SORTING

↳ Arrange in an order.

→ Basic sorting Algorithm:

- Bubble sort
- selection sort
- Insertion sort
- counting sort

** Bubble Sort:

- Idea: large elements come to the end of array by swapping with adjacent elements.

eg. arr[] = {5, 4, 1, 3, 2} → n=5

5, 4, 1, 3, 2
4, 5, 1, 3, 2
4, 1, 5, 3, 2
4, 1, 3, 5, 2
4, 1, 3, 2, 5
0th turn
0 to n-2

4, 1, 3, 2, 5
1, 4, 3, 2, 5
1, 3, 4, 2, 5
1, 3, 2, 4, 5
1st turn
0 to n-3
yha swapping nhi karenge, qki last wala largest h, ye already hum nikal chuke h.

1, 3, 2, 4, 5
1, 3, 2, 4, 5
1, 2, 3, 4, 5
2nd turn
0 to n-4

1, 2, 3, 4, 5
1, 2, 3, 4, 5
3rd turn
0 to n-5

Pseudo code:

```
// for turns.
for(turns = 0 to n-2){
    // for comparing & swaping
    for(int j=0 to n-2-turns)
    {
        if (arr[j] > arr[j+1]){
            int temp = arr[j]
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}
```

code:

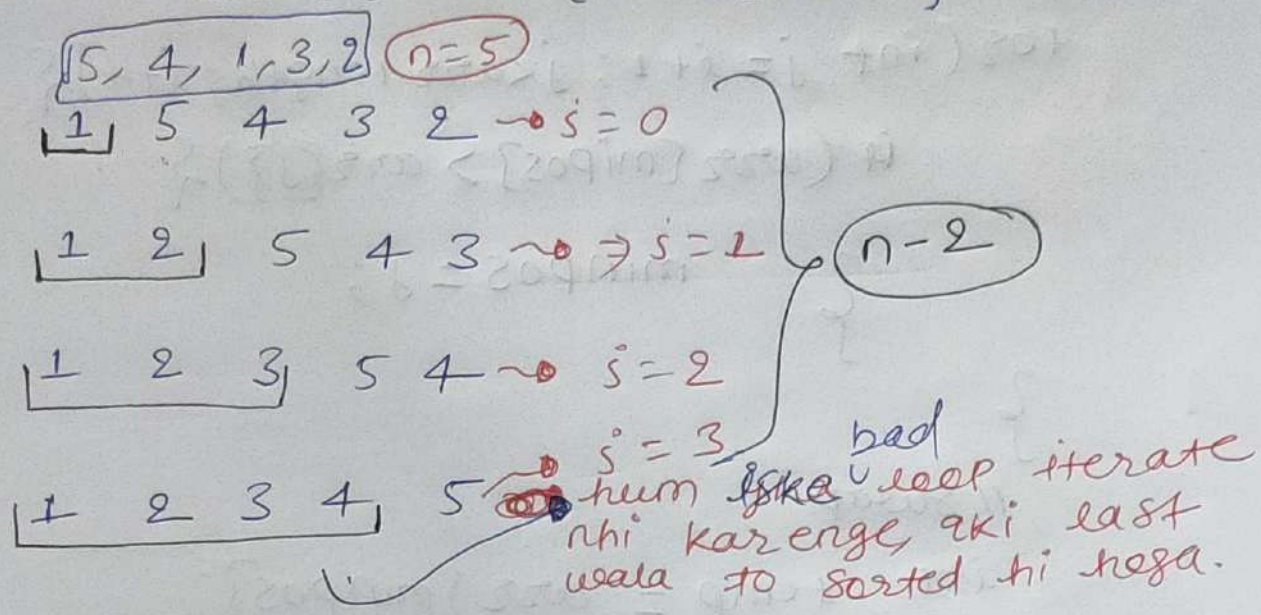
```
public static void main(String args[]){
    Scanner sc = new Scanner(System.in);
        int n = sc.nextInt;
    int arr[] = new int[n];
    for(int i=0; i<arr.length; i++){
        arr[i] = sc.nextInt();
    }
    for(int turns=0; turns<arr.length-1; turns++){
        int swap = 0;
        for(int j=0; j<arr.length-1-turns; j++){
            if(arr[j] > arr[j+1]){
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                swap++;
            }
        } if(swap==0){
            break;
        }
    }
```

```
for(int i=0; i<arr.length;
        i++){
    System.out.print(arr[i]+" ");
}
```

