1. Binary Search:

CODE:

```
import java.util.*;
class Main {
   public static int binarysearch(int a[], int x){
      int l=0,h=a.length-1;
      while(l<=h){
         int m=(l+h)/2;
         if(a[m]==x) return m;
         if(a[m]<x) l=m+1;
         else h=m-1;
      }
      return -1;
   }
   public static void main(String[] args) {
      int a[]={2,3,4,10,40};
      int n=a.length;
      int x=10;
      int r=binarysearch(a,x);
      if(r==-1) System.out.println("Target is not present");
      else System.out.println("Target is at the index:" +r);
   }
}
```

OUTPUT:

```
Target is at the index:3

=== Code Execution Successful ===
```

Time Complexity: O(log n)

2. Next Greater Element:

CODE:

```
import java.io.*;
import java.util.*;
class HelloWorld {
   public static void main(String[] args) {
      int arr[]={4,5,2,25};
      int arr2[]=nextgreater(arr);
      System.out.println("Next Greater Elements: ");
      for(int i=0;i<arr2.length;i++){
         System.out.print(arr2[i]+" ");
      }
   }
   public static int[] nextgreater(int[] arr){
      int n=arr.length;
      int nge[]=new int[n];
      Stack<Integer> st=new Stack<>();
      for(int i=n-1;i>=0;i--){
         while(!st.isEmpty() && st.peek()<=arr[i]){
            st.pop();
```

```
        }
        if(i<n){
            if(!st.isEmpty()){
                nge[i]=st.peek();
            }else{
                nge[i]=-1;
            }
        }
        st.push(arr[i]);
    }
    return nge;
  }
}
```

OUTPUT:

```
Next Greater Elements:
5 25 25 -1
=== Code Execution Successful ===
```

Time Complexity: O(n)

3. Parantheses Checker:

CODE:

```
import java.util.*;
class HelloWorld {
  public static boolean balance(String s){
    Stack<Character> st=new Stack<Character>();
    for(char i:s.toCharArray()){
      if(i=='('||i=='['||i=='{') st.push(i);
      else{
        if(st.isEmpty()) return false;
        char ch=st.pop();
        if((i==')'&&ch=='(')||(i==']'&&ch=='[')||(i=='}'&&ch=='{')) continue;
        else return false;
      }
    }
    return st.isEmpty();
  }
  public static void main(String[] args) {
    String s="()[{}()]";
    if(balance(s)==true)
    System.out.println("True");
    else
    System.out.println("False");
  }
}
```

OUTPUT:

```
True

=== Code Execution Successful ===
```

Time Complexity: O(n)

4. Union of two arrays with duplicate (unsorted):

CODE:

```
import java.util.*;
class Main {
   public static ArrayList<Integer> union(int[] a, int[] b){
      HashSet<Integer> s=new HashSet<>();
      for(int n:a) s.add(n);
      for(int n:b) s.add(n);
      ArrayList<Integer> r=new ArrayList<>();
      for(int i:s) r.add(i);
      return r;
   }
   public static void main(String[] args) {
      int[] a={1,2,3,2,1};
      int[] b={3,2,2,3,3,2};
      ArrayList<Integer> r=union(a,b);
      for(int n:r) System.out.print(n+" ");
   }
}
```

OUTPUT:

```
1 2 3
=== Code Execution Successful ===
```

Time Complexity: O(n+m)

5. Equilibrium Point:

CODE:

```
import java.util.*;
class Main {
   public static int equilibrium(long a[]){
      int n=a.length;
      int l=0,p=0,r=0;
      for(int i=1;i<n;i++){
         r+=a[i];
      }
      while(p<n-1 && r!=l){
         p++;
         r-=a[p];
         l+=a[p-1];
      }
      return (l==r)?p+1:-1;
   }
   public static void main(String[] args) {
      long[] a={1,7,3,6,5,6};
      int res=equilibrium(a);
      System.out.print(res);
   }
}
```

OUTPUT:

```
4
=== Code Execution Successful ===
```

Time Complexity: O(n)

6. Minimise height:

CODE:

```
import java.util.*;
class Main {
   public static int mindiff(int[] a, int k){
      int n=a.length;
      Arrays.sort(a);
      int r=a[n-1]-a[0];
      for(int i=1;i<a.length;i++){
         if(a[i]-k<0)
            continue;
         int minh=Math.min(a[0]+k,a[i]-k);
         int maxh=Math.max(a[i-1]+k,a[n-1]-k);
         r=Math.min(r,maxh-minh);
      }
      return r;
   }
   public static void main(String[] args) {
      int k=6;
      int[] a={12,6,4,15,17,10};
      int r=mindiff(a,k);
      System.out.println(r);
   }
}
```

OUTPUT:

```
8

=== Code Execution Successful ===
```

Time Complexity: O(n log n)

7. k-th smallest element:

CODE:

```
import java.util.*;
class Main {
   public static int kthsmall(int[] a, int n, int k){
      PriorityQueue<Integer> p=new PriorityQueue<>((x,y)->y-x);
      for(int i=0;i<n;i++){
         p.offer(a[i]);
         if(p.size()>k)
         p.poll();
      }
      return p.peek();
```

```
    }
    public static void main(String[] args) {
        int n=10;
        int k=4;
        int[] a={12,5,4,3,48,6,2,33,53,10};
        System.out.println("K-th Smallest Element:" +kthsmall(a,n,k));;
    }
}
```

OUTPUT:

```
K-th Smallest Element:5

=== Code Execution Successful ===
```

Time Complexity: O(n log k)