A) **String Compression - Implement a method to perform String Compression.**

Ans- In the given solution, the time complexity is O(n^2) because nested loop has been used and space complexity is O(1). The time complexity can be improved by using a single loop to iterate over the string as shown in the code below:-

```
public class Main
{
   public static String compress1(String str) {
      if(str.length()==0){
         return "";
      }
     String output=str.charAt(0)+"";
     int count=1;
     for(int i=1;i<str.length();i++){
        char curr=str.charAt(i);
        char prev=str.charAt(i-1);
        if(curr == prev){
           count++;
        }else{
           output=output+count+curr;
           count=1;
        }
     }
     return output+count;
   }
   public static String compress2(String str) {
      if(str.length()==0){
         return "";
      }
     String output=str.charAt(0)+"";
     for(int i=1;i<str.length();){
        if(i+2<str.length() && str.charAt(i) == str.charAt(i+2)){
           output+=str.charAt(i+1)+""+str.charAt(i);
```

```
            i=i+2;
         }else{
            if(output.charAt(output.length()-1)== str.charAt(i)){
               i=i+1;

            }else{
               output+=str.charAt(i);
                i=i+1;
            }

         }
      }
      return output;
   }
        public static void main(String[] args) {
             System.out.println(compress1("aabbcccaaa"));
        }
}
```

Bonus:1 The answer should be taken into second compressor and compress further.

Ans-

```
public class Main
{
   public static String compress1(String str) {
      if(str.length()==0){
         return "";
      }
      String output=str.charAt(0)+"";
      int count=1;
      for(int i=1;i<str.length();i++){
         char curr=str.charAt(i);
         char prev=str.charAt(i-1);
         if(curr == prev){
            count++;
            }else{
                output=output+count+curr;
                count=1;
            }
        }
        return output+count;
    }
    public static String compress2(String str) {
         if(str.length()==0){
             return "";
         }
        String output=str.charAt(0)+"";
        for(int i=1;i<str.length();){
            if(i+2<str.length() && str.charAt(i) == str.charAt(i+2)){
                output+=str.charAt(i+1)+""+str.charAt(i);
                i=i+2;
            }else{
                if(output.charAt(output.length()-1)== str.charAt(i)){
                    i=i+1;

                }else{
                    output+=str.charAt(i);
                     i=i+1;
                }

            }
        }
        return output;
    }
        public static void main(String[] args) {
                String output1=compress1("aabbcccaaa");
                System.out.println(compress2(output1));
        }
}
```

Bonus:2 Decompress ab2c1ac3 into aabbcaaaccc.

Ans-

```
import java.util.*;
public class Main {
   public static void main(String[] args) {
      String str = "ab2c1ac3";
      String output="";
      int count=0;
      for(int i=str.length()-1;i>=0;i--){
         char ch=str.charAt(i);
         if(Character.isDigit(ch)){
            count = ch-'0';
            continue;
         }
         for(int j=0;j<count;j++){
            output=str.charAt(i)+output;
         }
      }
      System.out.print(output);
   }
}
```

B) Find a way to find the kth to last element in a given linked list(Assuming length of Linked List is not known).

Ans-

```
public class LinkedListNode
{
   int data;
   LinkedListNode next;
   LinkedListNode(int data, LinkedListNode next)
   {
      this.data = data;
      this.next = next;
   }
```

```
}

public class Main
{
   public static LinkedListNode findKthNode(LinkedListNode
head, int k)
   {
      LinkedListNode curr = head;
      for (int i = 0; curr != null && i < k; i++) {
         curr = curr.next;
         if (curr == null && i != k-1) {
            return null;
         }
      }
      while (curr != null)
      {
         head = head.next;
         curr = curr.next;
      }
      return head;
   }

   public static void main(String[] args)
   {
      int[] keys = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

      LinkedListNode head = null;
      for (int i = keys.length - 1; i >= 0; i--) {
         head = new LinkedListNode(keys[i], head);
      }
      int k = 6;
      LinkedListNode node = findKthNode(head, k);

      if (node != null) {
         System.out.println("k-th node from the end is " +
node.data);
      }
   }
}
```

Bonus-1:Minimizing the number of times running through the loop.

