Sorting

Bubble Sort : large element come to end of array by swapping with adjacent element

```
for (int turn = 0; turn < arr.length; i++) {
    for (int j = 0; j < Arr.length - 1 - turn; j++) {
        if (arr[j] > arr[j+1]) {
            int temp = arr[j]
            arr[j+1] = arr[j+1]
            arr[j+1] = temp
        }
    }
}
```

Best & Worst time complexity : $O(n^2)$

Selection Sort : select the smallest & put it in the front.

```
for (int i=0; i< arr.length-1 ; i++) {
    int minPos = i;
    for ( int j= i+1; j< arr. length ; j++) {
        if (arr[minPos] > arr[j]) {
            minPos = j;
        }
    }
    int temp = arr [minPos];
    arr [minPos] = arr [i];
    arr [i] = temp;
}
```

Best & worst case : $O(n^2)$

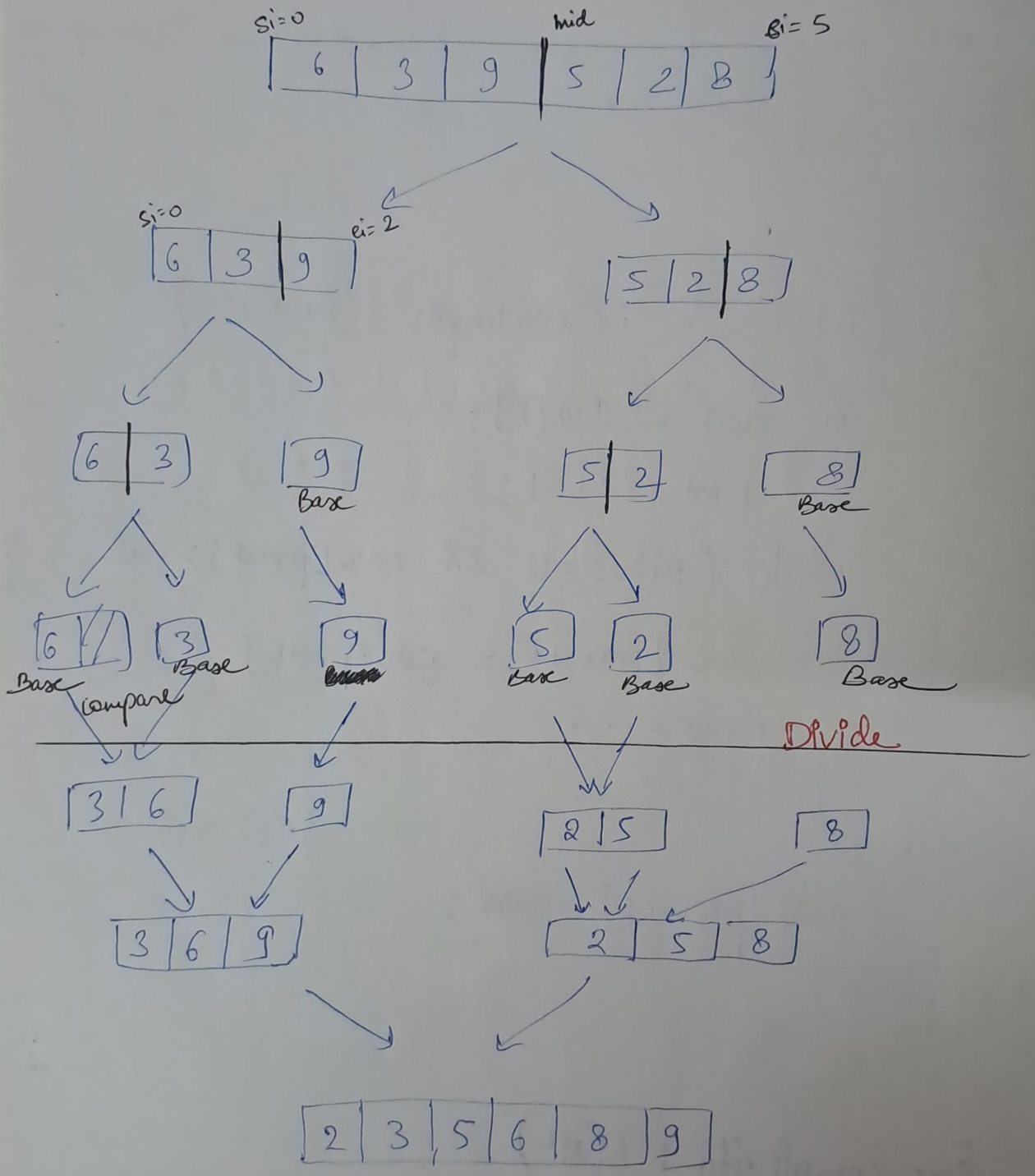
Insertion sort : Pick an element (from unsorted) and insert it in right position (sorted part) isme peeche ke saare elements se compare karte hain.

```
for (int i = 1; i < arr.length; i++) {
    int curr = arr[i];
    prev = i - 1;
    while (prev >= 0 && arr[prev] > curr) {
        arr[prev + 1] = arr[prev];
        prev--;
    }
    arr[prev + 1] = curr;
}
```

Time complexity : $O(n^2)$

Merge Sort

Divide & Conquer



Base cases of Recursion (Merge sort)-

1) $si > ei$ (invalid)
   $si = ei$ (single element)

Steps for recursion-

1) divide
2) Merge sort (left) } call for inner fn
   (right) }
3) Merge (temp array)