*) selection sort :

•) Idea : pick the smallest (from unsorted), put it at the beginning.

eg.- arr[ ] = {5, 4, 1, 3, 2}



pseudo code:

```
for(int i=0 to n-2)
{   minposition=i;          → max
    for(j=i+1 to n-1)       min
    {                       unsorted array → start
        if(minp.>arr[j])
        {
            minp. = j
        }
    }
    int temp = arr[minp];
    arr[minp] = arr[i];
    arr[i] = temp;
}
```

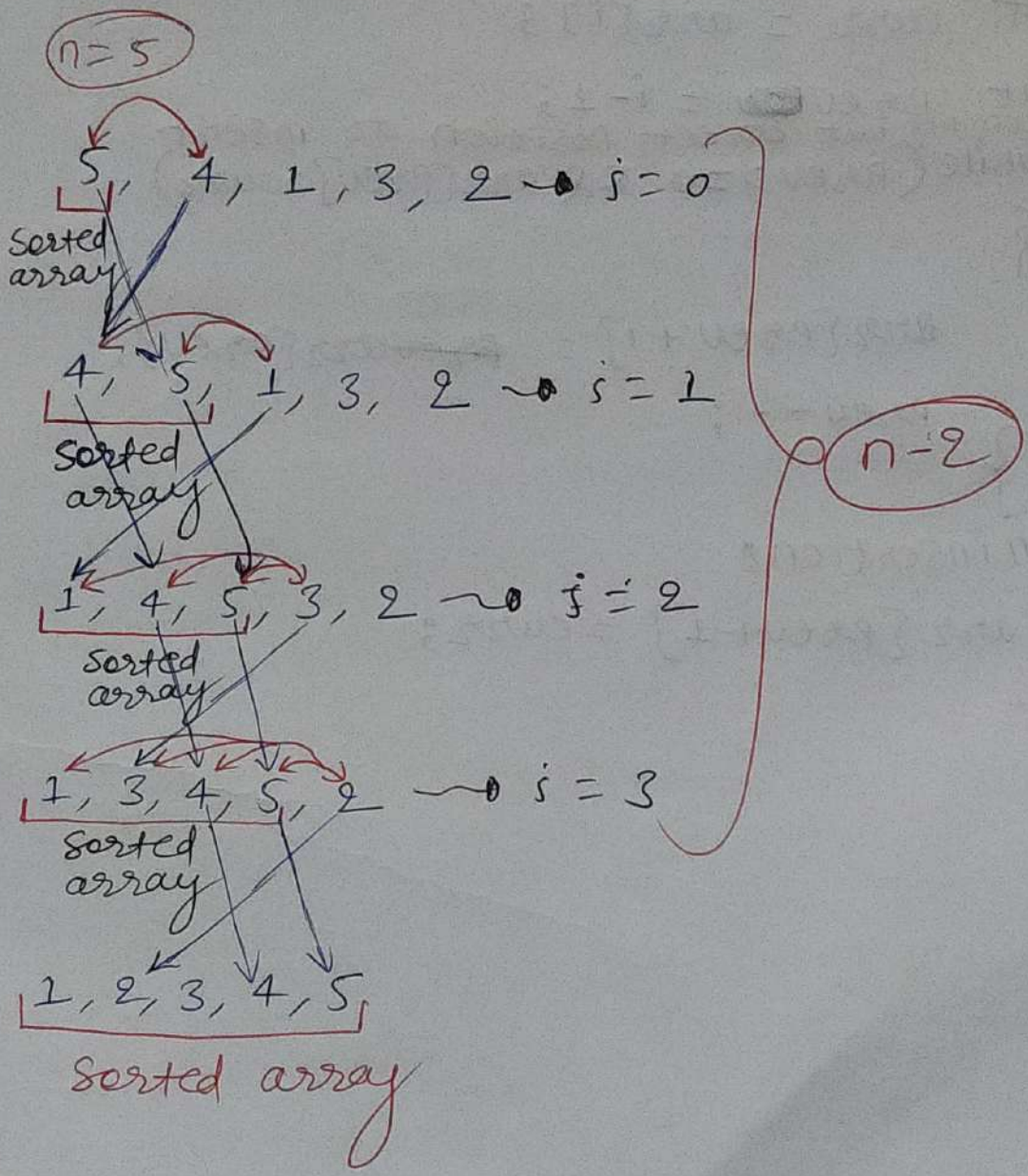
Actual code:

```
public static void selectionSort(int arr[]){
  for(int i = 0; i < arr.length-1; i++){
    int minpos = i
    for( int j= i+1; j<arr.length; j++){
      if(arr[mipos] > arr[j]){
        minpos = j;
      }
    }
    // swap
    int temp = arr[minpos];
    arr[minpos] = arr[i];
    arr[i] = temp;
}
```

