UNIT-II

Divide and Conquer (General method):

→ If the given Problem is a small problem then return a solution

→ If the given Problem is a large Problem then apply divide and conquer method.

→ If the given Problem is large then it can be divided into sub Problems.

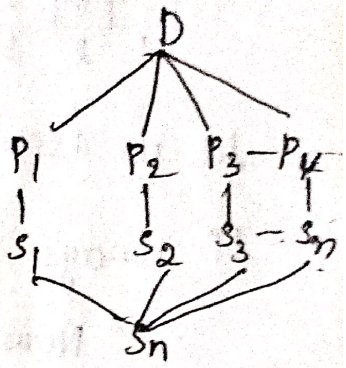
↔ for each sub Problem we have to find out the solution.

→ finally all the solutions are combined

Algorithm (or) total abstraction:

```
Algorithm DAC(P)
{
    if small(P) then
        return S(P);
    else
    {
        divide P into P1, P2, ---, Pn;
        Apply DAC(P1), DAC(P2), ---, DAC(Pn);
        combine (DAC(P1), DAC(P2), --, DAC(Pn));
    }
}
```

(Tree Represen)



NOTE: All the DAC algorithms are recursive hence we can find out the time complexity using Recurance Relation.

Binary Search:

↔ In the binary search method the elements are in sorted order i.e, ascending order

→ If we want to search for element say x then we first divided the list at middle so that two sub lists are created.

→ If x greater than middle then right sublist consider and x is searched in the Right side sublist otherwise x is searched in left sublist.

Eg: 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. the find a key element 42 is available (or) not.

Sol: 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 (or) equal to high

Now mid = $\frac{l+h}{2} = \frac{1+15}{2} = \frac{16}{2} = 8$.

if (key == mid) then

42 == 29 false

if (key < mid) then

42 < 29 false

if (key > mid) then

42 > 29 ~~(false)~~ true

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

Now l = 9, h = 15

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

if (key == mid) then

42 == 47 false

if (key < mid) then

42 < 47 (false) true

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

now l = 9, h = 11

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

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

$\therefore mid = \frac{l+h}{2} = \frac{22}{2} = 11$

if (key == mid) then

42 == 42 True

Hence the element is found

(2) 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99. Key element is 62.

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 (or) equal to high

now $mid = \frac{l+h}{2} = \frac{1+13}{2} = \frac{14}{2} = 7$

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

= 8

now l = 8, h = 13

mid = $\frac{l+h}{2}$ = $\frac{21}{2}$ = 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 = $\frac{l+h}{2}$ = $\frac{8+9}{2}$ = $\frac{17}{2}$ = 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 $l=9$; $h=9$

$mid = \frac{l+h}{2} = \frac{18}{2} = 9$.

if (key == mid) then

62 == 66 false

if (key < mid) then

62 < 66 true

then high = mid - 1

= 9 - 1

= 8

$l=9$, $h=8$

$l > h$

∴ The element is not present.

Complexity:

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;
}
```

Recursive Binsearch 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,
```

```
        }
    }
```

Time complexity:

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

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

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

$$T(n/2) = T(n/4) + 1 \rightarrow (2)$$

Sub equ (2) in equ (1)

$$T(n) = [T(n/4) + 1] + 1 \longrightarrow (3)$$

from (2) then (1) $\Rightarrow T(n) = T(n/4) + 1 + 1 \rightarrow (4)$

from (4) then (3) $\Rightarrow T(n) = T(n/8) + 1 + 1 + 1 \longrightarrow (5)$

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

$$\vdots$$

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

The algorithm will terminate if $\frac{n}{2^k} = 1$

$$n = 2^k$$

$$\log_2 n = k$$

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

$$= 1 + \log_2 n$$

Time complexity is $O(\log_2 n)$

finding maximum and minimum element.

→ Identify low index and high index

→ calculate mid value based on low and high indexes i.e, mid = $\left[\frac{l+h}{2}\right]$.

