Divide & Conquer

Algorithms :-

Divide :- Big problems are devided into small parts and solv

conquer :- converting small solution (into big) returning Answer

bubble sort
selection sort
insertion
counting } basic

Merge Sort

Time Complexity ⇒ n log n.

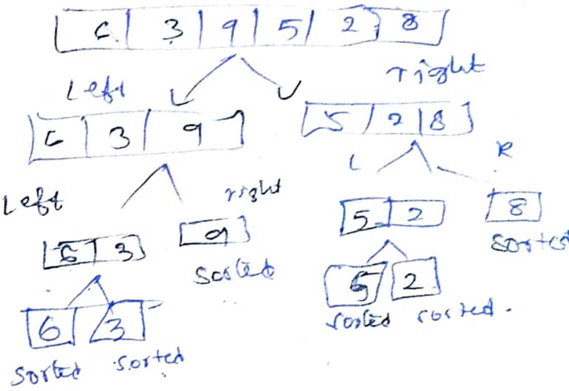
Sorting :- The process of arran
in particular order.

Unsorted

6	3	9	5	2	8

Sorted

2	3	5	6	8	9



Approach

① Divide ↳ mid

$mid = (si + ei)/2$

or

$\boxed{mid = si + (ei - si)/2}$

$si + [(ei - si)/2]$

$\frac{1 si}{1} + \frac{ei}{2} - \frac{si}{2}$

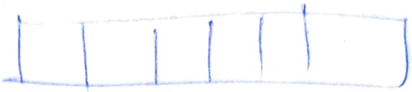
$\frac{ei}{2} + \frac{si}{2}$

$= \boxed{si + ei / 2}$

1) Divide ← mid (si+(ei-si))/2

2) merge sort (left)
merge sort (left)

