Sorting

arrange elements in increasing order

→ Type of sorting method.

1) selection sort

In selection sort we just have to find minimum & put it in there right position.

ex.

0	1	2	3	4
4	6	5	3	2

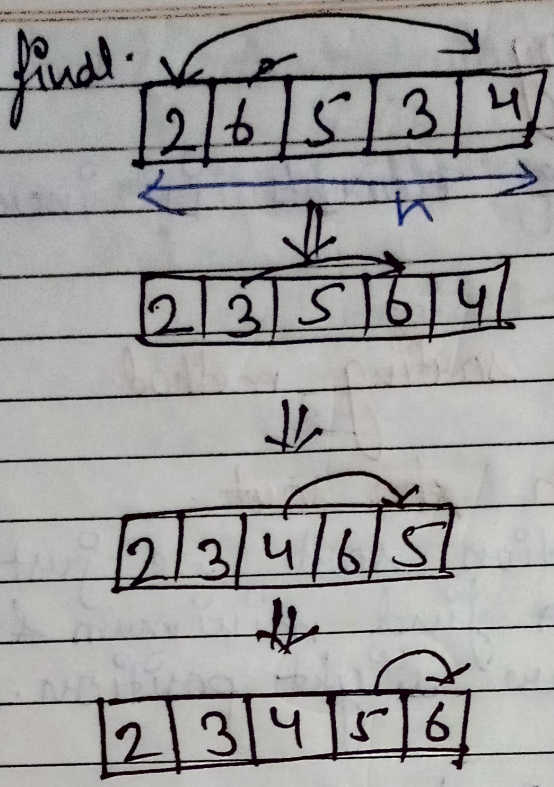
Now, there

1) Index 0 compare it's element to the rest of the index value in array.

2) and find minimum element from index 1 to last index.

3) if than compare it with the '0' index value if it is smaller than that.

4) then it swap the '0' index value to the minimum elements index value.



n = size of array

loop will run till (n-1).

→ main code

```
void selection(int a[], int size)
{
    for(int i=0; i<size-1; i++)
    {
        int minIndex = i;
        for(int j=i+1; j<size; j++)
        {
            if(a[minIndex] > a[j])
            {
                minIndex = j;
            }
        }
        swap(a[minIndex], a[i]);
```

sout code part — O(n)

O(n)

```
    }
}
```

print array part

```
void print(int a[], int size)
{
    for(int i=0; i<size; i++)
    {
        cout << a[i] << " ";
    }
}
```