*) Insertion sort:

•) Idea: pick an element (from unsorted array) & place in the right pos. in sorted array.

eg. arr[] = {5, 4, 1, 3, 2}



Actual dev. code

```
Public static void insertionSort(int arr[])
{
  for(int i=1; i<arr.length; i++){
    int curr = arr[i];
    int prev = i-1;
    //finding out correct position to insert
    while(prev>=0 && arr[prev]>curr)
    {
      arr[prev+1] = arr[prev];
      prev--;
    }
    //insertion
    arr[prev+1] = curr;
  }
}
```

*) Inbuilt Sort :

```
import. java.util.Arrays;
or, import java.util.*;

Arrays.sort(arr);
```

- bus ye function ko hume call karna hai, aur hamara array (arr) sort ho jayega. inc. order me
- Is function ki time complexicity approx (baki) sorting algorithm se kam hoti hai.
- O(nlogn);

*) Aur agar hume array ko kisi particular index se lekar kisi particular index tak sort karna hain to uske like :

```
Arrays.sort(arr, si, ei);
```

si → Starting index

ei → ending index

Ending index non-inclusive hota hain.

Starting index inclusive hota hain.

in dono ke bich me hamara array sort ho jayega.

⇒ for reverse order of sorting/decreasing order of sorting:

```
import java.util.Collections;
Arrays.sort(arr, Collections.reverseOrder());
```

Ye comparator ka use karta hain, dec. sort ke liye.

reverse order function object type variable ke upar kam karta hain, ye primitive ke upar kam nhi karta hai.

```
Arrays.sort(arr, si, ei, collection.reverseOrder());
```

*) counting sort :

- This sorting will be benificial to use for min. range.