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

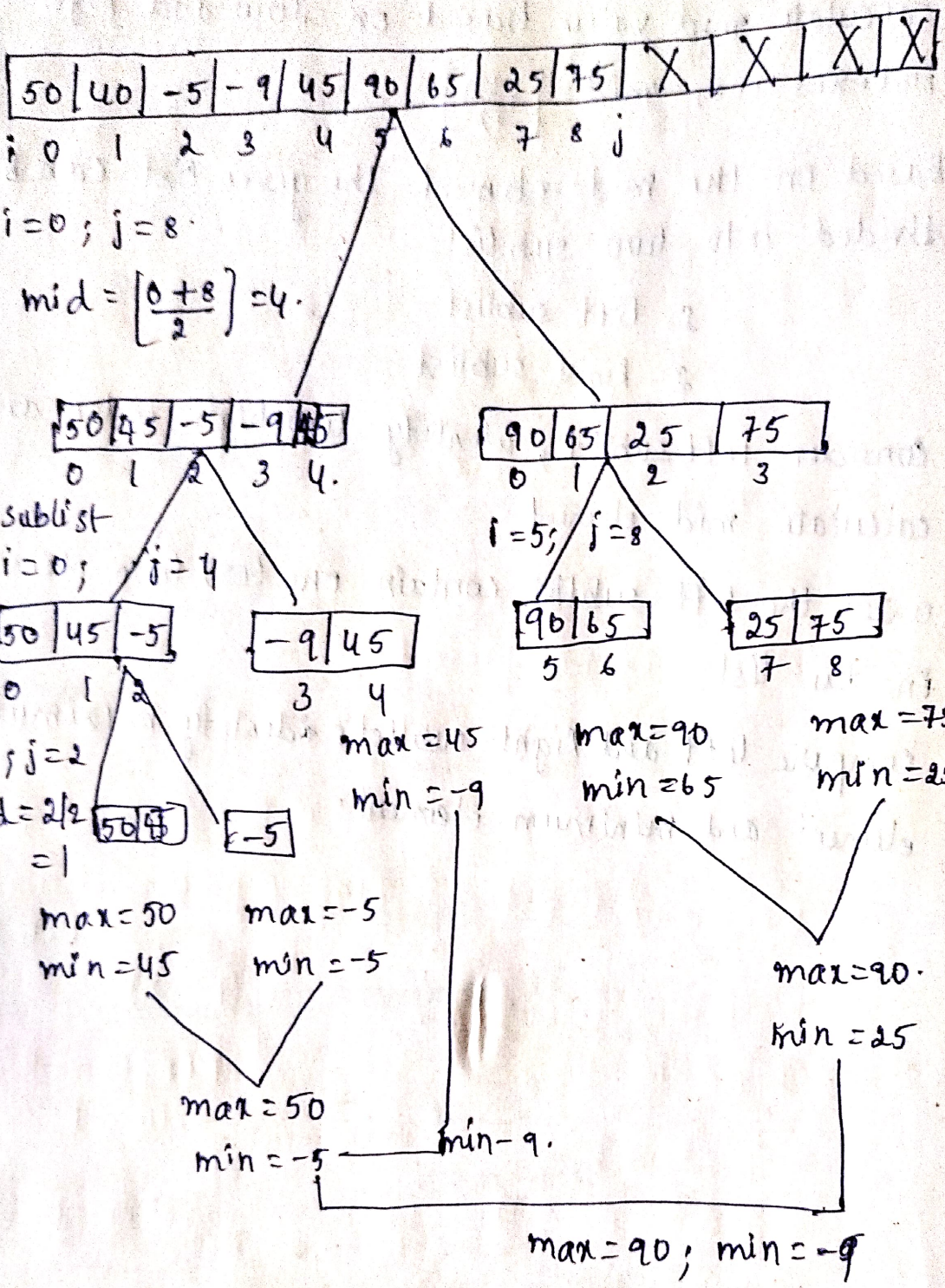
1. Left sublist
2. Right sublist.

→ Consider left sublist, identify low, high index and calculate mid element.

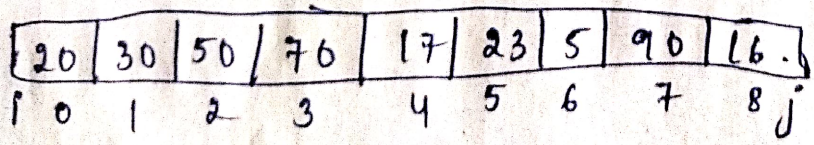
→ until the left sublist contain one (or) two elements in the list.

→ Compare left and Right sublists identify maximum element and minimum element.