main

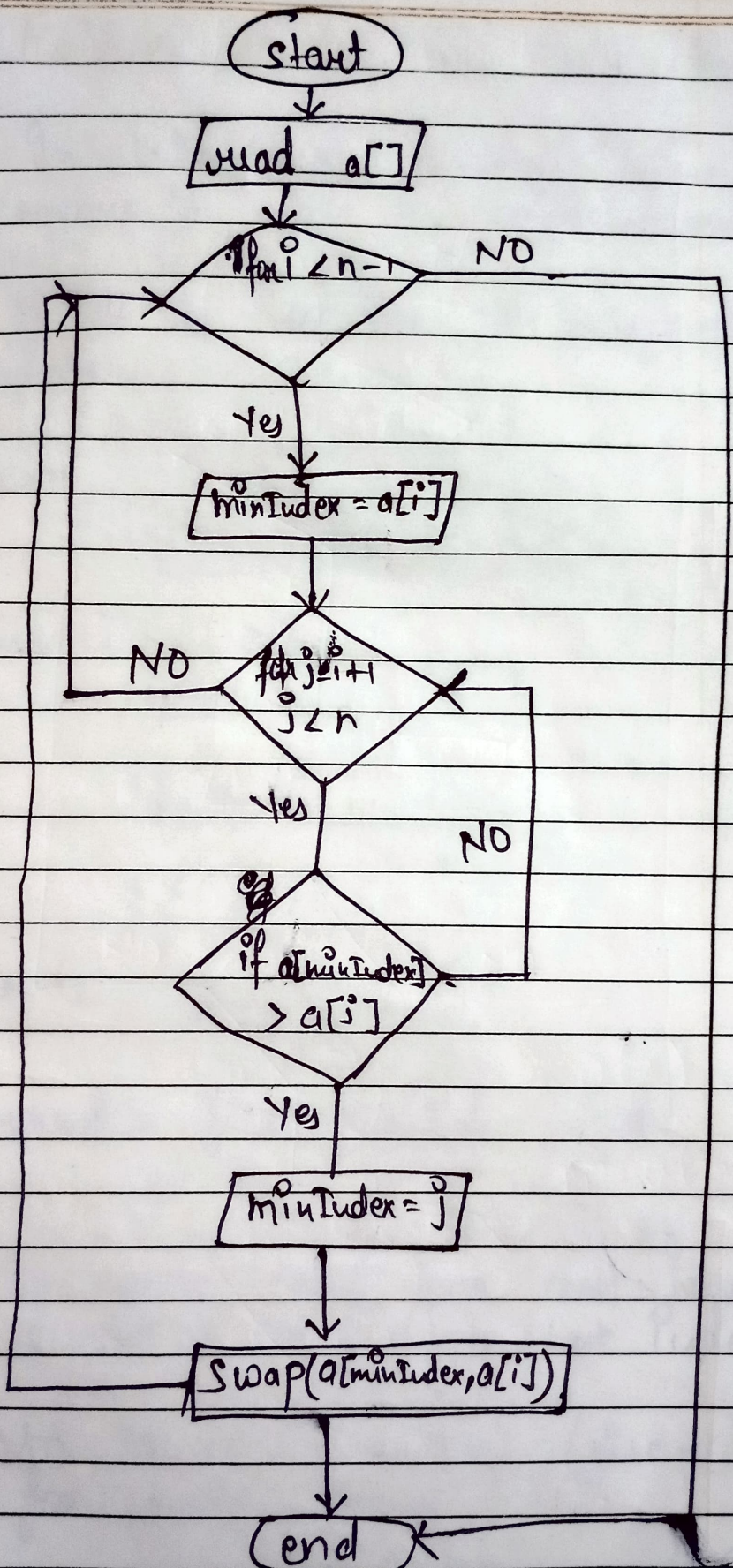
```
int main() {
    int a[7] = {4, 3, 6, 8, 7, 5, 2};
    selection(a, 7);
    print(a, 7);

return 0;
}
```

- Space complexity - O(1)
- Time complexity - $O(n^2)$

Best case - $O(n^2)$
Worst case - $O(n^2)$

* use when size of vector, array is small.

flow chart for selection sort



Bubble sort

In bubble sort every index element compare it with the next element.

if it is smaller it swap the numbers.

ex:-

0	1	2	3	4
11	8	6	4	14

round 1

8	11	6	4	14
8	6	11	4	14
8	6	4	11	14

round 2

6	4	8	11	14

round 3

4	6	8	11	14

final

In every round one most larger element is placed from last index.

So "Here" are total (size-1) round for every element.

→ main code

```
void Bubblesort(int a[], int size)
{
    for(int i=0; i<size-1; i++)
    {
        int swap1 = false;
        for(int j=0; j<size-i; j++)
        {
            if(a[j] > a[j+1])
            {
                swap(a[j], a[j+1]);
                swap1 = true;
            }
        }
        if(swap == false)
        {
            break;
        }
    }
}
```

