Interview Questions

a) String compression Implement a method to perform string compression. E.g. 'aabcccccaaa' should be a2b1c5a3. The code to implement this is given in the link - https://www.educative.io/answers/string-compression-using-run-length-encoding

Think about memory occupied and how it can be improved.

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

E.g. a2b2c1a3c3 should become ab2c1ac3

Bonus 2: decompress2 - ab2c1ac3 should return aabbcaaaccc.

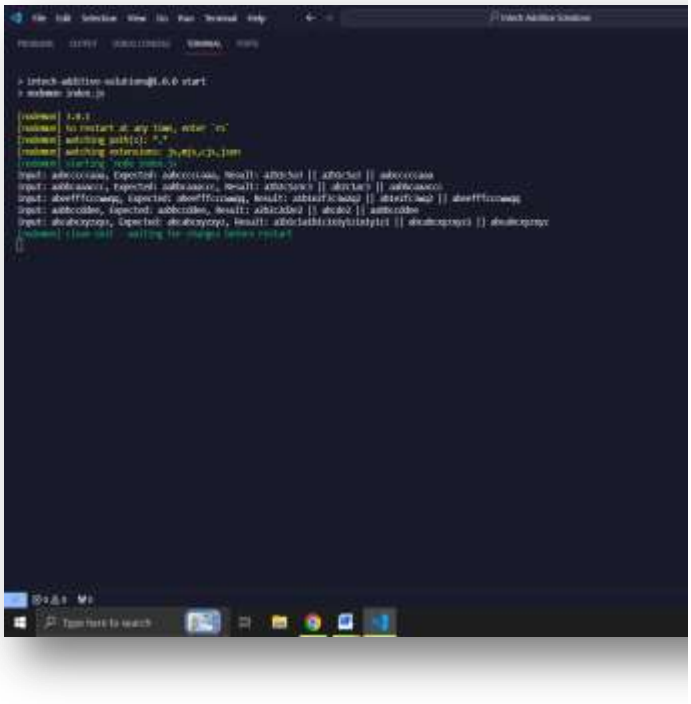
Think about how you will test this code.

⇨ Code: https://github.com/arunava-saha/signmentInterviewQuestions

```
// String Comprassion ---- 1
// Function to perform string compression
function compress1(s) {
  const str = [];
  let count = 1;
  for (let i = 0; i < s.length; i++) {
    if (s[i] === s[i + 1]) {
      count++;
    } else {
      str.push(s[i] + count);
      count = 1;
    }
  }
  return str.join("");
}

// Bonus 1: Second compressor
function compress2(s) {
  let compressed = s[0];
  let count = s[1];

  for (let i = 2; i < s.length; i++) {
    if (count === s[i + 1]) {
      compressed += s[i];
    } else {
      compressed += count;
      compressed += s[i];
      count = s[i + 1];
    }
    i++;
  }
  compressed += s[s.length - 1];
  return compressed;
}
//  Is a digit (0 to 9) and returns true if it is, and false
const is_Int = (ch) => "0" <= ch && ch <= "9";
```

```
// Bonus 2: Decompressor
function decompress(s) {
  let output = "";
  for (let i = s.length - 1; i >= 0; ) {
    let span = 0;
    while (is_Int(s[i])) {
      if (span === 0) {
        span = Number(s[i]);
      } else {
        span = span + 10 * Number(str[i]);
      }
      --i;
    }
    // collect non-digits into the `output`, stop at the end or at a digit
    while (!is_Int(s[i])) {
      if (i === -1) break;
      output += s[i].repeat(span);
      --i;
    }
    if (i === -1) break;
  }
  return output.split("").reverse().join("");
}

// Test the code
// Testing function
function testCompression() {
  const testCases = [
    { input: "aabcccccaaa" },
    { input: "aabbcaaaccc" },
    { input: "abeefffcccwwqq" },
    { input: "aabbccddee" },
    { input: "abcabcxyzxyz" },
  ];

  for (const testCase of testCases) {
    const result1 = compress1(testCase.input);
    const result2 = compress2(result1);
    const result = decompress(result2);
    console.log(
      `Input: ${testCase.input}, Expected: ${testCase.input}, Result: ${result1} || ${result2} || ${result}`
    );
    console.assert(result === testCase.input, "Test failed!");
  }
}
testCompression();
```