(i)

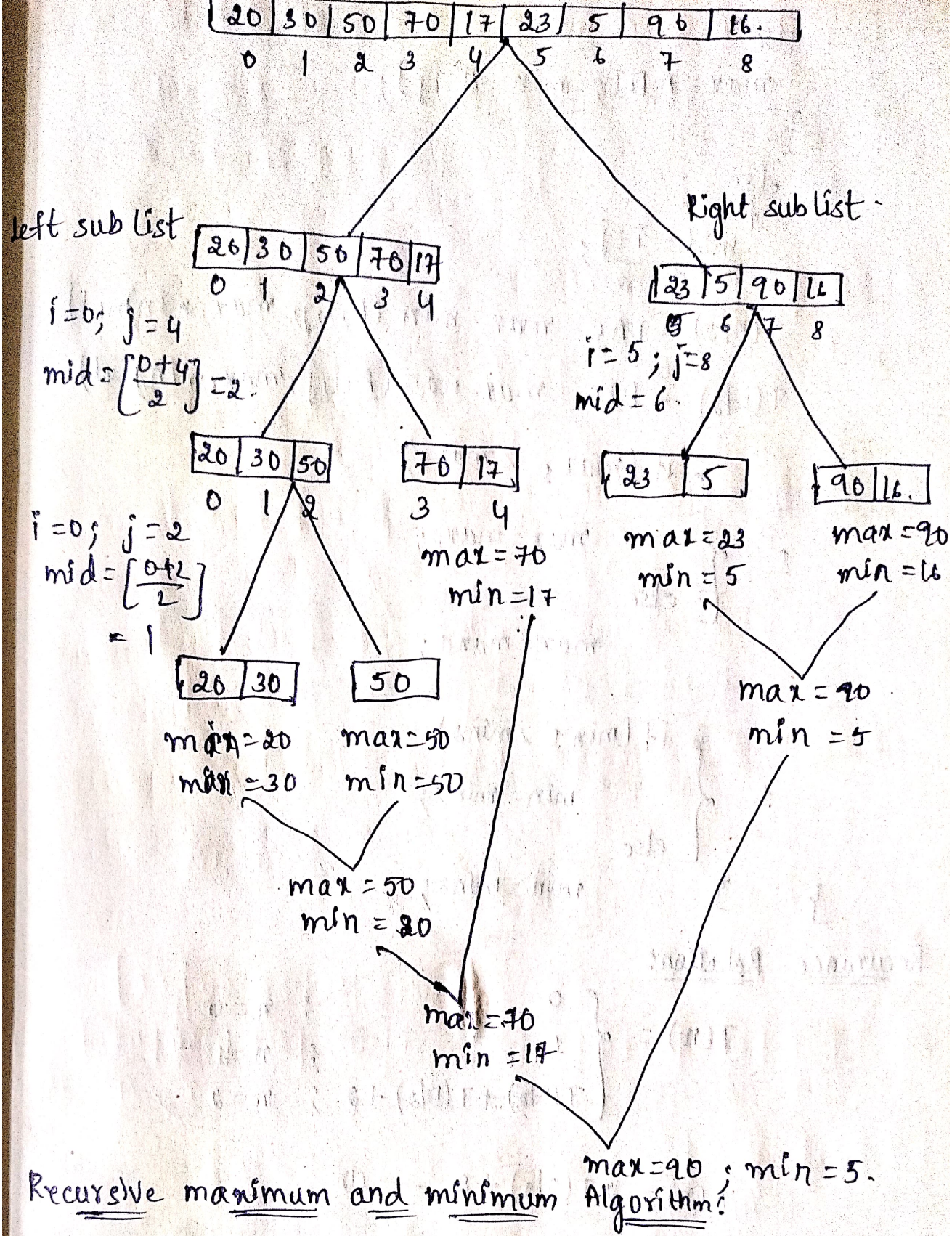


(ii)



i=0 ; j=8

mid = [0+8/2] = 4.



Recursive maximum and minimum Algorithm:

```
T(n) — DAC   max-min(A, i, j, max, min)
{
  if (i==j)
0 ——  max=min=A[i]
  else
  if (i=j-1)
  {
1 {   if (A[i] > A[j])
        max = A[i]; min = A[j]
```

- else

```
    max = A[i]; min = A[j];
}
else
{
    mid = (i+j)/2;
T(n/2) - DAC max-min (A, i, j, max, min, mid)
T(n/2) - DAC max-min (A, i, j, max, min, mid)
1 { if (max1 < max2)
        max = max2;
    else
        max = max1;
1 { if (min1 < min2)
        min = min1;
    else
        min = min2;
} }
```

Recurrance Relation:

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

$$T(n) = 2T(n/2) + 2 \quad \text{---} \textcircled{1}$$

Replace n by n/2 in equ ①

$$T(n/2) = 2T(n/4) + 2 \rightarrow \textcircled{2}$$

Sub ② in equ ①

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

$$T(n) = 4T(n/2^2) + 2^2 + 2 \rightarrow \textcircled{3}$$

Replace n by $n/2^2$ in equ (3)

$$T(n/2^2) = 2T(n/2^3) + 2 \rightarrow (4)$$

Sub equ (4) in (3)

$$T(n) = 4\left[2T(n/2^3) + 2\right] + 2^2 + 2$$

$$T(n) = 8T(n/2^3) + 2^3 + 2^2 + 2 \rightarrow (5)$$

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

The algorithm will terminate $\frac{n}{2^k} = 2 \Rightarrow \frac{n}{2} = 2^1$

$$\therefore T(n) = \cancel{2T(2)} \cdot \frac{n}{2} T(2) + \frac{2(2^k - 1)}{2-1} \qquad \boxed{\frac{a \cdot r^{n-1}}{r-1}}$$

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

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

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

$$T(n) = \frac{3n}{2} - 2$$

Hence, the time complexity is $O(n)$.

③ Merge Sort:

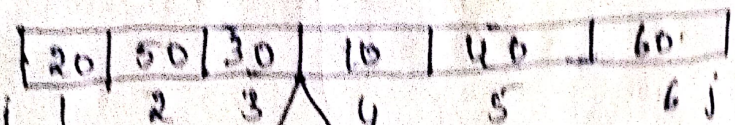
→ In merge sort, based on the middle element the list can be divided into left sublist and Right sublist

→ Again leftsublist and find out mid value again it can be divided into left sublist 1 and Rightsublist -1, untill we get 1 element in the list then combine all the elements

→ In the same way Right sublist also divides the list untill we get 1 element in the list.

→ finally, combine left and Right sorted lists.

Eg: 20 50 30 10 40 60



$mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

$mid = \frac{7}{2} = 3.0$

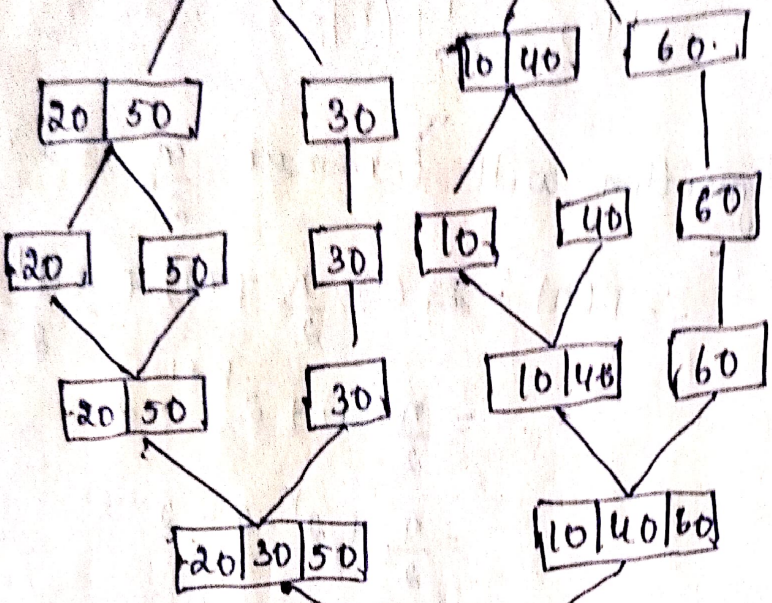
$= 3.$

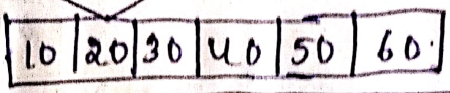
Left sublist

Right sublist

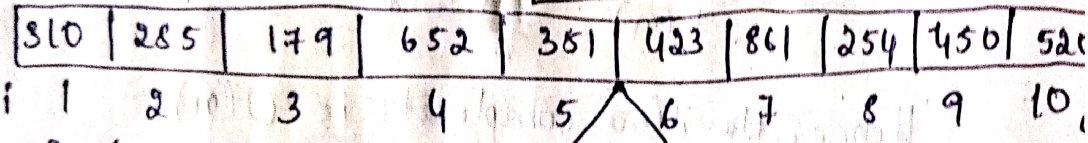
mid = 4/2

= 2.