CODE

PS void

Merge Sort Code

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

PS void mergeSort (int arr[], int si, int ei) {
Here  int mid = si + (ei-si)/2      //(si-ei)/2
    mergeSort (arr, 0, mid);        //left
    mergeSort (arr, mid+1, arr.length-1); //right
    merge (arr, si, mid, ei);       //merging them
    if (si >= ei) {
        return;
    }

ps void printArr (int arr[]) {
    for (int i=0; i<=arr.length; i++)
    { cout
    }
```

PTO

```
PS  void merge(int arr[], int si, int mid, int ei) {
    int temp[] = new int [ei-si+1];
    int i = si;                    // for left part
    int j = mid+1;                 // for right part
    int k = 0;                     // for temp array

    while (i <= mid && j <= ei) {
        if (arr[i] < arr[j]) {
            temp[k] = arr[i];      // compairing
            i++; }                 // & putting them
        else {                     // in temp array
            temp[k] = arr[j];
            j++;
        }
        k++;
    while (i <= mid) {             // remaining
        temp[k++] = arr[i++];      // left part
    }
    while (j <= ei) {              // remaining right
        temp[k++] = arr[j++];      // part
    }
    for (k=0, i=si ; k <temp.length ; k++, i++) {
        arr[i]= temp[k];           // copy them to
    }                              // original array
```

Quick Sort

Pivot & Partition

Steps-

1) Pivot (last element)
2) Partition (parts)
3) Call quick sort (left)
   " " " (right)

Base case-

* si >= ei;

Code -

```
ps void main (String args[]) {
    int arr = { 6, 7, 1, 4, 5, 9 }
    quickSort (arr, 0, arr.length-1);
    printArray (arr);
}

ps void quickSort (int arr[], int si, int ei) {
  if (si>=ei){
    return;
  }
  int pIdx = partition (arr, si, ei);
  quickSort (arr, si, pIdx-1);      // left
  quickSort (arr, pIdx+1, ei);      // right
}
```

```
ps int partition (int arr[], int si, int ei) {
    int pivot = arr[ei];
    int i = si - 1;   // to make place for els smaller than pivot

    for (int j = si; j < ei; j++) {
        if (arr[j] <= pivot) {
            i++;
            int temp = arr[j];
            arr[j] = arr[i];
            arr[i] = arr[j];
        }
    }

    i++;
    int temp = pivot;
    wrong → pivot = arr[i];   arr[ei] = arr[i];
    arr[i] = temp;

    return i;
}
```

Search in Rotated Sorted Array

Input: sorted, rotated array with distinct nos
It is rotated at a pivot point. Find index of given element

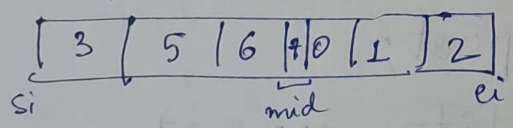
3	5	6	0	1	2

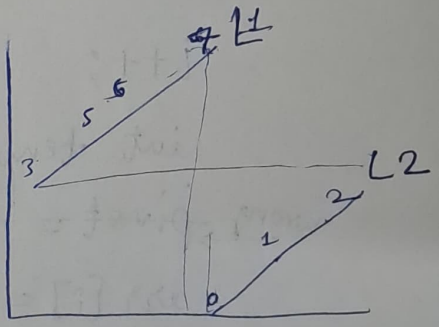
Output: 3

For this we will use

Modified binary Search

Approach: target = 0





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

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

case b: L1 right (else)

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

case c: L2 right [mid <= tar <= ei]

case d: L2 left (else

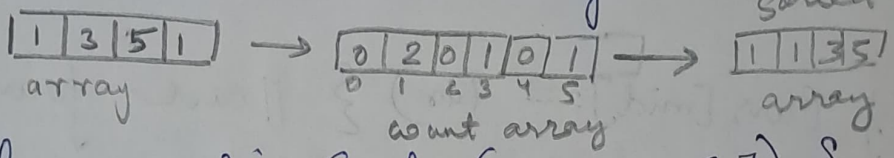
PS int search (int , int tar, int si, int ei)

```
{ // base case
  if (si > ei) {
      return -1;
  }

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

  if (arr[mid] == tar) { // case found
      return mid;
  }

  if (arr[si] <= arr[mid]) // mid on L1

L1 left if (arr[si] <= tar && tar <= arr[mid]) {
        return search (arr, tar, si, mid-1);
    }
L1 right else {
        return search (arr, tar, mid+1, ei);
    }
L2  }
  else { L2 right if (arr[mid] <= tar && tar <= arr[ei])
        { return search (arr, tar, mid+1, ei); }

L2 left else {
        return search (arr, tar, si, mid-1); }
  }
```

Counting Sort

$O(n + \text{largest})$

* first we count the freq of nos from 0 to 9 and store the freq in count array.
* Then we place (sort) nos in main array acc to count array.



Code -

```
ps void countingSort (int arr[]) {
    int largest = Integer.MIN_VALUE;
    for (int i=0; i< arr.length; i++) {          // for size of count array
        largest = Math.max (largest, arr[i]);
    }
    int count[] = new int [largest + 1];
    for (int i=0; i< arr.length; i++) {
        count [arr[i]] ++;                       // update that no
    }
    int j= 0;
    for (int i=0; i< count.length; i++) {
        while (count[i] >0) {
            arr[j] = i;                          // sorting
            j++;
    } } } count[i]--;
```