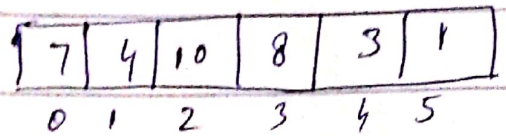
iv

Selection Sort :



where $n = 6$

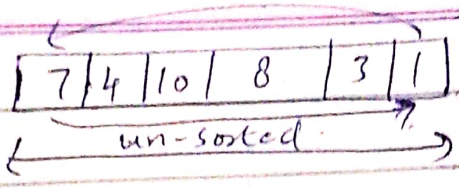
In selection sorting we have two types of list/array. one is sorted and other is un-sorted.

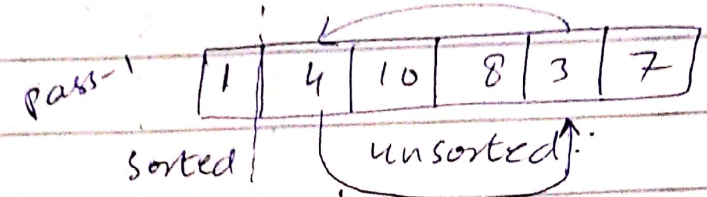
→ initially all the elements are in un-sorted array. after implementing operations, each element will move to its right in sorted list.

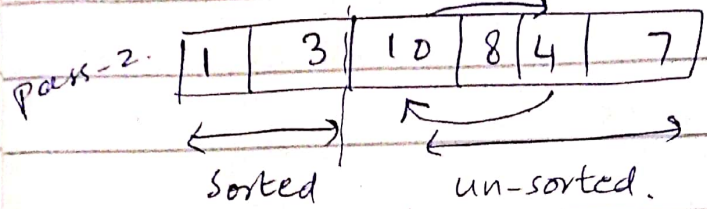
→ initially sorted array will be empty.

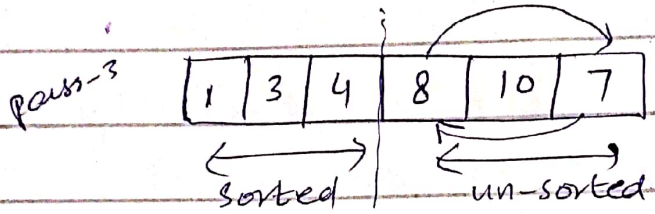
working :

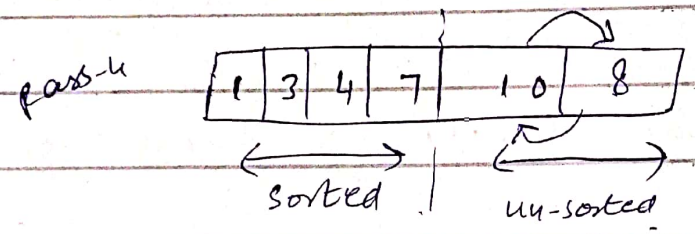
→ initially we have to find the minimum element in the given array and that element will be swapped with the first element of the ~~the~~ unsorted array.

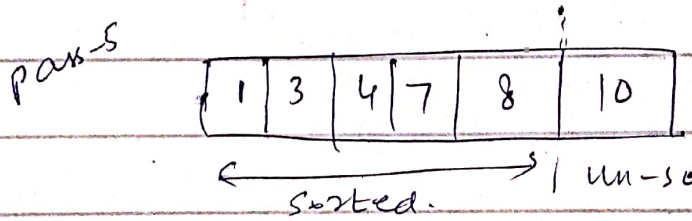










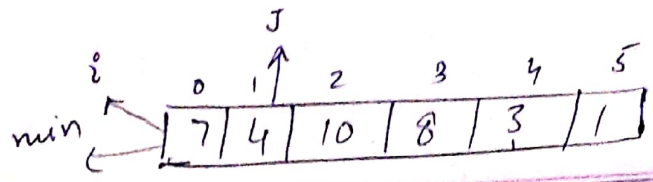


→ when there is one element remaining, then it means that element is sorted and on it's exact position.

total elements 6

No. passes → 5 $(n-1)$

always →



min = 0
i = 0

```
for (i = 0 ; i < n-1 ; i++)
{
    int min = i;
    for (j = i+1 ; j <= n ; j++)
    {
        if (a[j] < a[min])
        {
            min = j;
        }
    }
    if (min != i)
    {
        swap(a[i], a[min]);
    }
}
```

a[0] a[5]

tracing the code:

1st pass

min = 0, 1, 4, 5

J = 1, 2, 3, 4, 5, 6 outer will terminate.

a[1] < a[0] → if block will run.

a[2] < a[1] conditions will be true

a[3] < a[1] → elements will be swapped.

a[4] < a[1]

a[5] < a[4]

Performance Analysis :

Best Case : (Ascending order).

1 3 4 7 8 10

↙

min=1

comparisons would be

$n-1, n-2, \dots 0$

$1 + 2 + 3, \dots n-1$

$\frac{n(n-1)}{2} \Rightarrow O(n^2)$.

