Divide & Conquer

Merge Sort (nlogn) → T.C O(n) → S.C

```
Public static void printArr (int arr[]){
 for (int i=0; i<arr.length; i++){
     System.out.print (arr[i]+" ");
   }
    System.out.println();
 }
public static void mergeSort (int arr[], int si, int ei){
  if (si >= ei){
     return;
  }
   int mid = si + (ei -si)/2;   //(si+ei)/2
   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;  // iterator for left part
  int j = mid+1; // iterator for right part
  int K = 0; // iterator for temp arr
while (i<=mid && j<=ei){
    if (arr[i]<arr[j]){
        temp[k]=arr[i];
        i++;
    }else{
        temp[k]=arr[j];
        j++;
    }
    k++;
}
 // left part
 while (i<=mid){
     temp[k++]=arr[i++];
 }
 // right part
 while(j<=ei){
     temp[k++]= arr[j++];
 }
  // copy temp to original arr
 for(k=0, i=si ; k< temp.length ; k++, i++){
     arr[i] = temp[k];
   }
 }
```

main

```
int arr[] ={6, 3, 9, 5, 2, 8};
  mergeSort (arr, 0, arr.length-1);
  printArr(arr);
```

Quick Sort : $O(n\log n) \rightarrow$ average $\}$ T.C
$O(n^2) \rightarrow$ worst $\}$
pivot & partition
space $\rightarrow O(1)$

```
Public static void printArr(int arr[]){
    for(int i=0; i<arr.length; i++){
        Syso(arr[i]+" ");
    }
        Sysoln();
}
public static void quickSort(int arr[], int si, int ei){
    if(si >=ei){
        return;
    }
    //last element
    int pIdx = partition(arr, si, ei);
    quickSort(arr, si, pIdx-1);  // left
    quicksort(arr, pIdx+1, ei);  // right
}
public static int partition(int arr[], int si, int ei){
    int pivot = arr[ei];
    int i = si-1; // to make place for ele smaller than pivot

    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;
   returni;
 }
— main ———————————————
int arr[]= {6,3,9,8,2,5};
  quickSort (arr ,0, arr.length-1);
    printArr (arr);
 }
```

Worst case occurs when pivot is always the smallest or the largest element

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

target : 0

output : 4

```
public static int search (int arr[], int tar, int si, int ei){
  if (si > ei){
     return -1;
  }
   int mid = si + (ei-si)/2;  // (si+ei)/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);
      }

   }
 }
```

—— main ——

```
int arr[] = { 4,5 , 6, 7, 0,1, 2 };
 int  target  = 0;  // output = 4
  int  tarIdx = search (arr , target, 0, arr.length -1);
  sysoln ( tarIdx);
 }
}
```