Ans-Using Recursion

```
public class LinkedListNode
{
   int data;
   LinkedListNode next;
   LinkedListNode(int data, LinkedListNode next)
   {
      this.data = data;
      this.next = next;
   }
}

public class Main
{
   public static int findKthNode(LinkedListNode  node, int k)
   {
      if (node == null) {
         return 0;
      }

      int count = findKthNode(node.next, k) + 1;

      if (count == k) {
         System.out.println("k'th node from the end is " +
node.data);
      }
```

```
        return count;
    }


    public static void main(String[] args)
    {
        int[] keys = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        LinkedListNode  head = null;
        for (int i = keys.length - 1; i >= 0; i--) {
            head = new LinkedListNode (keys[i], head);
        }

        int k = 11;
        findKthNode(head, k);
    }
}
```

C) Stack minimum - Stack has functions of push and pop. Can you also add a function 'min' to the stack and it should also execute in O(1).

Ans-

```
import java.util.*;

public class MyStack  {
    Stack<Integer> stack;
    Integer minElement;

    MyStack () {
        stack = new Stack<Integer>();

    }

    void getMin()
    {
```

```
    if (stack.isEmpty())
        System.out.println("Stack is empty");
    else
        System.out.println("Minimum Element in the "
                         + " stack is: " + minElement);
}


void pop()
{
    if (stack.isEmpty()) {
        System.out.println("Stack is empty");
        return;
    }
    System.out.print("Top Most Element Removed: ");
    Integer t = stack.pop();
    if (t < minElement) {
        System.out.println(minElement);
        minElement = 2 * minElement - t;
    }
    else
        System.out.println(t);
}
void push(Integer x)
{
    if (stack.isEmpty()) {
        minElement = x;
        stack.push(x);
        System.out.println("Number Inserted: " + x);
        return;
    }
    if (x < minElement) {
        stack.push(2 * x - minElement);
        minElement = x;
    }
```

```
        else
           stack.push(x);

        System.out.println("Number Inserted: " + x);
    }
};
public class Main {
    public static void main(String[] args)
    {
        MyStack  s = new MyStack ();
        s.push(3);
        s.push(5);
        s.getMin();
        s.push(2);
        s.push(1);
        s.getMin();
        s.pop();
        s.getMin();
    }
}
```

Bonus:1 Real world use case where stack is better than the array is
Undo/Redo Functionality:
In many applications, such as text editors or image editing software, undo/redo functionality is implemented using a stack. Each action performed by the user is added to the stack, and can be undone by popping the last action off the stack. In this case, a stack is more suitable than an array because it provides efficient push and pop operations.

D) Given an array of integers representing the elevation of a roof structure at various positions, each position is separated by a unit length, Write a program to determine the amount of water that will be trapped on the roof after heavy rainfall.

Ans-

```
import java.lang.Math;
import java.util.Scanner;
public class Main {
   public static int findWater(int[] arr,int n){
      int[] left=new int[n];
      int[] right=new int[n];
      int total_water=0;
      left[0]=arr[0];
      for(int i=1;i<n;i++){
         left[i]=Math.max(left[i-1],arr[i]);
      }
      right[n-1]=arr[n-1];
      for(int i=n-2;i>=0;i--){
         right[i]=Math.max(right[i+1],arr[i]);
      }
      for(int i=0;i<n;i++){
         total_water+=Math.min(left[i],right[i])-arr[i];
      }
      return total_water;
   }
   public static void main(String[] args) {
      Scanner sc=new Scanner(System.in);
      int n=sc.nextInt();
      int[] arr=new int[n];
      for(int i=0;i<n;i++){
         arr[i]=sc.nextInt();
      }
      System.out.print(findWater(arr,n));
  }
}
```

E) You will be given a list coin denominations that you can use to tender change to your customers, find the most optimum way to tender the exact change to your customers, the optimum is when you use the least number of coins.

