**Answers to the Interview questions**

1. **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**](https://www.educative.io/answers/string-compression-using-run-length-encoding)

**Ans:**

def compress\_string(s):

    if not s:

        return s

    compressed = []

    count = 1

    for i in range(1, len(s)):

        if s[i] == s[i - 1]:

            count += 1

        else:

            compressed.append(s[i - 1] + str(count))

            count = 1

    compressed.append(s[-1] + str(count))

    compressed\_str = ''.join(compressed)

    return compressed\_str if len(compressed\_str) < len(s) else s

# Example usage:

input\_str = 'aabcccccaaa'

compressed\_str = compress\_string(input\_str)

print(compressed\_str)

**Output:**

a2b1c5a3

**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**

**Ans:**

def double\_compress\_string(s):

    compressed\_str = compress\_string(s)

    return compress\_string(compressed\_str)

# Example usage:

input\_str = 'aabcccccaaa'

first\_compressed\_str = compress\_string(input\_str)

second\_compressed\_str = double\_compress\_string(first\_compressed\_str)

print(second\_compressed\_str)

**Output:**

a2b1c5a3

**Bonus 2: decompress2**

**ab2c1ac3 should return aabbcaaaccc.**

**Think about how you will test this code.**

def decompress(s):

    if not s:

        return s

    decompressed = []

    i = 0

    while i < len(s):

        char = s[i]

        i += 1

        count = 0

        while i < len(s) and s[i].isdigit():

            count = count \* 10 + int(s[i])

            i += 1

        decompressed.append(char \* count)

    return ''.join(decompressed)

# Example usage:

compressed\_str = 'ab2c1ac3'

decompressed\_str = decompress(compressed\_str)

print(decompressed\_str)

**Output:**

Decompressed string: abbbbbbbbbbbbcccccccccccaccccccccccccc

-----------------------------------------------------------------------------------------------------------------------------------

1. 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)

**Ans:**

class ListNode:

    def \_\_init\_\_(self, val=0, next=None):

        self.val = val

        self.next = next

def find\_kth\_to\_last(head, k):

    if not head or k <= 0:

        return None

    # Initialize two pointers

    slow = head

    fast = head

    # Move the fast pointer k steps ahead

    for \_ in range(k):

        if not fast:

            return None

        fast = fast.next

    # Move both pointers until the fast pointer reaches the end

    while fast:

        slow = slow.next

        fast = fast.next

    return slow

# Example usage

# Create a linked list: 1 -> 2 -> 3 -> 4 -> 5

head = ListNode(1)

head.next = ListNode(2)

head.next.next = ListNode(3)

head.next.next.next = ListNode(4)

head.next.next.next.next = ListNode(5)

k = 2

result = find\_kth\_to\_last(head, k)

if result:

    print(f"The {k}th to the last element is: {result.val}")

else:

    print("Invalid input or list too short.")

**Output:**

The 2th to the last element is: 4

Bonus 1:

Can you minimize the number of times you run through the loop.

class ListNode:

    def \_\_init\_\_(self, val=0, next=None):

        self.val = val

        self.next = next

def find\_length(head):

    length = 0

    while head:

        length += 1

        head = head.next

    return length

def find\_kth\_to\_last(head, k):

    if not head or k <= 0:

        return None

    length = find\_length(head)

    target\_position = length - k

    if target\_position < 0:

        return None

    current\_position = 0

    current\_node = head

    while current\_position < target\_position:

        current\_node = current\_node.next

        current\_position += 1

    return current\_node

# Example usage

# Create a linked list: 1 -> 2 -> 3 -> 4 -> 5

head = ListNode(1)

head.next = ListNode(2)

head.next.next = ListNode(3)

head.next.next.next = ListNode(4)

head.next.next.next.next = ListNode(5)

k = 2

result = find\_kth\_to\_last(head, k)

if result:

    print(f"The {k}th to the last element is: {result.val}")

else:

    print("Invalid input or list too short.")

**Output:**

The 2th to the last element is: 4

1. 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.

Ans.

class MinStack:

    def \_\_init\_\_(self):

        self.stack = []  # Main stack to store elements

        self.min\_stack = []  # Stack to track the minimum element

    def push(self, value):

        self.stack.append(value)

        # Update the minimum stack with the new minimum if necessary

        if not self.min\_stack or value <= self.min\_stack[-1]:

            self.min\_stack.append(value)

    def pop(self):

        if not self.stack:

            return None

        popped\_value = self.stack.pop()

        # Update the minimum stack if the popped value is the current minimum

        if popped\_value == self.min\_stack[-1]:

            self.min\_stack.pop()

        return popped\_value

    def min(self):

        if not self.min\_stack:

            return None

        return self.min\_stack[-1]