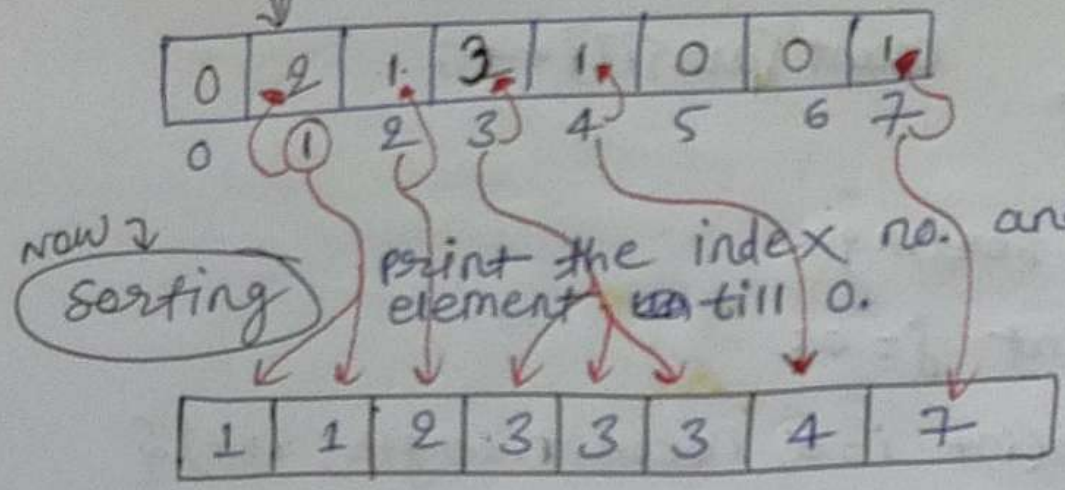
Ex:-

1, 3, 1, 3, 2, 4, 3, 7

max. no. → range = 7

make count array → frequency array

- size of this array is as range+1.

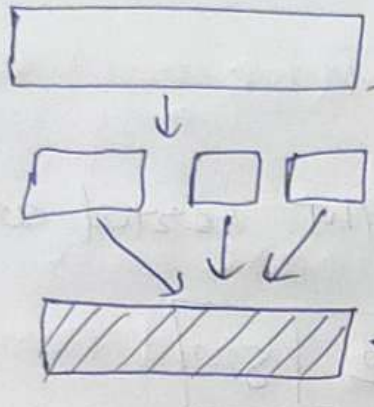


pseudo code

```
Public static void countingSort(int arr[]){
    int largest = Integer.MIN_VALUE;
    for(int i=0; i< arr.length; i++){
        largest = Math.max(largest, arr[i]);
    }
    int count[] = new int[largest+1];
    for(int i=0; i< arr.length; i++){
        count[arr[i]]++;
    }

    // sorting
    int j=0;
    for(int i=0; i<count.length; i++){
        while(count[i]>0){
            arr[j] = i;
            j++;
            count[i]--;
        }
    }
}
```

⇒ Divide & conquer Problem :



** merge sort :

unsorted

6	3	9	5	2	8

sorted

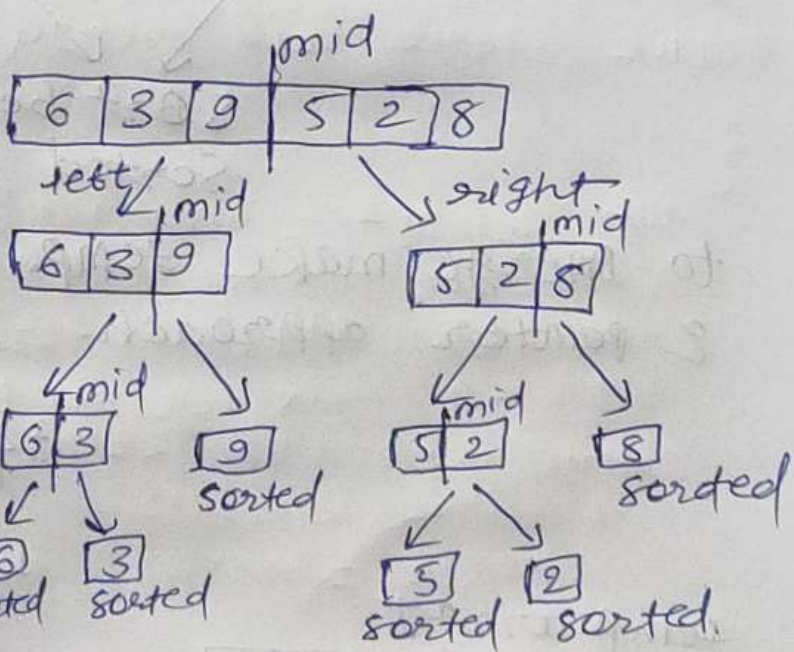
2	3	5	6	8	9

ascending order

- Approach :