Ans-

```
import java.util.ArrayList;
public class Main{
   public static void coinChange(int val,int[] deno){
      ArrayList<Integer> ans=new ArrayList<>();
      int temp=val;
      for(int i=deno.length-1;i>=0;i--){
         while(temp >= deno[i]){
            temp-=deno[i];
            ans.add(deno[i]);
         }
      }
      for(int i=0;i<ans.size();i++){
         System.out.print(ans.get(i)+" ");
      }
   }
   public static void main(String[] args){
      int n=7;
      int[] arr={1, 2, 5, 8, 10};
      coinChange(n,arr);
   }
}
```

Explain all the scenarios in better words and simpler to understand format compared to explanation available in the link below:

https://www.geeksforgeeks.org/coin-change-dp-7/

Ans-

Given Problem- Given an integer array of coins[ ] of size N representing different types of currency and an

integer sum, The task is to find the number of ways to make sum by using different combinations from coins[].
Note: Assume that you have an infinite supply of each type of coin.
Solution using Recursion-
We need to find all possible ways to make sum by using different denominations of coins, at each step we can either include or exclude a particular denomination.We can start the problem by creating a recursion tree

```
                       C({1,2,3}, 5)
                      /            \
                     /              \
            C({1,2,3}, 2)            C({1,2}, 5)
                 /     \                  /  \
                /        \              /       \
   C({1,2,3}, -1)  C({1,2}, 2)      C({1,2}, 3)   C({1}, 5)
                 /   \            /    \        /    \
                /     \          /      \      /      \
  C({1,2},0)  C({1},2)   C({1,2},1) C({1},3)    C({1}, 4)  C({}, 5)
                 /\        /\        /\        /    \
                /    \    /    \    /  \      /      \
           .     . .    . .    .         C({1}, 3)     C({}, 4)
                                           /\
                                          /  \                      .
```

- We have 2 choices for a coin of a particular denomination, either i) to include, or ii) to exclude.
- If we are at coins[n-1], we can take as many instances of that coin (unbounded inclusion) i.e count(coins, n, sum – coins[n-1]); then we move to coins[n-2].
- After moving to coins[n-2], we can’t move back and can’t make choices for coins[n-1] i.e count(coins, n-1, sum).
- Finally, as we have to find the total number of ways, so we will add these 2 possible choices, i.e count(coins, n, sum – coins[n-1]) + count(coins, n-1, sum);

As we can see in the above recursion tree that there are overlapping sub-problems so we can solve the problmem in a more optimum way using dynamic programming.
To solve the problem using dynamic programming we can follow the below steps-

- Using 2-D vector to store the Overlapping subproblems.
- Traversing the whole array to find the solution and storing in the memoization table.
- Using the memoization table to find the optimal solution.

Explain what is greedy algorithm and how dynamic programming helps in this case.
Ans- A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.

In the above problem, we can see that the problem has been solved using greedy algorithm but it does not give the most optimal solution in every case.
For example, it doesn’t work for denominations {1, 5, 7} and V = 18. The above approach would print 7, 7, 1, 1, 1 and 1. But we can use 4 denominations 5, 5, 7 and 1.
This problem can more optimally be solved using dynamic programming technique as in this technique we can get the best solution from all possible solutions.

Bonus Ques: Given a number N, remove one digit and print largest possible number.
Ans- Since in the given question, It was asked to remove one digit and print the largest possible number but the examples given in the question show that we need to remove the smallest digit and print the rest of the digits in the same order as they were occurring in the original number so I have solved both types of question.

First part- Remove the smallest digit present in the given number and print the largest possible number.

```
import java.lang.Math;
import java.util.Arrays;
import java.util.Scanner;
public class LargestNum {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num = sc.nextInt();
        int temp1 = num;
        int temp2 = num;
        int newNum = 0;
        int count = 0;
        int min = Integer.MAX_VALUE;
        while (temp1 > 0) {
            int rem = temp1 %10;
            if (rem < min) {
                min = rem;
            }
            temp1 = temp1 / 10;
            count++;

        }
        int[] arr = new int[count];
        int i=0;
        while(temp2>0){
            arr[i]=temp2%10;
            i++;
            temp2=temp2/10;
        }

        Arrays.sort(arr);
        for(int j= arr.length-1;j>0;j--){
          if(arr[j] != min){
             newNum+=(int)Math.pow(10, (count-2))*arr[j];
             count--;
          }
       }

       System.out.println(newNum);
    }
}
```