Online Compilar



Local System

b) Linked List - The link shows a program to find the nth element of a linked list. https://www.geeksforgeeks.org/nth-node-from-the-end-of-a-linked-list/

Find a way to find the kth to the last element of linked list (assume length of linked list is not known)
Bonus 1: Can you minimize the number of times you run through the loop.

```
import java.io.*;
class LinkedList {
  Node head;
  class Node {
    int data;
    Node next;
    Node(int d) {
      data = d;
      next = null;
    }
  }
```

```
  Node getKthFromEnd(int k) {
    Node slow = head;
    Node fast = head;
    for (int i = 0; i < k; ++i) {
      if (fast == null) {
        System.out.println("has less than k nodes.");
        return null;
      }
      fast = fast.next;
    }
    while (fast != null) {
      slow = slow.next;
      fast = fast.next;
    }
    return slow;
  }

  public void push(int new_data) {
    Node new_node = new Node(new_data);
    new_node.next = head;
    head = new_node;
  }

  // Main code
  public static void main(String[] args) {
    LinkedList llist = new LinkedList();
    llist.push(20);
    llist.push(4);
    llist.push(15);
    llist.push(35);
    Node kth = llist.getKthFromEnd(3);
    if (kth != null) {
      System.out.println("Kth node from end is: " + kth.data);
    }
  }
}
```

Online Compilar

c) Stack minimum- Details of stack data structure is available in https://www.geeksforgeeks.org/stack-data-structure/ 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). If you are not aware of O(1), refer to some videos online. E.g. https://en.wikipedia.org/wiki/Big_O_notation Bonus 1 – Explain one real world use case where stack is better used data structure than arrays.

```
import java.util.Stack;

class MinStack {
  private Stack<Integer> mainStack;
  private Stack<Integer> minStack;
  public MinStack() {
    mainStack = new Stack<>();
    minStack = new Stack<>();
  }
  // Push operation
  void push(int value) {
    mainStack.push(value);
    if (minStack.isEmpty() || value <= minStack.peek()) {
      minStack.push(value);
    }
  }
  // Pop operation
  void pop() {
    if (!mainStack.isEmpty()) {
      if (mainStack.peek().equals(minStack.peek())) {
        minStack.pop();
      }
      mainStack.pop();
    }
  }
  // Get the minimum element in the stack
  int min() {
    if (!minStack.isEmpty()) {
      return minStack.peek();
    } else {
      System.out.println("Stack is empty.");
      return -1;
    }
  }
  static void prnt(MinStack stack) {
    System.out.println("Current Min: " + stack.min());
  }
  public static void main(String[] args) {
    MinStack stack = new MinStack();
    prnt(stack);
    stack.push(31);
    stack.push(19);
    prnt(stack);
    stack.push(121);
    stack.push(16);
    prnt(stack);
    stack.pop();
    prnt(stack);
    stack.pop();
    prnt(stack);
  }
}
```

Referance: https://www.geeksforgeeks.org/stack-class-in-java/

Online Compilar

Explain one real world use case where stack is better used data structure than arrays.

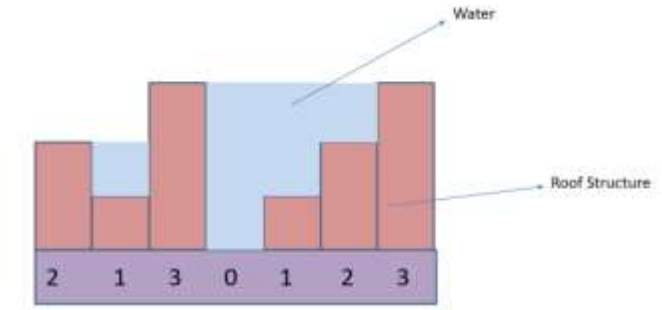
1. UNDO features , 2. Browser History etc.

