1. Next permutation:

CODE:

```
import java.util.*;
class next_permutation{
   static void next_perm(int[] a){
      int n=a.length;
      int piv=-1;
      for(int i=n-2;i>=0;i--){
         if(a[i]<a[i+1]){
            piv=i;
            break;
         }
      }
      if(piv==-1){
         reverse(a,0,n-1);
         return;
      }
      for(int i=n-1;i>piv;i--){
         if(a[i]>a[piv]){
            swap(a,i,piv);
            break;
         }
      }
      reverse(a,piv+1,n-1);
   }
   private static void reverse(int[] a, int st, int en){
      while(st<en){
         swap(a,st++,en--);
      }
   }
   private static void swap(int[] a, int i, int j){
      int t=a[i];
      a[i]=a[j];
      a[j]=t;
   }
   public static void main(String[] args){
      int[] a={2,4,1,7,5,0};
      next_perm(a);
      for(int i=0;i<a.length;i++)
      System.out.print(a[i]+" ");
   }
}
```

OUTPUT:

```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_7\" && javac next_permutation.java && java
next_permutation
2 4 5 0 1 7
```

Time complexity: O(n)

2. Spiral matrix:

CODE:

```
import java.util.*;
class HelloWorld {
   public static List<Integer> printSpiral(int[][] mat) {
      List<Integer> ans = new ArrayList<>();
      int n = mat.length;
      int m = mat[0].length;
      int top = 0, left = 0, bottom = n - 1, right = m - 1;
      while (top <= bottom && left <= right) {
         for (int i = left; i <= right; i++)
            ans.add(mat[top][i]);
         top++;
         for (int i = top; i <= bottom; i++)
            ans.add(mat[i][right]);
         right--;
         if (top <= bottom) {
            for (int i = right; i >= left; i--)
               ans.add(mat[bottom][i]);
            bottom--;
         }
         if (left <= right) {
            for (int i = bottom; i >= top; i--)
               ans.add(mat[i][left]);
            left++;
         }
      }
      return ans;
   }
   public static void main(String[] args) {
      int[][] mat = {{1, 2, 3, 4},
                     {5, 6, 7, 8},
                     {9, 10, 11, 12},
                     {13, 14, 15, 16}};
      List<Integer> ans = printSpiral(mat);
      for(int i = 0;i<ans.size();i++){
         System.out.print(ans.get(i) + " ");
      }
      System.out.println();
   }
}
```

OUTPUT:

```
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

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

Time complexity: O(m*n)

3. Longest substring without repeating characters:

CODE:

```
class long_substr_without_repeating_char{
   static int longsubstr(String s){
      if(s.length()==0) return 0;
      if(s.length()==1) return 1;
      int maxlen=0;
      boolean[] vis=new boolean[256]; //ascii value limit
      int l=0, r=0;
      while(r<s.length()){
         while(vis[s.charAt(r)]){
            vis[s.charAt(l)]=false;
            l++;
         }
         vis[s.charAt(r)]=true;
         maxlen=Math.max(maxlen, (r-l+1));   //r-l+1=size of sliding window
         r++;
      }
      return maxlen;
   }
   public static void main(String[] args){
      String s="geeksforgeeks";
      System.out.println(longsubstr(s));
   }
}
```

OUTPUT:

```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_7\" && javac long_substr_without_repeating_char.
java && java long_substr_without_repeating_char
7
```

Time complexity: O(n)

4. Remove linked list elements:

CODE:

```
class Node{
   int data;
   Node next;
   Node(int ndata){
      data=ndata;
      next=null;
   }
}
public class delete_in_linked_list{
   static Node delll(Node head, int k){
      Node curr=head, prev=null;
      while(curr!=null){
         if(curr.data==k){
            if(prev==null){
```

```
                head=curr.next;
            }else{
                prev.next=curr.next;
            }
            curr=curr.next;
        } else {
            prev=curr;
            curr=curr.next;
        }
    }
    return head;
  }
  static void printll(Node curr){
    while(curr!=null){
        System.out.print(" "+ curr.data);
        curr=curr.next;
    }
  }
  public static void main(String[] args){
    Node head=new Node(2);
    head.next=new Node(2);
    head.next.next=new Node(1);
    head.next.next.next=new Node(8);
    head.next.next.next.next=new Node(2);
    int k=2;
    head=delll(head,k);
    printll(head);
  }
}
```

OUTPUT:

```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_7\" && javac delete_in_linked_list.java && java
delete_in_linked_list
 1 8
```

Time complexity: O(n)

5. Palindrome linked list:

CODE:

```
class Node{
  int data;
  Node next;
  Node(int d){
    data=d;
    next=null;
  }
}
class Palindrome_linked_list{
  static Node reverse(Node head){
    Node prev=null;
```