Second Part- Remove the smallest digit and print the rest of the digits in the same order as they are occurring in the original number.

```
import java.lang.Math;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
       Scanner sc=new Scanner(System.in);
       int num=sc.nextInt();
       int temp=num;
       int newNum=0;
       int power=0;
       int min=Integer.MAX_VALUE;
       while(temp>0){
          int rem=temp%10;
          if(rem<min){
             min=rem;
          }
          temp=temp/10;
       }
        System.out.println(min);

       while(num>0){
          int rem=num%10;
```

```
      if(rem != min){
        newNum +=Math.pow(10,power)*rem;
        power++;
      }else{
        num=num/10;
        continue;
      }
      num=num/10;
    }
    System.out.print(newNum);
  }
}
```

F) What is dot product and cross product?Explain use cases of where dot product is used and cross product is used in graphics environment.

Ans-

Dot Product

The Dot Product is a vector operation that calculates the angle between two vectors. The dot product is calculated in two different ways.

Version 1

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$\vec{u} \cdot \vec{v} = u_1 v_1 + u_2 v_2 + u_3 v_3$$

In the above equation, information about the angle between the vectors is missing. However, the result from this equation can tell us the direction of each vector. For example, if the dot product is equal to 1, it means that both vectors have the same direction. If the dot product is 0, it means that both vectors are

perpendicular on each other. Finally, if the dot product is -1, it means that both vectors are heading in opposite directions.
Version 2
If we are interested in finding the angle between two vectors, the dot-product equation below can be used.

$$\vec{u} \cdot \vec{v} = ||u||\ ||v||\ \cos\theta$$

In graphics, the dot product is used to determine the angle between two vectors, the projection of one vector onto another, and the determination of whether two vectors are perpendicular.For example, in lighting calculations, the dot product is used to determine the amount of light that hits a surface. The dot product of the light vector and the surface normal vector gives the cosine of the angle between them, which represents the amount of light that is reflected by the surface.

Cross Product
Two vectors produces a plane. A cross product operation produces a vector that is perpendicular to both vectors.
The cross product of two vectors is calculated as follows:

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$\vec{u} \times \vec{v} = \begin{bmatrix} u_2v_3 - u_3v_2 \\ u_3v_1 - u_1v_3 \\ u_1v_2 - u_2v_1 \end{bmatrix}$$

In graphics, the cross product is used to determine the orientation of surfaces, the direction of rotations, and the determination of whether two vectors are parallel.For example, in calculating the normal vector of a surface, the cross product is used to find the vector that is perpendicular to the surface.

This normal vector is important for shading calculations and for determining how light interacts with the surface.

Source - https://www.haroldserrano.com/blog/vectors-in-computer-graphics#:~:text=They%20have%20two%20distinct%20type,the%20multiplicand%20and%20multiplier%20vectors.

G) Explain a piece of code that you wrote which you are proud of? If you have not written any code, please write your favorite subject in engineering studies. We can go deep into that subject.
Ans- I felt really proud when I build my first React App.It was a Cart App.I learnt lot of new things while making this app like how to use the component cards, how to use props and state, the code reuseability, learnt how to connect my project to the firebase database, rendering the App.

H) Random crashes- you are given a source code to test it and randomly crashes and it never crashes at the same place(you have attached a debugger and you find this).Explain what all you would suspect and how would you go about with isolating the cause.
Ans- To debug random crashes in a source code the following steps can be taken:

1. Collect data: First, you need to collect data about the crashes that are occurring. You can use logging, crash reports, or debuggers to collect this data. Look for patterns in the data, such as common error messages or the same set of variables being involved in multiple crashes.
2. Exception Handling: Exception handling can be done at every possible step to catch the different exceptions in the code.
3. Memory management issues: One common cause of random crashes is memory management issues, such as accessing

memory that has already been freed, writing to memory that has not been allocated, or accessing memory beyond the bounds of an array.
4. Race conditions: Another possible cause of random crashes is race conditions, which occur when multiple threads or processes access the same resources in an unpredictable order.
5. Compiler or optimization issues: Occasionally, random crashes can be caused by issues with the compiler or optimization settings used to build the code. These issues can be related to the way that the compiler and hardware interact to generate executable code.

Declaration- I declare that I have done the above work by myself and not worked with anyone or got help from any individual on the internet.