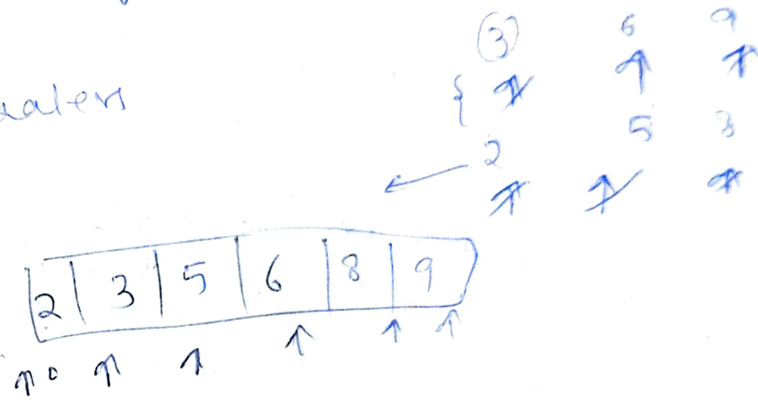
3) merge. →

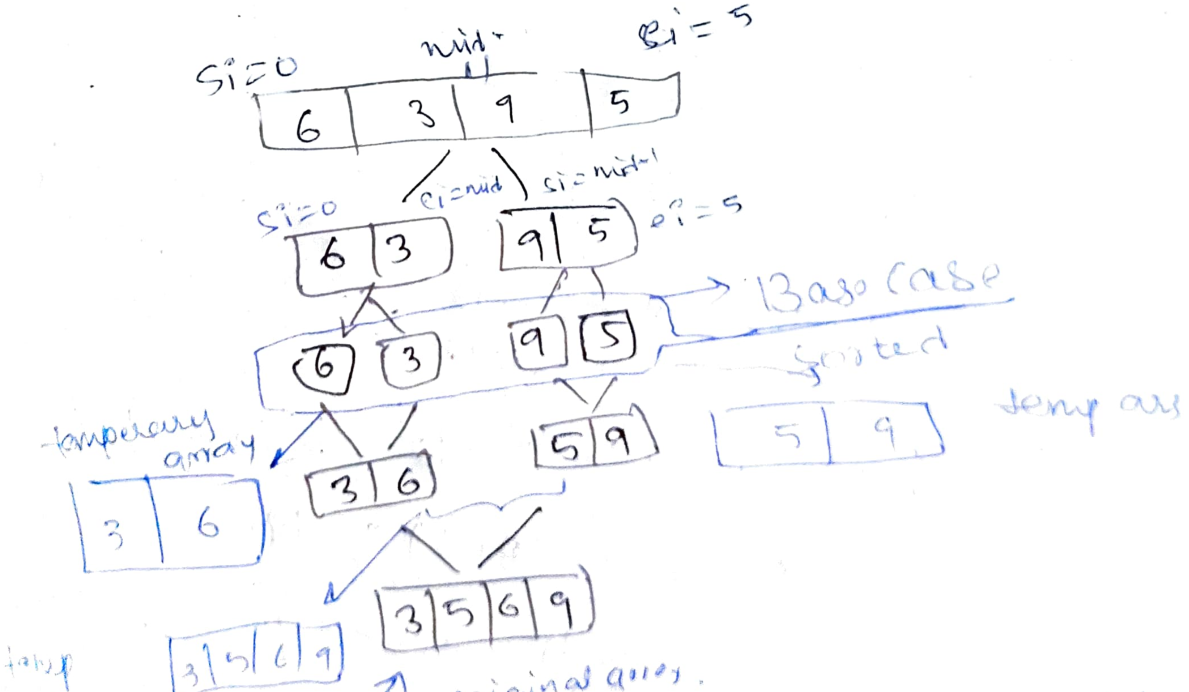


temporary array.

size = left + right elements

2 iterators





int temp[] = new int[ei-si+1];

Base case → si ≥ ei
si = ei

work → divide
merge sort (left)
merge sort (right)
merge
↓
merge

// Merge Sort

```
// for printing array:
public static void printArr (int arr[] ) {

 for (int i=0; i<arr.length ; i++) {
        syso (arr[i]+" ");

     }
    syso();

  }.

// merge sort Aproach.

public static void mergesort(int arr[], int si, int ei

   {
  // Base case.
   if (si ≥ ei) {
          return;
       }

   // work/implementation -
   // find mid ?
   // do recursion on leftside
   // do recursion on right side

  int mid = si+ (ei-si)/2  // si+ei/2.

  mergesort (arr, si, mid); //Left

  mergesort (arr, mid+1, ei); //right

// merging all the unsorted elements into another
                                              temp array.
 merge (arr, si, mid, ei);

}
```

```
// Temperary array
public static void merge(int arr[], int si, int mid,
                                   int ei) {

// left (0 → 3) = 4   Right (4 → 6) = 3 = (ei - si + 1)
       0,1,2,3               4,5,6        6 - 0 + 1 = 7
                                          3 + 4 = 7.

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 array

while (i ≤ mid && j ≤ ei) {

    if (arr[i] < arr[j]) { // if left value less
                              than right value
        temp[K] = arr[i];

        i++;  // or k++
    }                        // elements
                                 copied.
    else {
        temp[K] = arr[j];
            j++;  // or k++
    }
    k++;
}

// if some elements are present in left
   or right part
// eg:- Leftpart = 1 2 3 4   right = 5 6 7 8 !
first leftpart copies and right part remains
                     No for this anather
                          while condition
```

```
// Left part
while (i <= mid) {
    temp[k++] = arr[i++];
}
// right part
while (j <= ei) {
    temp [k++] = arr[j++];
}
// copy temp to original array

for (k=0, i=si , k< temp.length ; k++, i++) {
    {
      arr [i] =  temp[k] ;
      }
}

public static void main (string args[]) {
    int arr[] = {6,3,9,5,2,8};
    merge sort (arr, 0, arr.length-1);
                     si
    printArray();
}
```

Time Complexity = O(n logn)

Space Complexity O(n)

Depth first sort → merge sort

Quick Sort :-

→ average case = O(nlogn)

wo~~rst~~ case = O(n²)

Space complexity = O(1)

① pivot → random
median
~~first~~
Last

② partition.

③ Quicksort(left)
Quicksort(right)

6	3	9	8	2

pivot and partition.