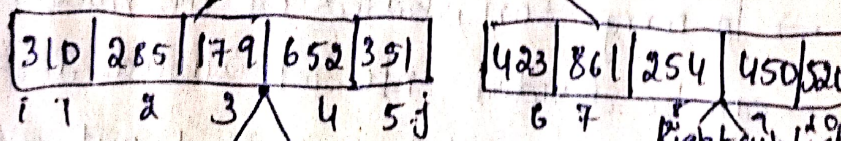


Eg-2:



$mid = \left\lfloor \frac{i+j}{2} \right\rfloor = \left\lfloor \frac{1+10}{2} \right\rfloor$

$= 11/2 = 5.5$

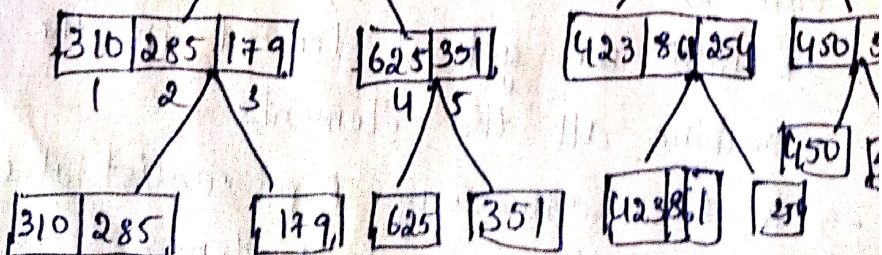


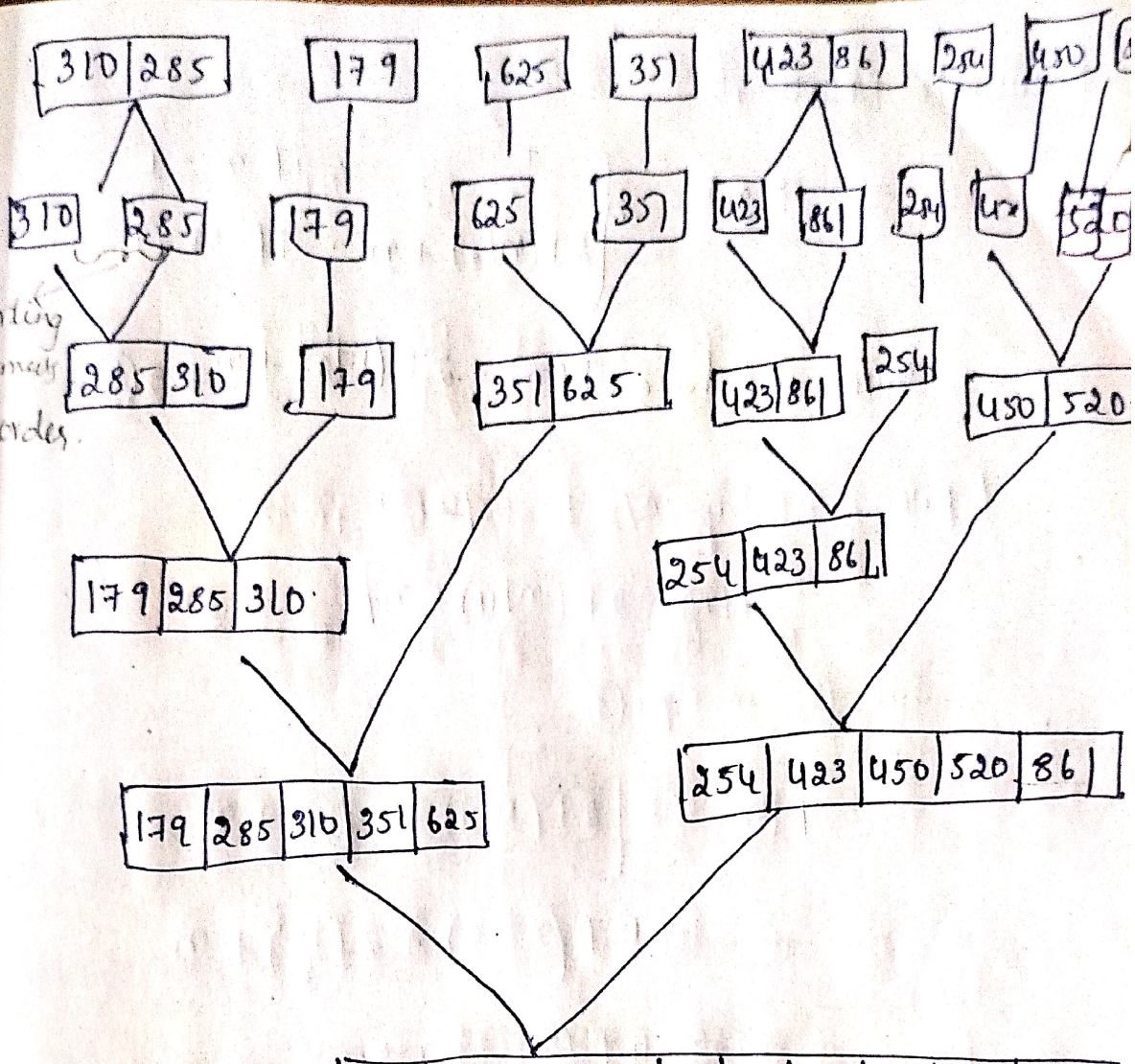
Left sublist

$mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

$= \left\lfloor \frac{6}{2} \right\rfloor$

mid = 3





Sorted list is :→ | 179 | 254 | 285 | 310 | 351 | 423 | 450 | 520 | 625 | 861 |

Algorithm for merge sort:

T(n) - Algorithm merge sort (low, high) (Recursive)

```
{
  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 $T(n) = 2T(n/2) + n \longrightarrow (1)$

Replace n by $n/2$ in equ (1)

$$T(n/2) = 2T(n/4) + n/2 \longrightarrow (2)$$

Sub (2) in equ (1)

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

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

$$T(n) = 4T(n/2^2) + 2n \longrightarrow (3)$$

Replace n by $n/2^2$ in equ (3)

$$T(n/2^2) = 2T(n/2^3) + n/2^2 \longrightarrow (4)$$

Sub (4) in equ (3)

$$T(n) = 4\left[2T(n/2^3) + n/2^2\right] + 2n$$

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

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

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

The algorithm will terminate if $\frac{n}{2^k} = 1$

$$n = 2^k$$

$$k = \log_2 n$$

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

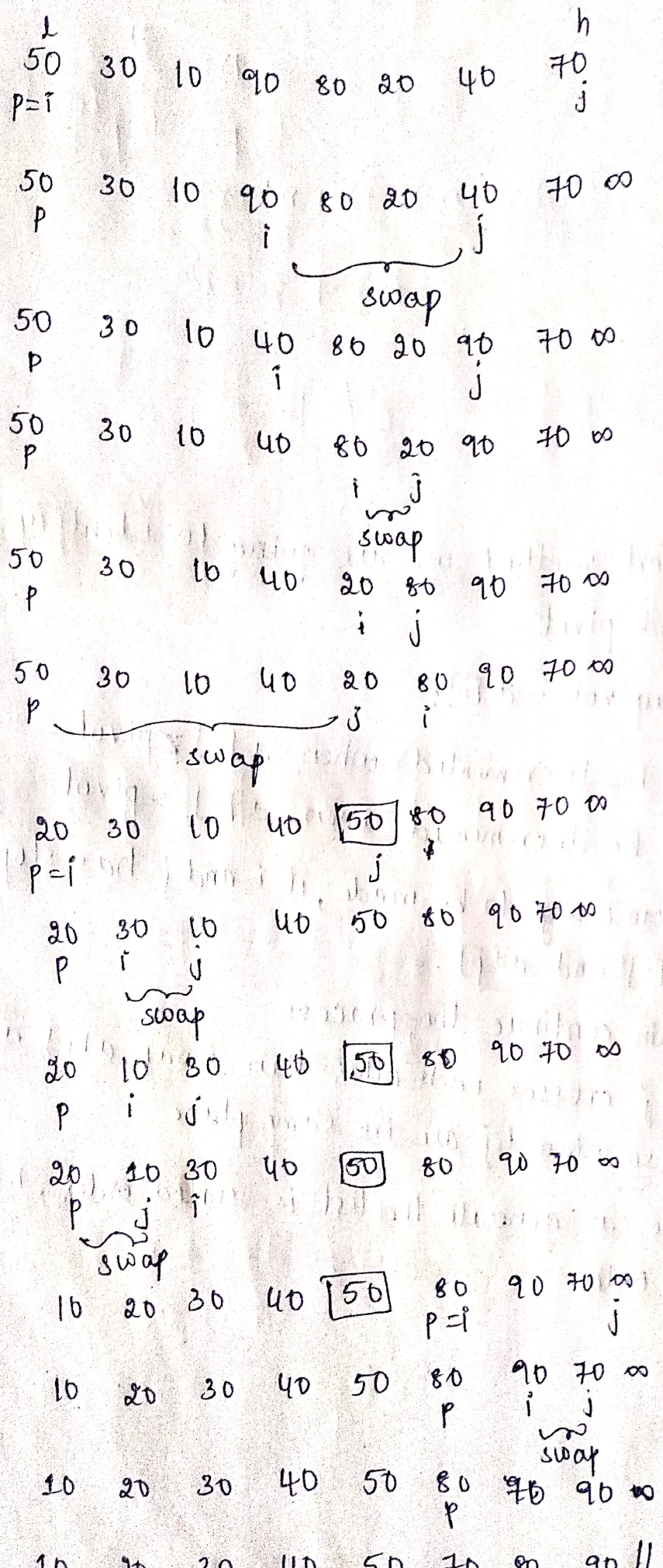
$$= n(1) + n\log_2 n$$

$$O(n\log_2 n)$$