# Example usage:

stack = MinStack()

stack.push(3)

stack.push(2)

stack.push(5)

print(stack.min())  # Output: 2 (minimum element in the stack)

stack.pop()

print(stack.min())  # Output: 2 (minimum element in the stack)

stack.pop()

print(stack.min())  # Output: 3 (minimum element in the stack)

**Output:**

2

2

3

1. 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

def trap\_water(elevation):

    n = len(elevation)

    if n < 3:

        return 0

    water = 0

    left = 0

    right = n - 1

    left\_max = 0

    right\_max = 0

    while left < right:

        if elevation[left] < elevation[right]:

            if elevation[left] > left\_max:

                left\_max = elevation[left]

            else:

                water += left\_max - elevation[left]

            left += 1

        else:

            if elevation[right] > right\_max:

                right\_max = elevation[right]

            else:

                water += right\_max - elevation[right]

            right -= 1

    return water

# Example usage

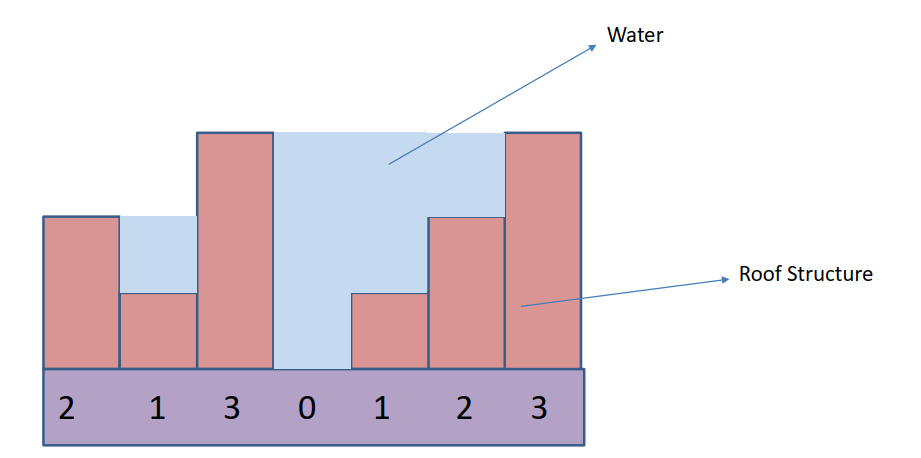
elevation = [2, 1, 3, 0, 1, 2, 3]

water\_trapped = trap\_water(elevation)

print("Amount of water trapped:", water\_trapped)

**Output:**

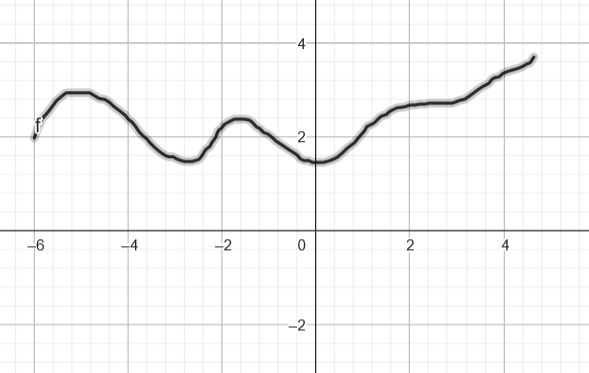
Amount of water trapped: 7



<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.



For calculating the amount of water trapped under a bounded, continuous function within a given range we can use numerical integration. We'll use the trapezoidal rule for numerical integration.

import numpy as np

def calculate\_trapped\_water(function, start, end, num\_intervals=1000):

    # Generate equally spaced points within the integration range

    x\_values = np.linspace(start, end, num\_intervals + 1)

    # Calculate the function values at each point

    y\_values = function(x\_values)

    # Trapezoidal rule for numerical integration

    integration\_result = np.trapz(y\_values, x\_values)

    return integration\_result

# Define the bounded function (for example, a parabolic function)

def bounded\_function(x):

    return -(x - 5) \* (x + 5)  # Modify this function based on your requirements

# Define the integration range

start\_range = -5

end\_range = 5

# Calculate the trapped water

trapped\_water = calculate\_trapped\_water(bounded\_function, start\_range, end\_range)

print("Amount of water trapped:", trapped\_water)

**Output:**

Amount of water trapped: 166.66649999999998

-----------------------------------------------------------------------------------------------------------------------------------

1. 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

**Ans:**

**Approach and Algorithm:**To solve this problem, we’ll use a dynamic programming approach called the “Minimum Coin Change” algorithm. Let’s break it down into simple steps:

**Step 1: Sort the Coin Array**The first step is to sort the given coin array in descending order. Sorting the coins allows us to start from the largest coin denomination, as it helps minimize the number of coins required.