① Divide
→ mid

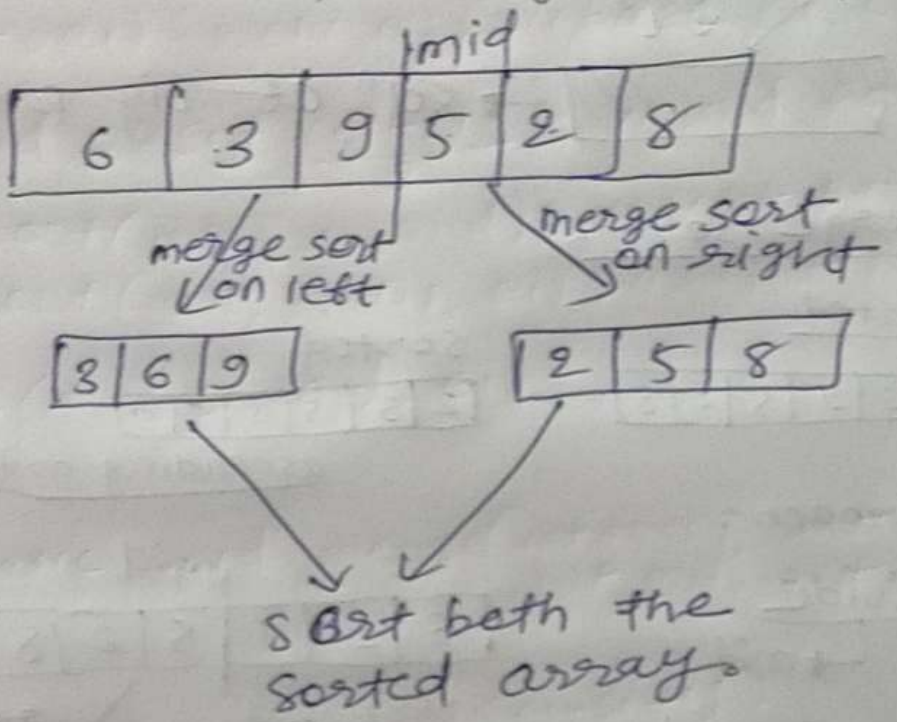
$mid = s + \frac{(e-s)}{2}$



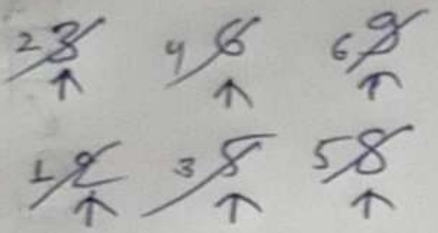
② Recursively :-

mergesort (left).
mergesort (right).

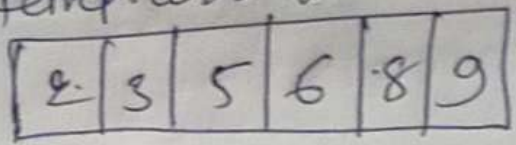
③ merge both left & right sorted array.



