```
import java.util.Arrays;
import java.util.Scanner;
/*Nhan Vo -CECS 328 Lab#4

*/
public class main {
   public static void main(String[] args){
       main m=new main();
       System.out.println("\n-----Test Cases-----\n");
       int[] a={10,4,2,15,18};
       int[] b={25,3,1,8,7,2,32};
       System.out.println("2 nearest element from the median of array:
"+Arrays.toString(a)+" is: "+Arrays.toString(m.FindCloset_K_ele(a,2))+"\n");
       System.out.println("4 nearest element from the median of array:
"+Arrays.toString(b)+" is: "+Arrays.toString(m.FindCloset_K_ele(b,4))+"\n");
       int min=-100;
       int max=100;
       System.out.println("\n-----User Interface-----\n");
       System.out.println("Please enter a positive interger:");
       Scanner scan=new Scanner(System.in);
       String n=scan.nextLine();
       while(!n.matches("\\d+")){
           System.out.println("Your input is not an approriate integer, Please
try again:");
           n = scan.nextLine();
       }
       int[] array=new int[Integer.parseInt(n)];
       for(int i=0;i<Integer.parseInt(n);i++){
           array[i]=(int)(Math.random()*(max-min+1)+min);
       }
       System.out.println("Generated Array: "+Arrays.toString(array));
       System.out.println("Please enter a number between 1 and "+n+":");
       String k=scan.nextLine();
       while(!k.matches("\\d+")){
           System.out.println("Your input is not an approriate integer, Please
try again:");
           k = scan.nextLine();
       }

System.out.println(Arrays.toString(m.FindCloset_K_ele(array,Integer.parseInt(k)
)));

       /*
       Answer Questions:

       Question #6: What is the time complexity of saving the differences from
median (a[i]-median) in a new array?
       Answer: The time complexity is O(n)
```

```java
       Explaination: When we find the differences between the median and each
number in the original array, we need to go one by one at a time.
       Hence, we need to go over the entire array, maybe except the median
since it difference would be 0. so we get T(n)=O(n-1) or T(n)=O(n)


       Question #7: What is the time complexity of shifting the found K numbers
back to their original value?
       Answer: The time complexity is constant T(n)=1
       Explanation: Since I created two array when doing the differences, one
stores the different value, and one store the index numbers that create those
differences.
       They will partition together in the partition function. When I got the
final array of K smallest differeces from the median,
       I just call array[index] to get the actual value of those numbers.


       Question #10: What is the total time complexity of the algorithm?
       Answer: From the time complexity calculation for each steps in the
m.FindCloset_K_ele algorithm,
       I got their sum of T(n)=O(n)
       */

   }
   /*
   Quick_select function:
   Take an array and a number as its arguments
   return the smallest k-th number in the list.