for sorting

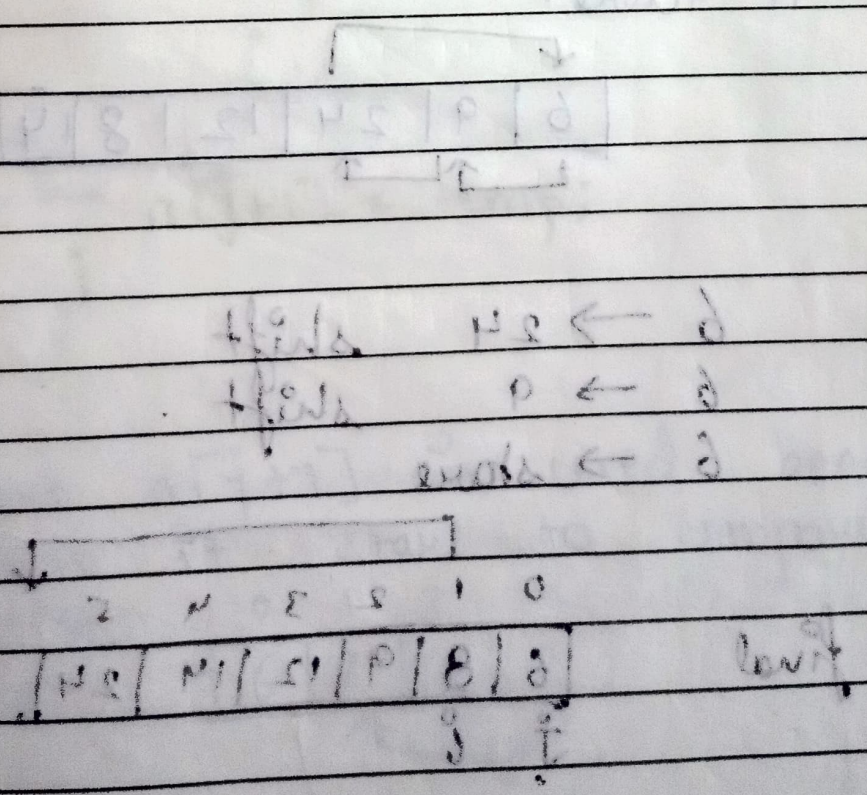
```
void print(int a[], int size)
{
    for(int i=0; i<size; i++)
    {
        cout << a[i] << " ";
    }
    cout << endl;
}
```

for print array

```
int main()
{
    int a[7] = { 4, 8, 6, 12, 36, 24
                 23};
    Bubblesort(a,7);
    print(a,7);

    return 0;
}
```

main function

→ Here in second loop we use (size - i)
as last element is set so we don't need to check these



Insertion sort

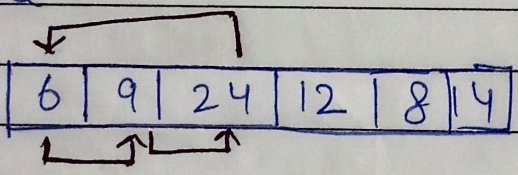
In insertion sort

9	24	6	12	8	14
0	1	2	3	4	5

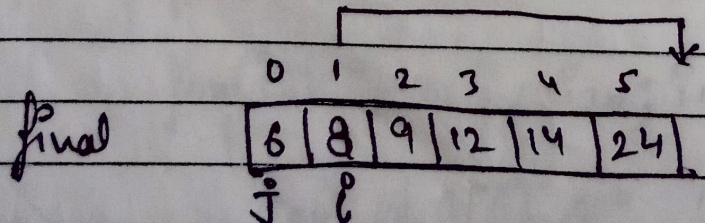
We let suppose that the starting element is a sorted / shorter.

temp = [0] index.

and take next index and compare to it if it is smaller than that we shift the current comparitor to next and move it there.



6 → 24 shift
6 → 9 shift.
6 → store



→ main code.

```
void insertionsort (int a[], int size)
{
  for(int i = 1; i<n; i++){
    int temp = i;
  int j= i-1;
   while(j>=0)
    {
      if(a[j] > temp)
      {
        a[j+1] = a[j];
      }else{

        break;
      }
    j--;
  }
(1)------ a[j+1] = temp;
  }
}
```

Time Complexity
B.C → O(n)
W.C → O(n²)

Here a[j+1] is used because when it start to compare back like

0	1	2	3
6	9	24	12

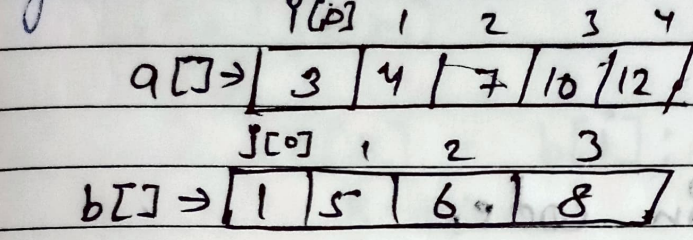
it change their position
we want it at index '2'
but it is on '1' right now
so we increase index by 1
so it will be on right position

→ it is stable a[i] = a[j]
→ it is adaptable
→ it is small

Merge sort on two array.

Merge sort

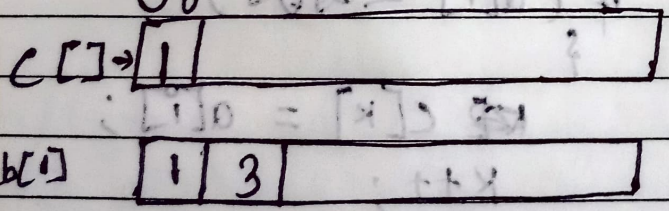
In Merge sort, we merge two sorted arrays in another array, which is also in sorted form.