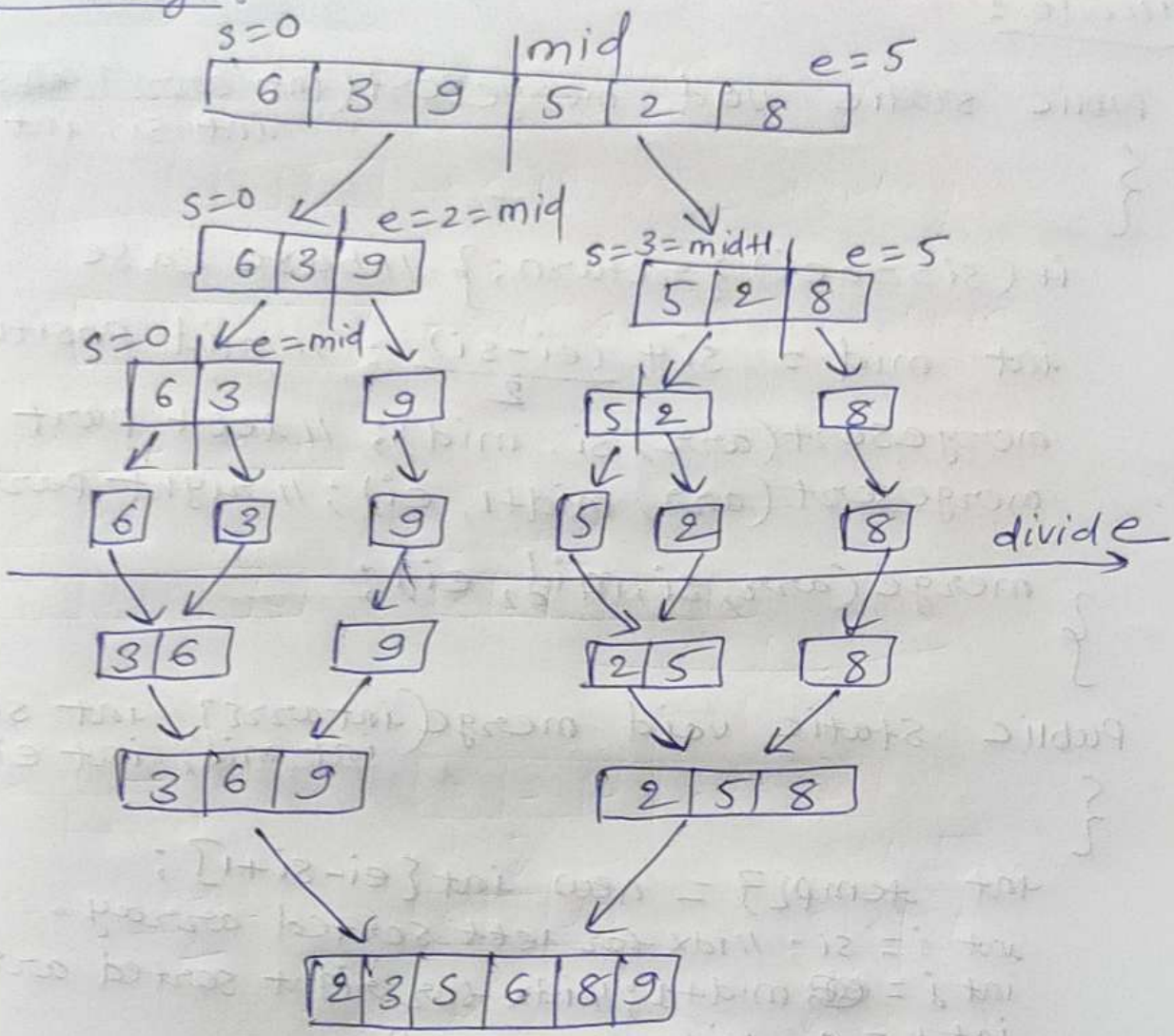
to merge make temp. array & use 2 pointer approach.



temp. array



summary:



Recursin:

① base case:
↳ (s > e or s == e)

