(5) Binary Search

```
class Binary Search Example {
public static binary Search(int arr[] .int first, int last, int key
  int mid =(first +last )/2;
  while (first < =last) {
    if (arr [mid]< key ) {
    first =mid +1;
    } else if (arr [mid] = =key) {
    system.out.print ("Element is found at index" +mid );
    break;
  }else {
    last =mid -1;
  }
  if (first >last ) {
    system .out. print("Element is not found ~~at index~~:
    break;
  }else {
    last =mid -1;
  }
  if (first >last ) {
      system.out.println("Element is not found !")
  }
}
public static void main (string args[]){
        int arr[] = {10, 20, 30, 40, 50};
        int key = 30;
        int last = arr.length-1;
         binary search(arr, 0, last, key);
  }
}
```

Binary Search Algorithm:

Warst case perfomance O(logn)

Best case perfomance O(1)

Averege case perfomance O(logn)

6. merge Sort

```
public class my merge Sort
{ void merge (int arr[], int beg, int mid, int end)
 { int l = mid - beg+1;
   int r = end - mid;
    in Left Array[] = New int [l];
    in Right Array[] = arr int [r];
```

```
for (int i =0; i<l; ++i]
Left Array [i] =arr [beg+i];

for (int j =0 j<r; ++j)
RightArray [j] = arr [mid +1 +j];

 int i=0, j=0;
 int k = beg;
 while (i<l&&j<r)
{
 if(Left Array[i] <= Right Array[j]
 { arr [k] = Left Array [i];
   i++
 }
  else
 { arr [k] =Right Array [j];
   j++;
 }
   k++;

}while (i<l)
{ arr [k] =Left Array [i];
i++;
k++;
}
 while(j<r)
 {
  arr[k] = Right Array [j];
  j++;
  k++;
 }
}
 void sort (int arr[], int beg, int end)
 {
  if(beg<end)
  {
   int mid =(beg+end)/2;
   sort (arr, beg, mid);
   sort (arr, mid+1, end);
   merge (arr, mid, end);
  }
 }
  public static void main (string args[])
 }
```

```
intarr[] ={90, 23, 101, 45, 65, 23, 67, 89, 34, 23}
my merge sort ob = new mergeSort();
ob.sort(arr, 0, arr.length -1);
System.out.println("\n Sorted arry");
for(int i=0; i<arr.length; i++)
{ system.out.print(arr[i]+"");
}
}
}
```

Time complexity of merge sort is O(n* logn) in all case

7. Quick Sort

```
public class QuickSort {
public static void main(string [] args){
int i;
int [] arr ={90, 23, 101, 45, 65, 23, 67,
          89, 34, 23};

quickSort(arr, 0, 9);
system.out.println("\n The sorted arry is; \n");
for (i=0, j<10; i++)
```

```
system.out.println(arr[i]);
}
public static int partition(int a[], int beg, int end)
{
 int left, right, temp, loc, flag;
 loc = left = beg;
 right = end;
 flag = 0;
 while(flag != 1)
 {
  while (a[loc] <= a[right] && (loc != right)
  right--;
  if (loc == right)
  {
   temp = a[loc];
   a[loc] = a[right];
   a[right] = temp;
   loc = right;
  }
   if (flag != 1)
  {
   while (a[loc] >= a[left] && (loc != left))
left++;
if(loc==left)
flag=1;
else if(a[loc]<a[left])
{
 temp=a[loc];

 a[loc]=a[left]
 a[left]=temp;
 loc=left;
 }
}
}
returnloc;
}
static void quickSort(inta[], int beg,int en
{
 intloc;
 If(beg<end)
 {
  loc=partition(a, beg, end);
  quick Sort(a, beg, loc-1);
  quick Sort(a, loc+1, end);
 }
}
}
```

Quick Sort Algorithm:

Worst case Perfomance O(n2)
Best case performance O(n)
Avarage case perfomance O(n logn)