↓

$\underbrace{6\ 3\ 2} < 8 < \underbrace{9}$

6	3	9	8	2	5

Pivot = 5 = ei.

i = -1

3	~~6~~ 2	9	8	6	5
			3	4	5

i=-1 0 1 2

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

swap.

i++;

//swap

```
{
int temp = arr[j]
arr[j] = arr[i])
arr[i] = temp;
```

swap pivot element.

3	2	5	8	6	9

Code :-

```
public class Quicksort {

    public static void printarr (int arr[]) {
    for (int i=0 ; i< arr.length. ; i++) {
        sysO ( arr [i] ) ;
    }

public static void Quicksort (int arr[], int si, intei
    {
    // base case
    if ( ei <= si ) {
        return ;
        }

int pidx = partition (arr, si, ei);
    Quick sort (arr, si, pIdx -1) ; // Left part
    Quick sort ( arr, pIdx+1 , ei ) ; // right

}
    public static int partition (int arr[], int si, int ei) {
public static int
        int pivot = arr [ei] ;
        int i = si-1 ; // to make place for elemen
                                smaller than pivot
                   si  j < ei ; j++)
    for (int j =   47 pivot) {
if (arr[i] <= pivot) { i++;
        int temp = arr [j] ;
        arr [j] = arr [i] ;
        arr [i] = temp ;
        }
// to make pivot at that place
    i++;
  int temp = pivot;
   arr[ei] = arr[i];   // pivot = arr[i] X
   arr[i] = temp;          pivot = variable
   return i;               variables changes
                           does not reflect in
                           functions
                           call by reference

public static void main (String args[]) {
   int arr[] = {1, 4, 6, 2, 8) $
     printarr (arr);
  }
}
```

Worst case Important

worst case ~~occurs~~ occurs when pivot is always the smallest or Largest element.

[1 | 2 | 3 | (4)] Largest

Ascending order & sorted only.

[1 | 2 | 3 | 4] (4) (n-1)

[1 | 2 | 3] 3 (n-2)

[1 | 2] 2 (n-3)

[1] (n-4)

$(n + n-1) + (n-2) + \ldots 3+2+1)$

$= AP.$

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

Search in rotated Sorted Array.

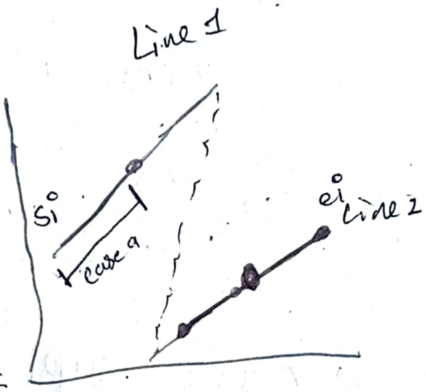
Input :- Sorted, rotated array with distinct numbers (in ascending order). it is rotated at a pivot point. find the index of given element.

[4 | 5 | 6 | 9 | 2] target = 9

output = 4.

(1, 2, 3 (4), 5, 6)
pivot

(5, 6, 1, 2, 3, 4)



case 1 : mid on L1
arr[si] <= mid.

case a : L1 Left (si ≤ tar ≤ mid)
↓
case b : ~~L1~~ right   else
mid

case 2 : mid on L2, arr[mid] <= arr[ei]

case c : L1 right (mid ≤ tar ≤ ei)

case d :

Aproach

1) if si > ei return -1 (Base case)

2) if (mid) = target → return it

3) calculate mid.

(A) → ~~ca~~ mid on Left
case A : si → (m-1)
case b : mid+1 → ei

(B) → mid on right
case c : mid+1 to ei
case d : mid-1 to si
vice verse
si to mid-1

4) main function f call ()

```
public class Divide &Conqu.
  public static int Search (int arr[], int tar, int si, int ei)
     {
       //Base case
        if ( si > ei) { return -1 };
          int mid = (si + ei)/2;
       // case found.
        if (arr [mid] == tar) {
                return mid;
        }

       // mid on Line 1

      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 Line 2.
      else { case c : right
         if (arr[mid] ≤ tar && tal ≤ arr[ei]) {
       return Search (arr, tar, mid+1, ei) }
        else { case d : left
          return Search (arr, tar, si, mid-1);

        }
      }}
Public c v m s A {
   int arr[]= {4,5,6,7,0,1,2};
   int target = 0;
   int tarIdx = search ( arr, target, 0, arr.length-1)
       syso (tarIdx);
```