② kam:
- divide
- mergesort(left)
- mergesort(right)
- merge

//code:

```
Public static void mergeSort(int arr[],
                             int si, int ei)
{
    if(si>=ei){return;} //base case
    int mid = si + (ei-si)/2; // mid position
    mergesort(arr, si, mid); // left part
    mergesort(arr, mid+1, ei); // right part
    merge(arr, si, mid, ei);
}
Public static void merge(int arr[], int si,
                         int mid, int ei)
{
    int temp[] = new int[ei-si+1];
    int i = si; //idx for left sorted array.
    int j = mid+1; //idx for right sorted array.
    int k = 0; //idx for temp.
    while(i<=mid && j<=ei){
      if(arr[i]<arr[j]){
        temp[k] = arr[i];
        i++;
      }else{
        temp[k] = arr[j];
        j++;
      }
      k++;
    }
```

```
//for leftover elements of 1st sorted part
while(i<=mid){
    temp[k++] = arr[i++];
}
//for leftover elements of 2nd sharted part
while(j<=ei){
    temp[k++] = arr[j++];
}
//copy temp to original array:
for(int i=si, k=0; k<temp.length; k++, i++){
    arr[i] = temp[k];
}
}
```

TC → nlogn

Space → O(n)

↑ because of temp array.

→ depth first sorting.

⇒ Quick sort :

- average → O(nlogn)
- worst → O(n²)
- space → O(1).

6	3	9	8	2	5

pivot & partition :

① Pivot → element on which quick sort will sort the array.

→ let's take 8 as pivot element.

lesser ele. than 8 ← 8 → greater element than 8.

2 3 5 6 8 9 → sorted

→ random, medium, first, last.

take last element 5 as pivot.

② partition : then partition :

along parts.

3, 2, < 5 < 6, 9, 8

(5 = pivot)

↓ quick sort

③ Quick sort(left)
Quick sort(Right)

Base (single element)

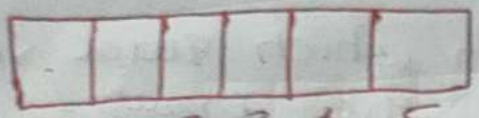
Partition :

si=0 ei=5

6	3	9	8	2	5

pivot = 5
pivot index = ei = 5

`i = -1` ← iterator for making places for elements which are lesser than pivots.



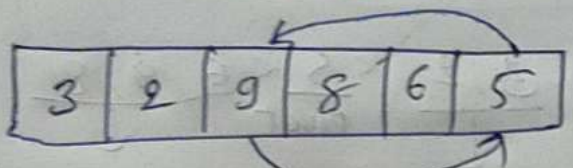
-1 0 1 2 3 4 5

`j = 0`

(6) (3) (9) (8) (2) (5)

↑ ↑
6>5 3<5
↓ j=1
↑ i++, and then swap ~~it~~ to ~~first~~ idx.
j=0 //swap -

```
int temp = arr[j]
arr[j] = arr[i+1]
arr[i+1] = temp;
```



3	2	9	8	6	5

swap one more time,

3	2	5	8	6	9

//code:

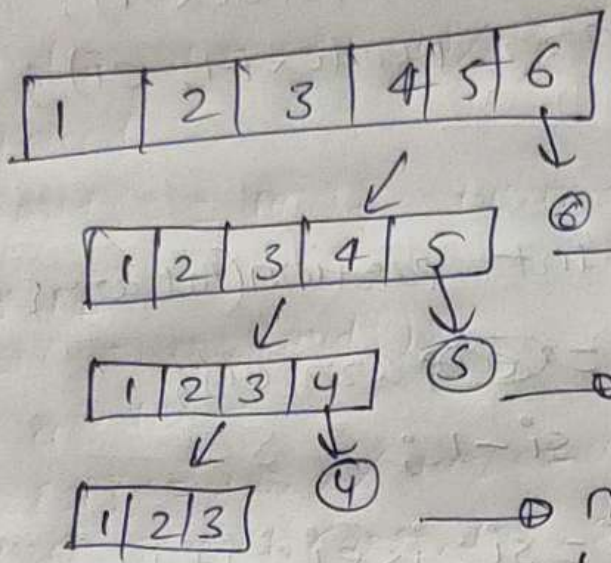
```
Public static void quicksort(int arr[], int si, int ei)
{
    if(si >= ei){ return; }
    int pIndex = partion(arr, si, ei);
    quicksort(arr, si, pIndex-1); //left
    quicksort(arr, pIndex+1, ei); //right
}

Public static int partion(int arr[], int si, int ei){
    int pivot = arr[ei];
    int i = si-1;
    for(int j=si; j<ei; j++){
        if(arr[j] <= pivot){
            i++;
            //swap
            int temp = arr[j];
            arr[j] = arr[i];
            arr[i] = temp;
        }
    }
    i++;
    int temp = pivot;
    arr[ei] = arr[i];
    arr[i] = temp;
    return i; // pivot index
}
```