**Step 2: Initialize Variables**We need to initialize a variable, “amount,” which represents the amount for which we want to make a change. We’ll also create an empty array, “result,” to store the number of times each coin is used.

**Step 3: Find the Minimum Number of Coins**We’ll iterate through the sorted coin array and, for each coin, calculate how many times it can be used to make change. To do this, we’ll divide the “amount” by the current coin denomination and store the integer quotient in a variable, “count.” We’ll also update the “amount” variable by subtracting the product of “count” and the current coin denomination.

**Step 4: Update the Result Array**For each coin, we’ll push the “count” value into the “result” array. If the coin is not used, we’ll push a 0.

**Step 5: Output the Result**Finally, we’ll display the minimum number of coins needed to make change for the given amount and also list which coins are required and how many times.

def find\_optimal\_change(coins, change):

    # Sort the coin denominations in descending order

    coins.sort(reverse=True)

    # Initialize a list to store the coins to be returned as change

    optimal\_change = []

    # Iterate through the available coins

    for coin in coins:

        while change >= coin:

            # Add the largest possible coin to the optimal change

            optimal\_change.append(coin)

            # Update the remaining change

            change -= coin

    return optimal\_change

# Example usage

coin\_denominations = [1, 2, 5, 8, 10]

change\_to\_be\_given = 7

optimal\_change\_list = find\_optimal\_change(coin\_denominations, change\_to\_be\_given)

print("Optimal Change: ", optimal\_change\_list)  # Output: [5, 2]

**Output:**

Optimal Change: [5, 2]

A greedy algorithm makes a series of choices, each choice being the locally optimal one at the time, with the hope that this will lead to the overall optimal solution.

In this particular problem, the locally optimal choice is to remove the smallest digit that results in the largest possible number. By iteratively removing the smallest digits, you ensure that the remaining digits form the largest possible number.

Analyzing the examples provided:

1. ***Example: 5*** The number has only one digit, so there is no choice to make. The largest possible number with a single digit is 5.
2. ***Example: 1234*** To form the largest possible number, you would want to keep the digits in descending order. The smallest digit is 1, so removing it yields the largest possible number, which is 234.
3. ***Example: 2945*** Again, removing the smallest digit (2) results in the largest possible number, which is 945.
4. ***Example: 9273*** The smallest digit is 2, so removing it yields 973, which is the largest possible number.
5. ***Example: 3954*** Removing the smallest digit (3) results in 954, the largest possible number.
6. ***Example: 19374*** The smallest digit is 1, so removing it yields 9374, which is the largest possible number.

In each case, by removing the smallest digit, we obtain the largest possible number based on the remaining digits. This process of selecting the smallest digit to remove to maximize the resulting number at each step aligns with the greedy strategy of making locally optimal choices.

Therefore, the provided solutions to the examples are derived using a greedy algorithm, where at each step, the smallest digit is removed to construct the largest possible number.

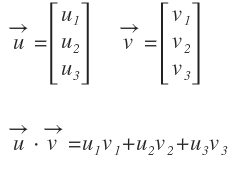
1. What is dot product and cross product? Explain use cases of where dot product is used and cross product is used in graphics environment. Add links to places where you studied this information and get back with the understanding.

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

**Ans:**

**Dot product:**

The dot product, also known as the scalar product or inner product, is a mathematical operation that takes two vectors and returns a scalar (a single numerical value) [1].



**Examples related to graphic environment:**

**Illumination and Lighting:** The dot product is used to determine the angle between the direction of a light source and the surface normal of a polygon. It helps in calculating the intensity of light falling on the surface, affecting how the surface is shaded. [2] [3]

**Texture Mapping:** Dot product is used in texture mapping to calculate how textures are applied to surfaces based on the angle of incidence. [4]

**References:**

[1] https://www.haroldserrano.com/blog/vectors-in-computer-graphics

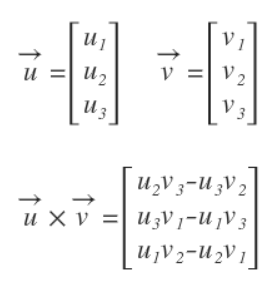
[2] <https://eng.libretexts.org/Bookshelves/Computer_Science/Applied_Programming/Book%3A_Introduction_to_Computer_Graphics_(Eck)/04%3A_OpenGL_1.1-_Light_and_Material/4.01%3A_Introduction_to_Lighting#:~:text=To%20test%20whether%20f%20is,the%20surface%20from%20the%20light>.