```
        Node curr=head;
        Node next;
        while(curr!=null){
            next=curr.next;
            curr.next=prev;
            prev=curr;
            curr=next;
        }
        return prev;
    }
    static boolean identical(Node n1, Node n2){
        while(n1!=null && n2!=null){
            if(n1.data!=n2.data) return false;
            n1=n1.next;
            n2=n2.next;
        }
        return true;
    }
    static boolean palindrome(Node head){
        if(head==null || head.next==null) return true;
        Node slow=head, fast=head;
        while(fast.next!=null && fast.next.next!=null){
            slow=slow.next;
            fast=fast.next.next;
        }
        Node head2=reverse(slow.next);
        slow.next=null;
        boolean ret=identical(head,head2);
        head2=reverse(head2);
        slow.next=head2;
        return ret;
    }
    public static void main(String[] args){
        Node head=new Node(1);
        head.next=new Node(2);
        head.next.next=new Node(3);
        head.next.next.next=new Node(2);
        head.next.next.next.next=new Node(1);
        boolean res=palindrome(head);
        if(res) System.out.println("true");
        else System.out.println("false");
    }
}
```

OUTPUT:

```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_2\" && javac Palindrome_linked_list.java && java
Palindrome_linked_list
true
```

Time complexity: O(n)

6. Minimum path sum:

CODE:

```
import java.util.*;
class minimum_path{
   static int minpath(int i, int j, int[][] mat, int[][] dp){
      if(i==0 && j==0) return mat[0][0];
      if(i<0 || j<0) return (int) Math.pow(10,9);
      if(dp[i][j]!=-1) return dp[i][j];
      int up=mat[i][j]+minpath(i-1, j, mat, dp);
      int left=mat[i][j]+minpath(i, j-1, mat, dp);
      return dp[i][j]=Math.min(up,left);
   }
   static int minsum(int n, int m, int[][] mat){
      int dp[][]=new int[n][m];
      for(int row[]:dp) Arrays.fill(row,-1);
      return minpath(n-1, m-1, mat, dp);
   }
   public static void main(String[] args){
      int mat[][]={{5,9,6},{11,5,2}};
      int n=mat.length;
      int m=mat[0].length;
      System.out.println(minsum(n,m,mat));
   }
}
```

OUTPUT:

```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_7\" && javac minimum_path.java && java minimum_path
21
```

Time complexity: O(n*m)

7. Validate binary search tree:

CODE:

```
class Node{
   int data;
   Node left, right;
   Node(int val){
      data=val;
      left=right=null;
   }
}
class validate_bst{
   static boolean validbst(Node root){
      Node curr=root;
      Node pre;
      int preval=Integer.MIN_VALUE;
      while(curr!=null){
         if(curr.left==null){
            if(curr.data<=preval){
```

```
                return false;
            }
            preval=curr.data;
            curr=curr.right;
        }else{
            pre=curr.left;
            while(pre.right!=null && pre.right!=curr){
                pre=pre.right;
            }
            if(pre.right==null){
                pre.right=curr;
                curr=curr.left;
            }else{
                pre.right=null;
                if(curr.data<=preval){
                    return false;
                }
                preval=curr.data;
                curr=curr.right;
            }
        }
    }
    return true;
  }
  public static void main(String[] args){
    Node root=new Node(4);
    root.left=new Node(2);
    root.right=new Node(5);
    root.left.left=new Node(1);
    root.left.right=new Node(3);
    if(validbst(root)){
        System.out.println("True");
    }else{
        System.out.println("False");
    }
  }
}
```

OUTPUT:

```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_7\" && javac validate_bst.java && java validate_bst
True
```

Time complexity: O(n)

8. Word ladder:

CODE:

```
import java.util.*;
class word_ladder{
  static int shortchainlen(String st, String k, Set<String> dict){
    if(st==k) return 0;
```

```
        if(!dict.contains(k)) return 0;
        int lvl=0, wordlen=st.length();
        Queue<String> q=new LinkedList<>();
        q.add(st);
        while(!q.isEmpty()){
            ++lvl;
            int qlen=q.size();
            for(int i=0;i<qlen;i++){
                char[] word=q.peek().toCharArray();
                q.remove();
                for(int pos=0; pos<wordlen;pos++){
                    char orig=word[pos];
                    for(char c='a'; c<='z';c++){
                        word[pos]=c;
                        if(String.valueOf(word).equals(k)) return lvl+1;
                        if(!dict.contains(String.valueOf(word))) continue;
                        dict.remove(String.valueOf(word));
                        q.add(String.valueOf(word));
                    }
                    word[pos]=orig;
                }
            }
        }
        return 0;
    }
    public static void main(String[] args){
        Set<String> dict=new HashSet<String>();
        dict.add("poon");
        dict.add("plee");
        dict.add("same");
        dict.add("poie");
        dict.add("plie");
        dict.add("poin");
        dict.add("plea");
        String st="toon";
        String k="plea";
        System.out.print(shortchainlen(st, k, dict));
    }
}
```

OUTPUT:

```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_7\" && javac word_ladder.java && java word_ladder
7
```

Time complexity: O(n*m)