When you need to bring things out in the opposite order like(FILO) that they were put in, use a stack.

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

Example:
input : [2 1 3 0 1 2 3]

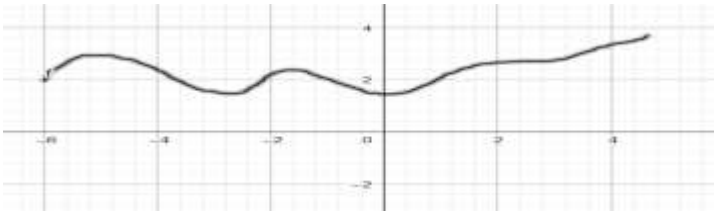
Ans : 7 units of water will be trapped



https://www.geeksforgeeks.org/trapping-rain-water/

Go through the above code for the solution.

The next phase is that the values are now not discrete but analog. E.g. I give an equation of function that is bounded, can you predict how many units of water gets trapped.



```
import java.util.function.Function;
// Java program to find maximum amount of water that can
// be trapped within given set of bars.

class Test {
    static int arr[]
        = new int[] { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };
    static int findWater(int n)
    {
        int left[] = new int[n];
        int right[] = new int[n];
        int TotalWater = 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++)
            TotalWater += Math.min(left[i], right[i]) - arr[i];

        return TotalWater;
    }

    static double smallIncrement = 0.001; // Width of each segment

    static double trapWaterVolume(Function<Double, Double>
profileFunction, double start, double end) {
        double TotalWater = 0.0;
        double groundLevel = 0.0;

        for (double x = start; x <= end; x += smallIncrement) {
            double barHeight = profileFunction.apply(x);
            double trappedHeight = Math.max(0, barHeight -
groundLevel);
            TotalWater += trappedHeight * smallIncrement;
        }

        return TotalWater;
    }

    public static void main(String[] args) {
        // Test cases as discrete
        System.out.println("TotalWater: " + findWater(arr.length));

        // Test cases as analog
        // Test case 1: Linear equation
        Function<Double, Double> linearProfile = x -> 2 * x; // Example:
y = 2x
        double linearResult = trapWaterVolume(linearProfile, 0, 5);
        System.out.println("Trapped Water (Linear): " + linearResult);

        // Test case 2: Quadratic equation
        Function<Double, Double> quadraticProfile = x -> -(x - 2) * (x - 4)
+ 5; // Example: y = -(x-2)(x-4) + 5
        double quadraticResult = trapWaterVolume(quadraticProfile, 1,
5);
        System.out.println("Trapped Water (Quadratic): " +
quadraticResult);
    }
}
```

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.

Example:
input => [1, 2, 5, 8, 10] (available coins)
Input => 7 (Change to be given)
Ans : [2, 5]

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/

Explain what is a greedy algorithm and how dynamic programming helps in this case.

Bonus question: given a number N, remove one digit and print the largest possible number. E.g. Why is the above solution part of a greedy algorithm?

5 -
1234
2945
9273
3954
19374

Answers:

234
945
973
954
9374

```java
import java.util.*;
// JAVA program for coin change problem.
// And Bonus Problem
class CoinChange {
    static int countRe(int coins[], int n, int sum){
        if (sum == 0)
            return 1;
        if (sum < 0)
            return 0;
        if (n <= 0)
            return 0;
        return countRe(coins, n - 1, sum)
            + countRe(coins, n, sum - coins[n - 1]);
    }
    static int countMe(int[] coins, int sum, int n, int[][] dp){
        if (sum == 0)
            return dp[n][sum] = 1;
        if (n == 0 || sum<0)
            return 0;
        if (dp[n][sum] != -1)
            return dp[n][sum];
        return dp[n][sum]
            = countMe(coins, sum - coins[n - 1], n, dp)
            + countMe(coins, sum, n - 1, dp);
    }
    static long countDp(int coins[], int n, int sum) {
        int dp[] = new int[sum + 1];
        dp[0] = 1;
        for (int i = 0; i < n; i++) {
            for (int j = coins[i]; j <= sum; j++) {
                dp[j] += dp[j - coins[i]];
            }
        }
        return dp[sum];
    }
    public static String RemovingSmallestDigit(String number) {
        int n = number.length();
        StringBuilder result = new StringBuilder();
        int index = 0;
        while (index < n - 1 && number.charAt(index) >=
number.charAt(index + 1)) {
            index++;
        }
        if (index == n - 1) {
            return number.substring(0, n - 1);
        }
        result.append(number.substring(0,
index)).append(number.substring(index + 1));
        return result.toString();
    }
    public static void main(String args[]) {
        int coins[] = {1, 2, 5, 8, 10};
        int n = coins.length;
        int sum = 7;
        System.out.println("Number of ways to make change in
Recursion: " + countRe(coins, n, sum));
        System.out.println("Number of ways to make change in
DP: " + countDp(coins, n, sum));
        int tc = 1;
        while (tc != 0) {
            int n1, sum1;
            n1 = 5;
            sum1 = 7;
            int[] coins1 = {1, 2, 5, 8, 10};
            int[][] dp = new int[n1 + 1][sum1 + 1];
            for (int[] row : dp)
                Arrays.fill(row, -1);
            int res = countMe(coins1, sum1, n1, dp);
            System.out.println("Number of ways to make change in
Memorization: " + res);
            tc--;
        }
        // Test cases
        System.out.println(RemovingSmallestDigit("1234")); //
Output: 234
        System.out.println(RemovingSmallestDigit("2945")); //
Output: 945
        System.out.println(RemovingSmallestDigit("9273")); //
Output: 973
        System.out.println(RemovingSmallestDigit("3954")); //
Output: 954
        System.out.println(RemovingSmallestDigit("19374")); //
Output: 9374
    }
}
```

Code: At GitHub Please read the Readme file

Greedy algorithm: it chooses the best option available at the current moment without considering the consequences of that choice on future steps. It typically doesn't consider the same sub problem multiple times. In this problem that is less than or equal to the remaining amount at each step. This works for certain cases but fails for others (e.g., denominations of {1, 3, 4} and target sum 6).

Dynamic programming: It is particularly useful when a problem has overlapping sub problems and optimal substructure. This avoids redundant computations and significantly improves the algorithm's efficiency. In this problem, it computes the number of ways to make change for each value from 0 to the target sum, considering all possible coin denominations. The solutions to sub problems are reused to build the solution for the larger problem.

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. Add links to places where you studied this information and get back with the understanding.

Bonus - How do you calculate the intersection between a ray and a plane/sphere/triangle?

⇨ The dot product can be used to find the length of a vector or the angle between two vectors. The cross product is used to find a vector which is perpendicular to the plane spanned by two vectors.

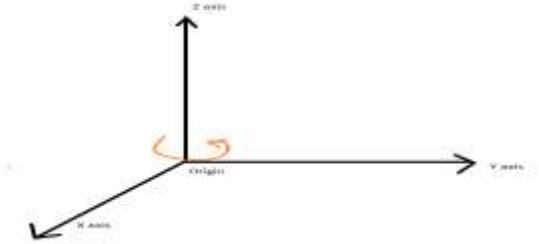
Reference: https://www.geeksforgeeks.org/dot-and-cross-products-on-vectors/

dot product = a . b = b . a = |a||b| cos θ
θ is the angle between them.


```
public static double dotProduct(double[] a, double[] b) {
    double result = 0;
    for (int i = 0; i < a.length; i++) {
        result += a[i] * b[i];
    }
    return result;
}
```

Use Cases in Graphics: illumination - To calculate the cosine of the angle between light direction and surface normal to determine shading. Reflection - Finding the reflection direction of a vector off a surface.