$\Rightarrow$ No swapping. $O(1)$

Worst Case : (Deseending order)

10 8 7 4 3 1

comparisons.

$n-1, n-2, \dots 0$

$1 + 2, + 3, \dots n-1$

$\frac{n(n-1)}{2}$

$O(n^2)$

Swapping.

$O(n)$

$O(n)$

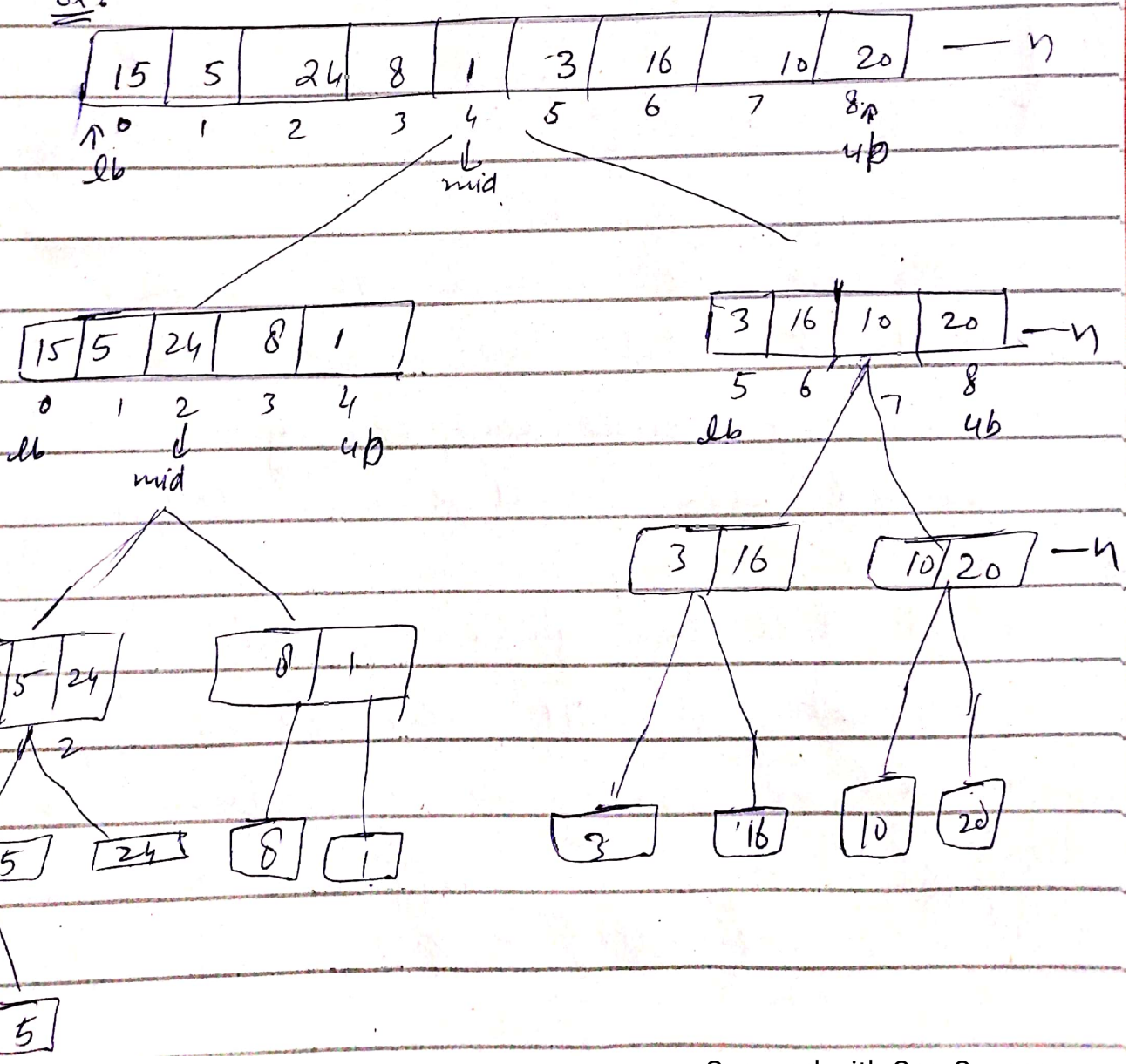
$O(n^2)$

Merge Sort : (DAC)

This sorting Algo works on divide and conquer technique.

⇒ Divide the big problem into sub-problems and solve it. At the end combine those small solutions into one big solution.

EX:



Algorithm :

```
mergesort (A, lb, ub)
{
    if (lb < ub)
    {
        mid = (lb + ub)/2 ;
        mergesort (A, lb, mid);
        mergesort (A, mid+1, ub);
        merge (A, lb, mid, ub)
    }
}
```

Time Complexity :-

⇒ because the problem is divided until it finds the single element.

⇒ So the time complexity of divided list would be $\log n$.

And while initially the whole list ~~was~~ would be traversed that would be n. So overall this would be

$O(n \log n)$.