*) worst case in quicksort :

•) worst case occurs when pivot is always the smallest or the laargest element.

Ex:-



$n + n-1 + n-2$

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

$\Rightarrow O(n^2)$.

*) Search in Rotated sorted array:

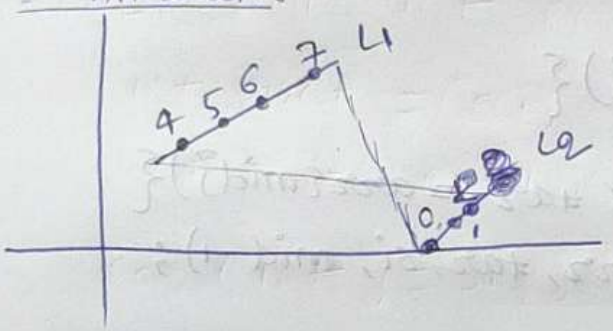
input: sorted, rotated array with distinct numbers (in ascending order) It is rotated at a pivot point. Find the index of given element.

4	5	6	7	0	1	2
0	1	2	3	4	5	6

target : 0

output = 4

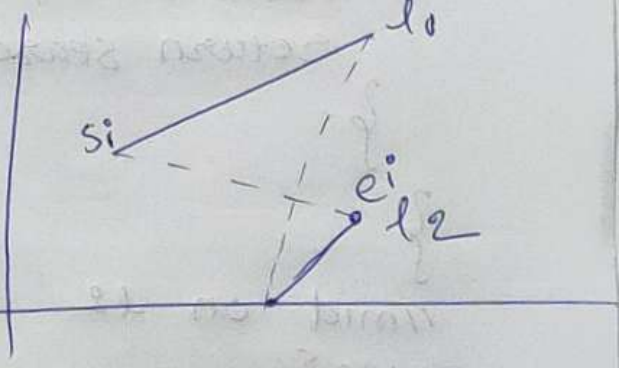
1st Approach:



(si<=target<=mid)
↓ left
↓ Right (false)

(mid<=target<=ei)
↓ ~~right~~
↓ left (false)

2nd Approach:



case-1 mid on l1

case-a: L1 left (si<=tar<=mid)

case-b: ~~L1~~ mid Right ↓ else

case-2 mid on l2 ← arr[mid]<=arr[ei]

case-c: L1 Right (mid<=tar<=ei)

case-d: mid left ↓ else

//code:

```
public static int search(int arr[], int tar, int si int ei)
{
  if(si>ei) return -1; //base case
  //kam
  int mid = si + (ei-si)/2;
  //case found
  if(arr[mid]==tar){return mid;}
  //mid on l1
  if(arr[si]<=arr[mid]){
    //case a: left
    if(arr[si]<=tar && tar<=arr[mid]){
      return search(arr, tar, si, mid-1);
    }else{
      //case b: Right
      return search(arr, tar, mid+1, ei);
    }
  }
  //mid on l2
  else{
    //case c: right
    if(arr[mid]<=tar && tar<=arr[ei]){
      return search(arr, tar, mid+1, ei);
    }else{
      //case d: left
      return search(arr, tar, si, mid-1);
    }
  }
}
```