Question-1:-

Apply mergesort to sort an array of strings (Assume that all the characters in all the strings are in lowercase) (EASY)

```
public static String[] mergesoot (String arr[], int lo, int hi
  {
   if (lo == hi) {
    string [] A = {arr[lo]};
     returnA;
   }
  int mid = lo+ (hi-lo) /2;
  string[] arr1 = merge sort (arr, lo, mid);
 String [] arr2 = mergesort (arr, mid+1, hi );

  String[] arr3 = merge (arr1, arr2);
   return arr3;
 }
Static string[] merge (string[] arr1, string[] arr2)
  {
   int m = arr1.length;
   int n = arr2.length;
   String[] arr3 = newstring [n+m];

    int idx = 0
     int i = 0, j = 0;
  while (i<m && j<n) {
    if (isAlphabetic (arr1[i], arr2[j]) {
       arr3[idx] = arr1[i];
         i++; idx ++;
else {
  arr3 [idx ] = arr2 [j];
       j++;  idx++;
    }
 while ( i<m) {
    arr3 [idx ] = arr1[i];
       i++; idx++;

 while ( j<n ) {
    arr3 [idx] = arr2[j];
     }
  return arr3;

static boolean isAlphabetic (String str1, Stg str2)
       if (str1. CompareTo (str2) <0) {
             return true;
       }
       return false;
   }
  public static void main (String [] args)
     {
     String [] arr = { "an", "tn", "cn" };
     String[] a = mergesort (arr, 0, arr.length-1)
     for (int i=0; i<a.length; i++) {
          syso (arr[i])
     }
   }
 }
```

Q-2

Count the numbers in array and return most repeated number

```
Public static int majorityelenet (int num[]) {
    int majorcount = nums.length/2;
    for (int i=0; i< num.length ;, i++) {
        int count = 0;
        for (int j=0; j < num. length ; j++) {
            if ( nums [j] == nums [i] {
                count += 1;
            }
        }
        if (count > majority count) {
            return nums [i];
        }
    }
    return -1;
public static void main String args[] {
    int nums[] = {2, 2, 1, 2, 2, 1};
    system.out.print (majorityelnt (nums));
    }
}
```

(or)

```
public static int CountinRange (int nums [], int num,
                                  int lo, int hi)
{
   int count = 0 ;
   for (int i = lo ; i <= hi ; i++ ; ) {
       if (nums[i] = = num) {
       count ++ ;
       }
   }
   return count ;
}
public static int major Emt recursion (int nums [], int lo,
                                          int hi)
{
   // base case . if only one element is there of size 1
       is the majority element
   if (lo == hi) {
       return nums [lo] ;
   }
   // recursion on left & right side .
   int mid = (hi + lo) / 2 ;
   int left  = majorityEmt recursion ( nums , lo, mid) ;
   int right = major    "      "  (nums , mid+1, hi) ;

   // if two halfs are on the majority . return it
       if (left == right)
       return left ;
       }
```

```
int leftcount = Count in Range( nums , left , lo , hi);
int right count =   "    "   ( nums, right , lo , hi);

return leftcount > right count ? left : right;

}
public static int majority elmt ( int [] nums){
return majorityEntrecursion ( nums , 0 , nums. length-1);

}
Public static void main ( string args [] ) {
        int nums [] = {2, 2, 1, 2, 1, 3 };
     syso ( major Elent ( nums));
        }
     }
```

Question-3 :-

Given an integer array find the inversion count in the array. (HARD)

Aproach:-

- Traverse through the array from start to end
- For every element, find count of elements smaller than the current up to the index using anathor loop
- Sum up the count of inversion for every index
- print the count of inversions.

code

```
public static int get inverseCount (int arr[]){
      int n= arr.length;
      int inversioncount =0;

      for (int i=0 ; i< n-1 ; j++){
          for (int j = i+1; j <n; j++){
              if (arr[i] > arr[j]){
                  inversion count ++;
              }
          }
      }

psvmsA[] {
   int arr[]= {1,20, 6,4,5};
   syso (getInverse count (arr));
}
```