   */
   public int Quick_select(int[] a, int[]b,int k, boolean duplicate){
       int piDex=Partition(a,b,duplicate);
       //System.out.println(piDex);
       //System.out.println(Arrays.toString(a));
       //System.out.println(k);
       if (k-1==piDex){
           //System.out.println(a[piDex]);
           return a[piDex];
       }
       else if(k-1<piDex){

           return
Quick_select(Arrays.copyOfRange(a,0,piDex),Arrays.copyOfRange(b,0,piDex),k,dupl
icate);
       }

       else if(k-1>piDex){
           return
Quick_select(Arrays.copyOfRange(a,piDex+1,a.length),Arrays.copyOfRange(b,piDex+
1,a.length),k-piDex-1,duplicate);

       }
       return -1;
   }

   /*
   Partition function:
   Separate an array into two part that any of the left side will be smaller
than the right side
   */
   private int Partition(int[] a, int[]b,boolean duplicate){
       int len=a.length-1;
       int pivot = MedianofThree(a[0],a[(int)(a.length/2)],a[len]);
       //System.out.println(pivot);
       //System.out.println((int)(a.length/2));
       int left=0;
       int right=len;
       int tempNum;
       //System.out.println(Arrays.toString(a));
       while(left<right){

           if(duplicate==true) {
               while (a[left] <= pivot) {
                   left++;
                   if (left > right) {
                       break;
                   }
               }
           }
           else{
               while (a[left] < pivot) {
                   left++;
               }
           }

           while(a[right]>pivot){
               right--;
           }
           //System.out.println(left+" "+right);
           if(left>=right){
               break;
           }

           tempNum=a[left];
           a[left]=a[right];
           a[right]=tempNum;
           tempNum=b[left];
           b[left]=b[right];
           b[right]=tempNum;

           //System.out.println(Arrays.toString(a));
       }
       //System.out.println("Hi: "+Arrays.toString(a));
       return left;

   }

   /*
 Medianofthree function:
 Find the median from tree number
 */
   private int MedianofThree(int a, int b, int c){
       //System.out.println(a+" "+b+" "+c);
       if(a>=b & a<c){
           return a;
       }
       else if((b>=a & b<c)||(a==c & a>b)){
           return b;
       }
       else{
           return c;
       }
   }

   public int[] FindCloset_K_ele(int[] a, int k){
       int[] out=new int[k]; // T(n)=k
       int size= a.length; // T(n)=1
       int[] diff=new int[size-1]; // T(n)=n
       int[] arrDex=new int[size-1]; // T(n)=n
       int median=0; // T(n)=1
       int index=0; // T(n)=1
       int j=0; // T(n)=1
       if(size%2!=0){
           index=(size/2)+1;//T(n)=1
           median=Quick_select(a, new int[size],index,false); //T(n)= O(n)
       }
       else{
           index=(size/2);//T(n)=1
           median=Quick_select(a, new int[size],index,false); //T(n)= O(n)
       }
       //System.out.println(index);
       //System.out.println(median);
       Partition(a,new int[size],median); //T(n)= O(n)
       for(int i=0;i<size;i++){ //T(n)= O(n)

           if(Math.abs(median-a[i])!=0){
               diff[j]=Math.abs(median-a[i]);
               arrDex[j]=i;
           }
           if(Math.abs(median-a[i])!=0){
               j++;
           }

       }
       //System.out.println(Arrays.toString(a));
       //System.out.println(Arrays.toString(arrDex));
       //System.out.println(Arrays.toString(diff));

       int[] temp=Quick_selectMin(diff,arrDex,k);//T(n)=O(n)
       for(int i=0;i<k;i++){ //T(n)=k
           out[i]=a[temp[i]];
       }
       return out;

   }

   /*
   Modified Partition function
   Take an array and a pivot point as its arguments
   return the right array from the pivot
   */
   private int[] Partition(int[] a, int[]b, int n){
       int len=a.length-1;
       int pivot = n;
       //System.out.println(pivot);
       //System.out.println((int)(a.length/2));
       int left=0;
       int right=len;
       int tempNum;
       //System.out.println(Arrays.toString(a));
       while(left<right){

           while(a[left]<=pivot){
               left++;
               if(left>right){
                   break;
               }
           }

           while(a[right]>pivot){
               right--;
           }
```

```java
            //System.out.println(left+" "+right);
            if(left>=right){
                break;
            }

            tempNum=a[left];
            a[left]=a[right];
            a[right]=tempNum;


            tempNum=b[left];
            b[left]=b[right];
            b[right]=tempNum;


            //System.out.println(Arrays.toString(a));
        }
        return Arrays.copyOfRange(b,0,left);
    }

    /*
    Part B
    Quick_selectMax function:
    Take an array and a number as its arguments
    return the largest k numbers in the list.
    */
    public int[] Quick_selectMin(int[] a, int[]b, int k){
        int pivot= Quick_select(a,b,k,true);
        return Partition(a,b, pivot);
    }


}
```