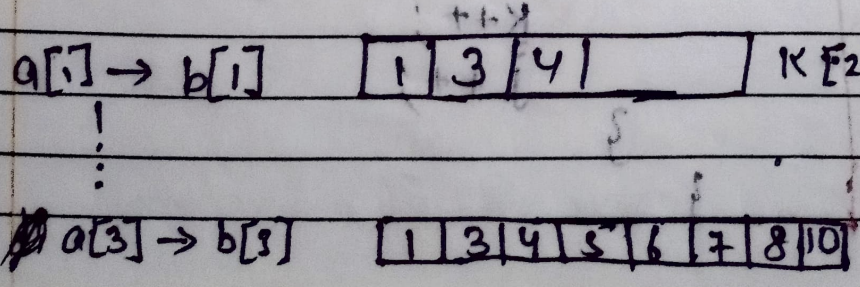
Here we compare

a[i] — b[j] at 0 index

which is smaller store in another array, 1 is smaller.



Here after storing the value we increament that particular array index.



and then copy the left index of 'a' in 'K' away. cuz no more element in b is left.

final

1	3	4	5	6	7	8	10	12	K[9]

→ main code:

```
void mergesort (int a[], int n, int b[], int m
, int c[])
  {
    int i=0; int j=0; int k=0;

    while (i<n && j<m)
    {
      if ( a[i] < b[j] )
      {
        c[k] = a[i];
        k++;
        i++;
      }else
      {
        c[k] = b[j];
        k++;
        j++;
      }
    }
```

when both array have same sizes,

PAGE NO.
DATE

when a[] is greater

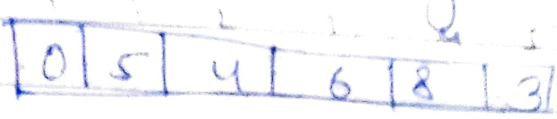
```
while (i < n)
{
    c[k] = a[i];
    k++;
    i++;
}
```

when b[] is greater

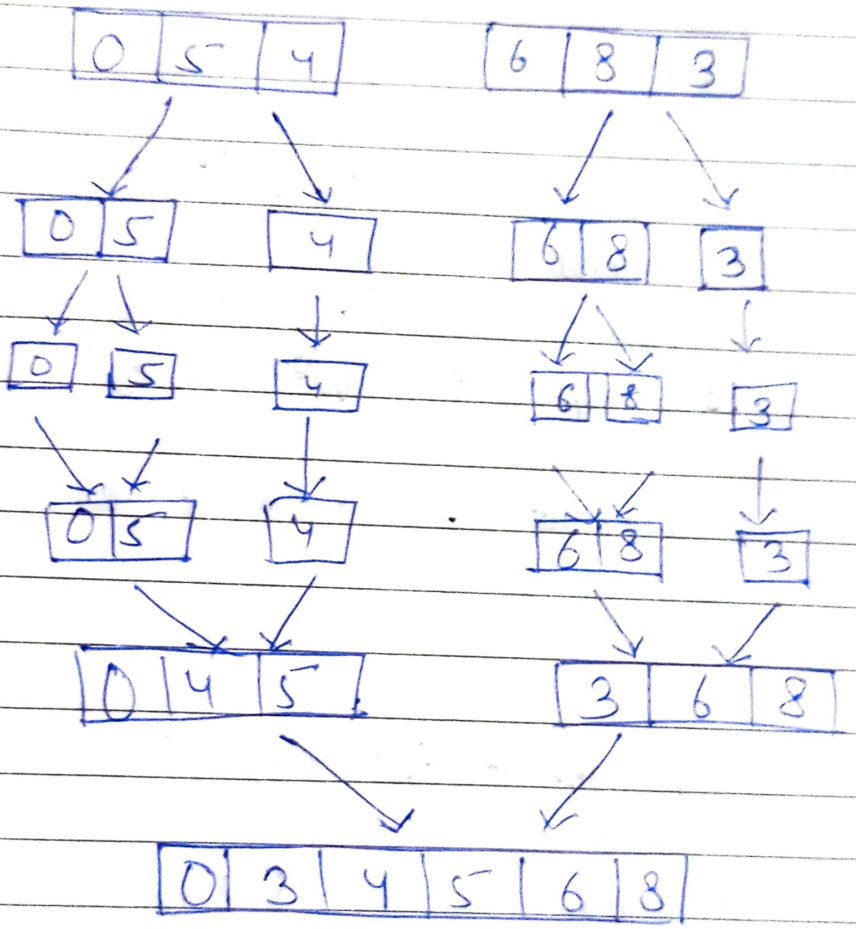
```
while (j < m)
{
    c[k] = b[j];
    k++;
    j++;
}
```