Cross product = a × b = - b x a = |a| |b| sin θ



Use Cases in Graphics: Surface Normal - For computing the normal vector for shading and lighting calculations. Rotation - Determining rotation axes and generating perpendicular vectors.

```
public static double[] crossProduct(double[] a, double[] b) {
    double[] result = new double[3];
    result[0] = a[1] * b[2] - a[2] * b[1];
    result[1] = a[2] * b[0] - a[0] * b[2];
    result[2] = a[0] * b[1] - a[1] * b[0];
    return result;
}
```

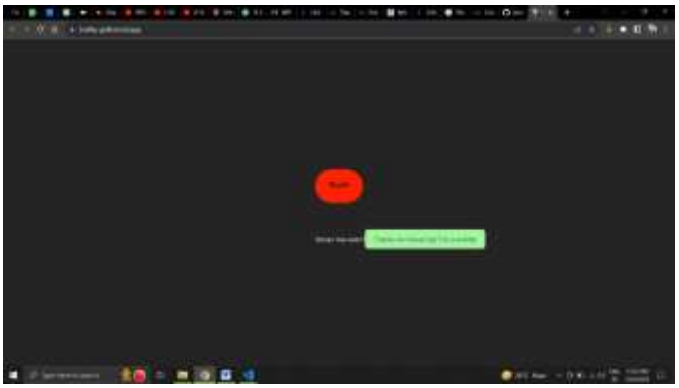
A plane is defined by the equation: Ax + By + Cz + D = 0, A ray is defined by: R0 = [X0, Y0, Z0] .
so R(t) = R0 + t * Rd , t > 0

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.

⇨ tooltip-gold.vercel.app
In this resent project I use this code.

```
  const [pos, setPos] = useState("left");
  const positionChanger = () => {
    if (pos === "left") {
      setPos("top");
    } else if (pos === "top") {
      setPos("bottom");
    } else if (pos === "bottom") {
      setPos("right");
    } else {
      setPos("left");
    }
  };
const Tooltip = ({ text, pos, children }) => {
  const [tooltip, setTooltip] = useState(false);
  return (
    <>
      <div
        className="tool"
        onMouseEnter={() => setTooltip(true)}
        onMouseLeave={() => setTooltip(false)}
      >
        {children}
        {tooltip && <span className={`tooltip ${pos}`}>{text}</span>}
        {tooltip && <span className={` ${pos} t-${pos} t-
arrow`}></span>}
```

```
    </div>
   </>
  );
};
```



This code is a React component that implements a simple Tooltip. The Tooltip displays additional information when the user hovers over a specified area (trigger element). The position of the Tooltip can be changed among four options: "left," "top," "right," and "bottom." The positionChanger function is used to cycle through these positions.

The useState hook is used to manage the state of the Tooltip's position (pos). The initial position is set to "left."
Another state, tooltip, is used to control whether the Tooltip should be displayed or not. It is initially set to false.
The function cycles through the positions: "left" ➡ "top" ➡ "bottom" ➡ "right" ➡ "left."

h) Random crashes – you are given a source code to test and it randomly crashes and it never crashes in 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.

Bonus – The deeper you go into computer architecture and explain, better.

⇨ The Memcheck part of the package is probably the place to start.
After that,
Checking the code:
The application may be using some random variable, like random number generator or a specific time of the day or a user input etc which may be causing this.
May be there is some uninitialized variable, which takes an arbitrary value each time and those values are causing such drastic behavior.
The program may have run out of memory, maybe heap overflow or something.
The program may depend on some other application which is causing this.
Maybe there are some other processes, running on machine causing it.
Referance:
https://www.reddit.com/r/embedded/comments/bpp8kt/how_to_debug_random_crashes/?rdt=50182