DIVIDE AND CONQUER TECHNIQUES

MERGE SORT

```
public class mergesort {

    //merge sort aproach
    public static void mergeSort(int arr[], int si, int ei){
        //base case
        if(si>=ei){
            return;
        }
        //implementation
    //find mid
    //do reccursion on left and right sides
        int mid=si+(ei-si)/2;
        mergeSort(arr, si, mid);
        mergeSort(arr, mid+1, ei);
        // merging or combining sorted array
        merge(arr,si,mid,ei);
    }

    public static void merge(int arr[],int si,int mid,int ei){
        // create temperary array
    int temp[]= new int[ei-si+1];// size = L(0->3)=4, R(4->6)=3 l+r=7 & ei-si+1=7
      int i= si;
      int j=mid+1;
      int k=0;
      while(i<=mid && j<=ei){
     if (arr[i]<arr[j]){// if left values are less than right side
        temp[k]=arr[i];// copy elemnets
        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 array[]
    for( k=0,i=si;k<temp.length;k++,i++){
        arr[i]=temp[k];

    }

   }
// for printing array
   public static void printarr(int arr[]){
    for(int i=0;i<arr.length;i++){
        System.out.print(arr[i]+" ");
    }}

   public static void main(String args[]){
    int arr[]= {1,6,4,3,5,-1};//array
    mergeSort(arr, 0,arr.length-1);//fcall
    printarr(arr);// fcall
   }

}
Output:
-1 1 3 4 5 6
```

QUICKSORT

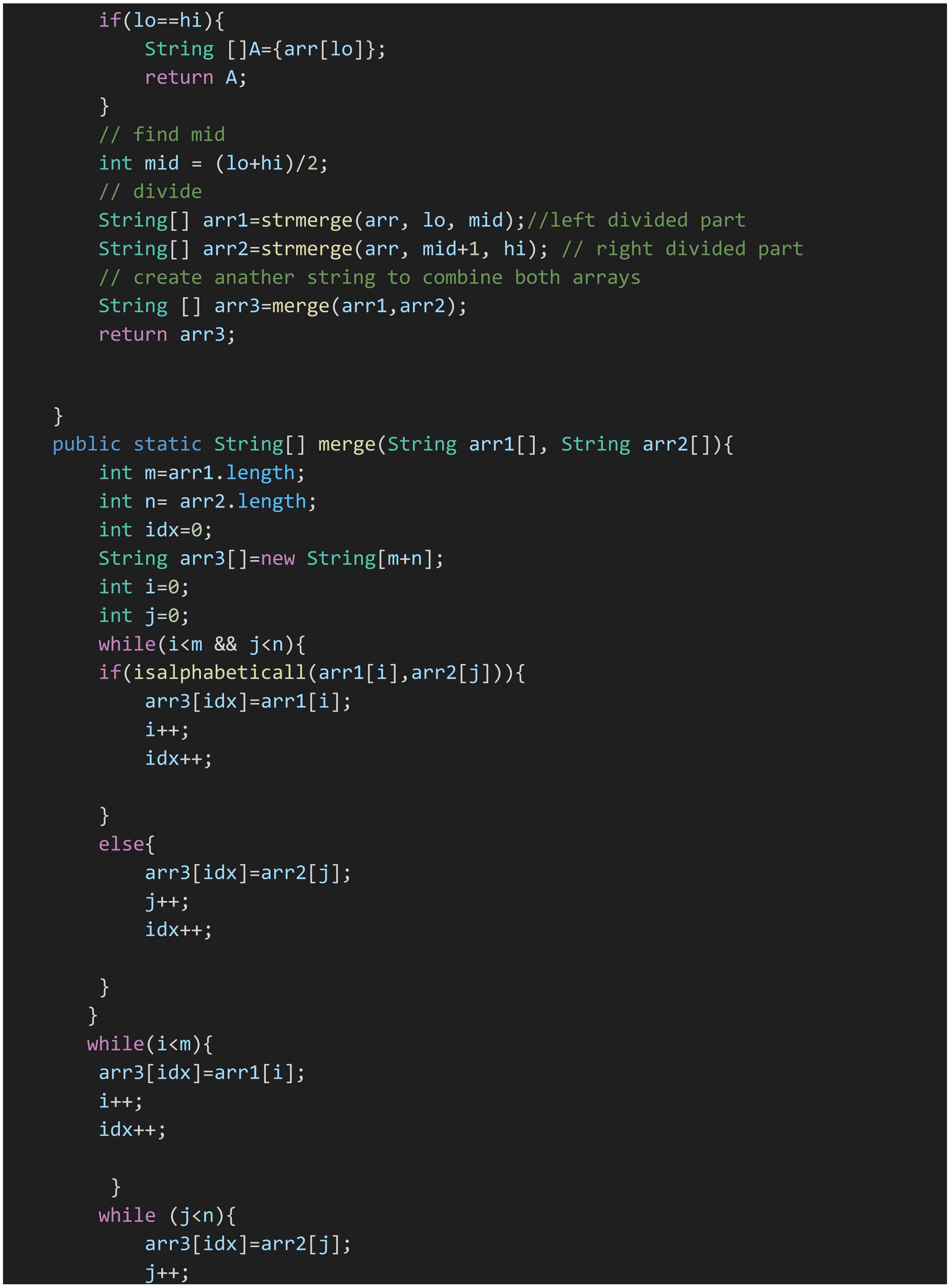
```
public class quicksort {
    //print array function
    public static void printarr(int arr[]){
        for(int i=0;i<arr.length;i++){
            System.out.print(arr[i]);
        }
    }
    // Quick sort
    public static void quickSort(int arr[],int si,int ei){
        //base case
        if(si>=ei){
            return;
        }
        // implementation
        int pIdx= partition(arr,si,ei);
        quickSort(arr,si,pIdx-1);//left part reccursion
        quickSort(arr, pIdx+1, ei);// right part reccursion
    }
    public static int partition(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++;
            //increment and swap
            int temp=arr[j];
            arr[j]=arr[i];
            arr[i]=temp;
            }
        }
        i++;
        int temp=pivot;
        arr[ei]=arr[i];// pivot=arr[i] here pivot = variable  variable values
does not reflect in functions (call by value)
        arr[i]=temp;
        return i;
    }

    public static void main(String args[]){
        int arr[]={2,1,3,4};
        quickSort(arr, 0, arr.length-1);
        printarr(arr);
    }



}
OUTPUT: 1234
```

