A[j]);
        }
    } swap (A[l], A[j]);
      return j;
}
```

Time Complexity for Quick sort :-

(i) **Best and average Case :** (Partition at the middle)

To Construct left sublist i.e $T(n/2)$

To Construct right sublist i.e $T(n/2)$

Time to build a partition i.e n

$$\therefore T(n) = 2T(n/2) + n$$

Now the recurrence relation is

$$T(n) = \begin{cases} 1 & ; n = 1 \\ 2T(n/2) + n & ; n > 1 \end{cases}$$

Consider $T(n) = 2T(n/2) + n$ ——— (1)

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

$$T(n) = 2^2 T(n/2^2) + 2n \text{ ——— (2)}$$

$$T(n) = 2^2\left[2T(n/2^3) + \frac{n}{2^2}\right] + 2n$$

$$T(n) = 2^3 T(n/2^3) + 3n \text{ ——— (3)}$$

$$\vdots$$

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

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

$\Rightarrow k = \log_2 n$

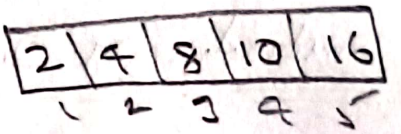
$$\therefore T(n) = n\,T(1) + \log_2 n \cdot n$$

$$= n \cdot 1 + n \log_2 n$$

$$= n + n \log_2 n$$

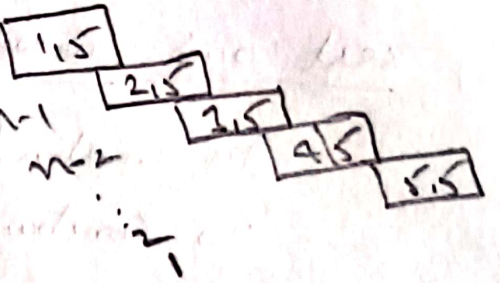
Hence Time Complexity is $O(n \log_2 n)$

(ii) worst case :-



the recurrence relation is

$$T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-1)+n & \text{if } n>0 \end{cases}$$



Consider $T(n) = T(n-1) + n$ ——— (1)

$T(n) = T(n-2) + (n-1) + n$ ——— (2)

$T(n) = T(n-3) + (n-2) + (n-1) + n$ ——— (3)

$\vdots$

$T(n) = T[n-(k+1)] + (n-k) + \cdots + (n-1) + n$

Assume $n-(k+1) = 0 \Rightarrow n-k = 1$

$\therefore T(n) = n + (n-1) + \cdots + 1 + T(0)$

$= n + (n-1) + \cdots + 1 + 1$

$= 1 + 2 + 3 + \cdots + n + 1$

$= \frac{n(n+1)}{2} + 1$

$= \frac{n^2}{2} + \frac{n}{2} + 1.$

Hence time Complexity of $O(n^2)$

Insertion Sort Algorithm :-

```
Algorithm insertion sort (A, n)
{
    for (j=1; j<n; j++)
    {
        key = A[j];
        i = j-1;
        while (i >= 0 and A[i] > key)
        {
            A[i+1] = A[i];
            i = i-1;
        }
        A[i+1] = key
    }
}
```

Comparison & swapping

Time Complexity :- (i) worst case $O(n^2)$ [elements are in descending order]

(ii) Best case $O(n)$ [elements are in ascending order]

Bubble Sort Algorithm (Adaptive) and stable :-

```
Algorithm Bubble sort (A, n)
{
    int flag;
    for (i=0; i<n-1; i++)
    {
        flag = 0;
        for (j=0; j<n-1-i; j++)
        {
            if (A[j] > A[j+1])
```

```
        {
            swap (A[j], A[j+1]);
            flag = 1;
        }
    }
    if (flag == 0) break;
}
}
```

Time Complexity :- (i) worst case $O(n^2)$ (elements are standing reverse)
(ii) Best case $O(n)$ (elements are sorted)

Selection sort Algorithm :-

```
Algorithm selection sort (A, n)
{
    for (i = 0; i < n-1; i++)
    {
        int min = i;
        for (j = i+1; j < n; j++);
        {
            if (A[j] < A[min])
                min = j;
        }
        swap (A[min], A[j]);
    }
}
```

Time Complexity :- (1) worst case $O(n^2)$
(1) Best case $O(n^2)$

Contrast the Quick sort with Merge sort

Quick sort	Merge sort
(1) In Quick sort, the splitting of a list of elements is not necessarily divided into halfs.	(1) In Merge sort, array or list is always divided into halfs ($n/2$) based on divide-and-conquer approch.
(2) Worst-case time complexity of Quick sort is $O(n^2)$	(2) Worst-case time complexity of Merge sort is $O(n \log n)$
(3) Quick sort work well on smaller array.	(3) Merge sort operates fine in any type of array.
(4) Faster than other sorting algorithms for small arrays (Bubble sort, selection sort, insertion sort).	(4) Consistent speed in all types of array.
(5) Less additional storage space requirement.	(5) More additional storage requirement for storing the auxiliary array.
(6) Quick sort is not suitable for larger arrays.	(6) Merge sort is suitable for any type of array.
(7) It is less efficient than Merge sort	(7) Merge sort is more efficient than Quick sort
(8) In Quick sort, the first element is used for the sorting.	(8) Merge sort does not use pivot element for performing the sorting.
(9) It is the internal sorting method because the data that has to be sorted is adjusted at a time in main memory.	(9) It is the external sorting method because the data that has to sorted that can not be accomodated in main memory at the same time and some has to be kept in auxiliary memory.