Merge sort

first break → arrange → merge.



find mid. $\frac{s+e}{2}$ → 2



→ Code

main fun

```
int main(){
    int arr[] = {4, 3, 2, 8, 5};
    arrange(arr, 0, 4);
    for(int i=0; i<5; i++){
        cout << arr[i] << " ";
    }
```

O(n) space complexity
O(n log n) time complexity.

```
return 0;
}

void merge(int *arr, int s, int e){

    // base case
    if(s >= e){
        return;
    }

    int mid = s + (e-s)/2;
    // process.
    merge(arr, s, mid);
    merge(arr, mid+1, e);

    mergesort(arr, mid, s, e);
}

void mergesort(int *arr, int mid, int s, int
{
    int len1 = mid - s + 1;
    int len2 = e - mid;
    int mainindex = s; int *firstarr = new int[
    int * secondarr = new int [len2];
    // coping.
    for(int i=0; i< len1; i++){
        firstarr[i] = arr[mainindex++];
    }
    for(int i=0; i< len2; i++){
```

O(n) recursion fun.

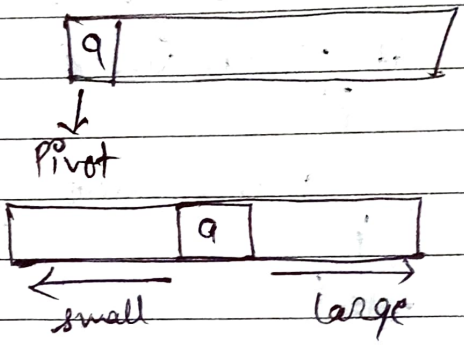
mainindex.mid

```
secondarr[i] = arr[mainindex++];
}

// sorting process.
index1 = 0;
index2 = 0;
mainindex = s;
while (index1 < len1 && index2 < len2){
    if (firstarr[index1] < secondarr[index2])
    {
        arr[mainindex++] = firstarr[index1++];
    }
    else {
    arr[mainindex++] = secondarr[index2++];
    }
}
    // when index1 < len1 but index2 is not < len2
while (index1 < len1){
 arr[mainindex++] = firstarr[index1++];
}
while (index2 < len2){
  arr[mainindex++] = secondarr[index2++];
}

}

// when index2 < len2 but index1 is not < len1 }
```

merge sort algo.

→ Quick sort.

1) arr[0] = pivot.
2) Count number of element smaller than pivot.
3) put it in right position = S + Count.
4) put smaller element on left & larger element on right.



5) further use recursion.

→ code

```
int main(){
 int arr[] = {2, 5, 9, 6, 3};
 mergesort(arr, 0, 4);

 for(int i=0; i<5; i++){
    cout << arr[i];
 }

return 0;
}
```

~~void merge~~

SC = O(n)
TC = O(nlogn)
Worst case TC = O(n²)

```
void mergesort(int *arr, int s, int e){
    if(s >= e)
        return;

    // partition part.
    int p = partition(arr, s, e);

    // left part sort.
    mergesort(arr, s, p-1);
    // right part sort.
    mergesort(arr, p+1, e);

}

int partition(int *arr, int s, int e){
    int pivot = arr[s];
    int count = 0;
    for(int i=s+1; i<=e; i++){
        if(arr[i] < pivot)
            count++;
    }
    int index = s + count;
    swap(arr[index], pivot);

    // manage left and right
    int i=s, j=e;

    while(i < index && j > index){
        while(arr[i] < pivot){
            i++;
        }
    }
```

```
while ( arr[j] > arr[index] ) {
        j--;
    }

if ( i < index && j > index )
    {
    swap ( arr[i++] , arr[j--]);
    }
}
return index;
}
```