Search in sorted array

```
import java.util.SortedMap;

public class searchinsortedarray {
    public static int search(int arr[], int tar, int si, int ei){
        // base case
        if(si>ei){
            return -1;
        }
        int mid=(si+ei)/2;
        // is found
        if(arr[mid]==tar){
            return mid;
          }
          //  mid on line 1
        if(arr[si]<=arr[mid]){
            if(arr[si]<= tar&& tar<=arr[mid]){
          // case a left
            return  search(arr, tar, si, mid-1);
            }
              else{// case b right
           return search(arr, tar, mid+1, ei);
            }}
       else {// mid on line 2

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

        }
    }
    public static void main(String args[]){
        int arr[]={4,5,6,7,0,1,2,3};
        int target=0;
        int pidx=search(arr, target, 0, arr.length-1);
        System.out.println(pidx);

    }
}
Output: 4
```

String mergesort:

```
public class stringmergesort {
    // merge sort aproach
    public static String[] strmerge(String arr[],int lo,int hi){
        // base case
```



```
        if(lo==hi){
            String []A={arr[lo]};
            return A;
        }
        // find mid
        int mid = (lo+hi)/2;
        // divide
        String[] arr1=strmerge(arr, lo, mid);//left divided part
        String[] arr2=strmerge(arr, mid+1, hi); // right divided part
        // create anather string to combine both arrays
        String [] arr3=merge(arr1,arr2);
        return arr3;


    }
    public static String[] merge(String arr1[], String arr2[]){
        int m=arr1.length;
        int n= arr2.length;
        int idx=0;
        String arr3[]=new String[m+n];
        int i=0;
        int j=0;
        while(i<m && j<n){
        if(isalphabeticall(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];
            j++;
            idx++;

        }
        return arr3;


    }
    static boolean isalphabeticall(String str1,String str2){
        if(str1.compareTo(str2)<0){
            return true;
        }

            return false;

    }


    public static void main(String args[]){
        String  arr[]={"ban","can","aan","tan"};
        String []a=strmerge(arr, 0, arr.length-1);
        for(int i=0;i<a.length;i++){
            System.out.print(a[i]+" ");
        }

    }
}
Output: aan ban can tan
```

Find most repeated element in array

```
public class printmostrepeatedinarray {
    public static int countinrange(int arr[], int num,int si,int ei){
        int count=0;
        for(int i=si;i<=ei;i++){
            if(arr[i]==num){
                count++;
            }
        }
        return count;
    }
    public static int majorelmtrecc(int arr[],  int si, int ei){
        // base case
        if(si==ei){
            return arr[si];
        }
        // mid
        int mid=(si+ei)/2;
        //recursion left and right
        int left =majorelmtrecc(arr,  si, mid);
        int right= majorelmtrecc(arr, mid+1, ei);
        // if two halfs are same
        if(left==right){
            return left;
        }
        //otherwise
        int leftcount=countinrange(arr,left,si,ei);
        int righttcount=countinrange(arr,right,si,ei);

        return leftcount>righttcount?left:right;//ternary
    }

        public static int majorelmt(int arr[]){
            return majorelmtrecc(arr, 0, arr.length-1);

        }
    public static void main(String args[]){
        int arr[]={2,2,1,2,3,2};
        System.out.println(majorelmt(arr));
    }

}
Output: 2
```

Find the inversion of array

```
public class inversecount {
    public static int getinversecount(int arr[]){
        int n= arr.length;
        int inversioncount=0;
        for(int i=0;i<n-1;i++){
            for(int j=i+1;j<n;j++){
                if(arr[i]>arr[j]){
                    inversioncount++;
                }
            }
        }
        return inversioncount;
    }
    public static void main(String args[]){
        int arr[]={2,4,1,3,5};
        System.out.println(getinversecount(arr));
    }

}
Output: 3
```