Quick sort:

- In Quicksort method we are going to identify i(low) 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 and j crosses each other then swap a[j] and pivot also when i,j are in same place
- finally we can generate the list is sorted order, i.e ascending order.

Eg: 50, 30, 10, 90, 80, 20, 40, 70.



Eg 2: 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 swap

62 42 72 80 82 60 52 51 71 ∞
P i j

62 42 72 80 82 60 52 51 71 ∞
i j
swap

62 42 51 80 82 60 52 72 71 ∞
i j

12 42 51 80 82 60 52 72 71 ∞
P i j
swap

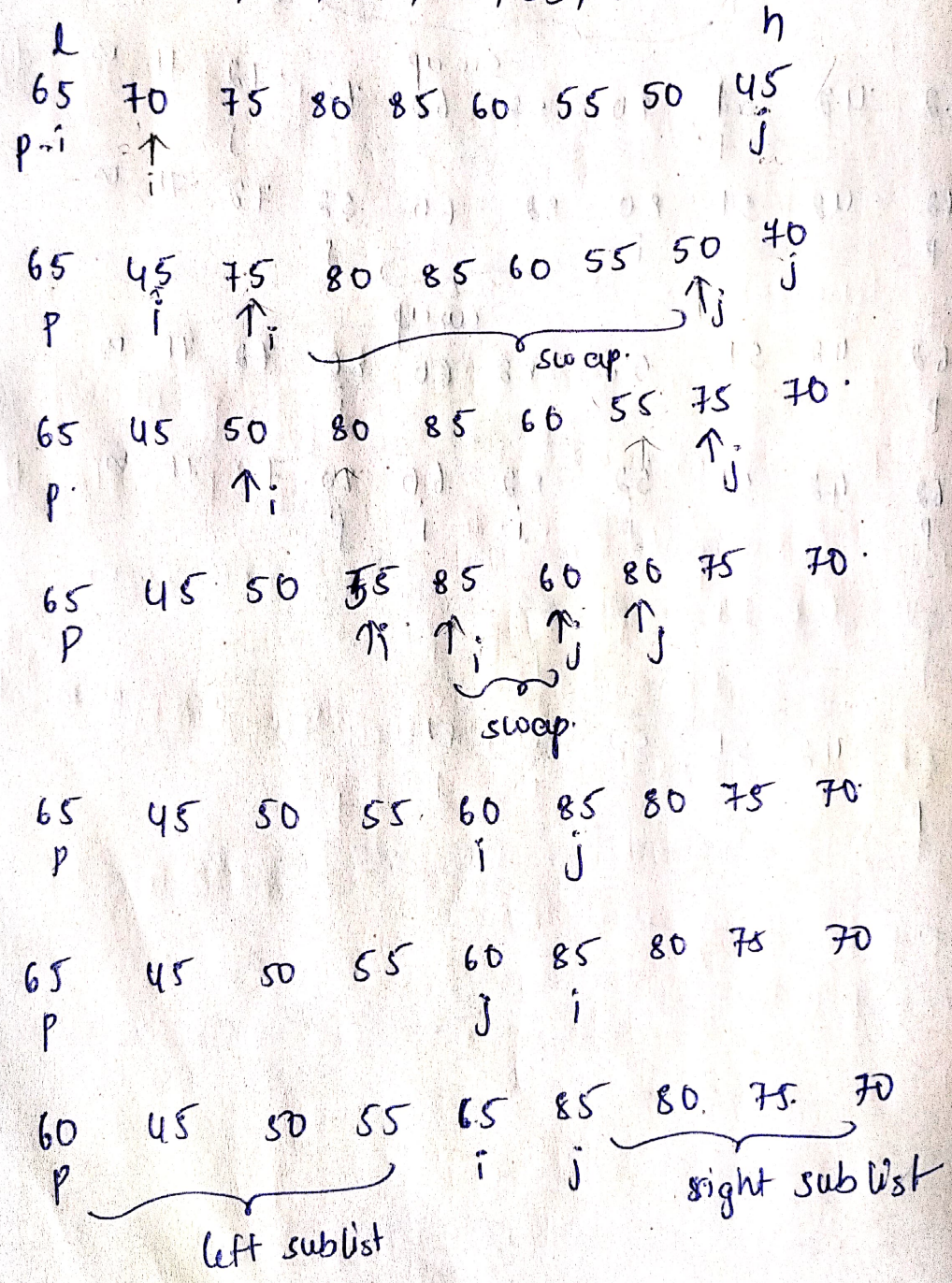
62 42 51 52 82 60 80 72 71 ∞
P i j

62 42 51 52 82 60 80 72 71 ∞
P i j

62 42 51 52 82 60 80 72 71 ∞
P j i

62 42 51 52 60 82 80 72 71 ∞
P i j

3 65, 70, 75, 80, 85, 60, 55, 50, 45



left sublist

60 45 50 55 ∞
pi ↑ ↑ j i

55 45 50 60
p j j

50 45 55 60
p j

45 80 55 60

i
85 80 75 70 ∞
p j i

70 80 75 85
p i j

70 75 80 85

45	50	55	60	65	70	75	80	85

Quick sort algorithm:

```
Algorithm Quicksort(i, j, k)  ——— T(n)
{
if(l<h) then
  {
  j = Partition(A, l, h);  ——— n
  Quicksort (l, j-1);
  Quicksort (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:

1. Best and average case (Partition at the middle)

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

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

To build partition i.e, n

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

now the recurrance 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)

Replace n by $n/2$

$$T(n/2) = 2T(n/4) + n/2 \rightarrow (2)$$

Sub equ ② in equ ①

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

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

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

Replace n by $n/2^2$ then

$$T(n/2^2) = 2T(n/2^3) + n/2^2 \rightarrow ④$$

sub equ ④ in equ ③

$$T(n) = \cancel{2^3[T(n}$$

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

$$T(n) = 2^3 T(n/2^3) + 3n \longrightarrow ⑤$$

$$\vdots$$

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

The algorithm will terminate at $\frac{n}{2^k} = 1$

$$n = 2^k$$

$$k = \log_2 n$$

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

$$= n + n\log_2 n$$

time complexity $= O(n\log_2 n)$

2. 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 $T(n) = T(n-1) + n \rightarrow (1)$

Replace n by $n-1$ in equ (1)

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

Sub equ (2) in equ (1) then

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

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

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

Replace n by $n-2$ in equ (1) then

$$T(n-2) = T(n-3) + n-2 \rightarrow (4)$$

sub equ (4) in equ (3)

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

$$T(n) = T(n-3) + 3n-3$$

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

$$\vdots \qquad \vdots$$

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

the algorithm will terminate at $n-(k+1) = 0$

$$n = k+1$$

$n - k = 1$

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

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

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

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

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

the time complexity is $O(n^2)$

* Distinguish b/w quicksort and merge sort

Quick sort	Merge sort
1. In Quicksort, the spliting of a list of elements in not necessarily divided into half.	1. In merge sort, array (or) list is always divided into half $(n/2)$ based on divide and conquer approach.
2. The worst case time complexity of quick sort is $O(n^2)$	2. The worst case time complexity is $O(n \log \frac{n}{2})$
3. Quick sort works 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 sortings
5. Less additional storage space requirment.	5. more additional storage space requirment for storing the auxillary array

6. Quick sort is not suitable for larger arrays	6. merge sort is suitable for any type of arrays.
7. It is less efficient than merge sort	7. merge sort is more efficie than Quick sort.
8. In Quick sort the pivot element is used for the sorting.	8. In merge sort does not use pivot element for sorting
9. It is the internal sorting method because the data that has to be sorted is adjus-ted at a time in main memory.	9. It is the external sorting method because the data that has to be sorted that cannot be accomodat-ed in main memory at the same time and some has to be kept in auxillary memory.

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 = j-1;
    }
    A[i+1] = key
  }
}
```

comparision & 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:

```
Algorithm Bubble sort (A, n)
{
  int flag;
  for(int i=0; i<n-1; i++)
  {
    flag = 0;
    for(int 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 descending)
(ii) Best case $O(n)$

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: (i) worst case $O(n^2)$

(ii) Best case $O(n^2)$