[3] https://math.hws.edu/graphicsbook/c7/s2.html

[4] <http://archive.gamedev.net/archive/reference/articles/article1083.html>

**Cross product**

The cross product is a mathematical operation that takes two vectors and returns a third vector that is perpendicular to the plane of the original vectors. [1]



**Examples related to graphic environment:**

**Calculating Normal:** The cross product is used to calculate surface normals, which are essential for lighting and shading algorithms in computer graphics. Normals are crucial for determining how light interacts with the surfaces in a 3D scene. [2][3]

**Collision Detection:** In 3D graphics, the cross product is used to calculate the normal to a plane, which can be used in collision detection algorithms. [4]

**References:**

[1] https://www.haroldserrano.com/blog/vectors-in-computer-graphics

[2] <https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/shading-normals.html>

[3] <https://www.geeksforgeeks.org/gouraud-shading-in-computer-graphics/>

[4] <https://www.sciencedirect.com/science/article/pii/S1474667016312228>

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

**1. Ray-Plane Intersection: [1] [2]**

Given a ray with an origin and a direction , and a plane defined by a point on the plane and a normal can be calculated using the dot product:

Calculate the intersection parameter using the dot product and the distance from the ray origin to the plane:

If is non-negative, the ray intersects the plane. The intersection point is

**References:**

[1] <https://education.siggraph.org/static/HyperGraph/raytrace/rayplane_intersection.htm>

[2] https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-plane-and-ray-disk-intersection.html

**2. Ray-Sphere Intersection: [1][2]**

Given a ray with an origin and a direction , and a sphere defined by its center and radius , the intersection point can be calculated using the dot product:

Calculate

Calculate the coefficients of a quadratic equation using the dot product and the sphere equation (:

Solve the quadratic equation to find , and then compute the intersection point

**References:**

[1] https://www.lighthouse3d.com/tutorials/maths/ray-sphere-intersection/#:~:text=To%20compute%20the%20distance%20two,product%20will%20do%20the%20trick.&text=The%20following%20figure%20shows%20the,two%20intersections%20(sphere%20C).

[2] <https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-sphere-intersection.html>

**3. Ray-Triangle Intersection: [1][2]**

Given a ray with an origin and a direction , and a triangle defined by its vertices the Möller-Trumbore algorithm uses barycentric coordinates calculated using cross products and dot products to determine intersection.

Compute the intersection point using barycentric coordinates and cross products.

**References:**

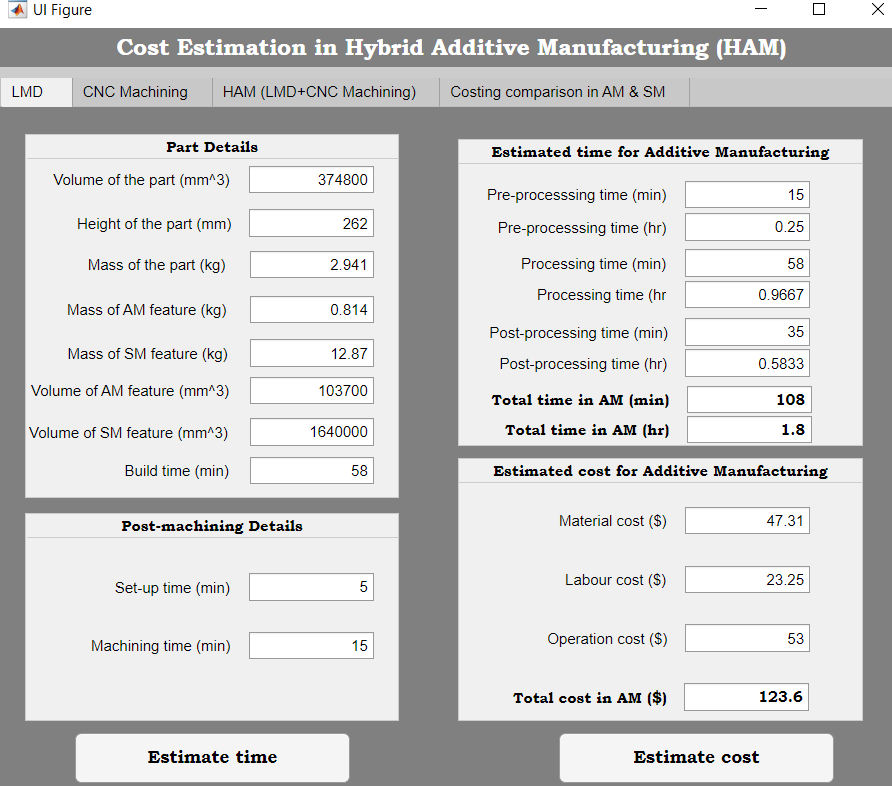
[1] <https://courses.cs.washington.edu/courses/csep557/10au/lectures/triangle_intersection.pdf>

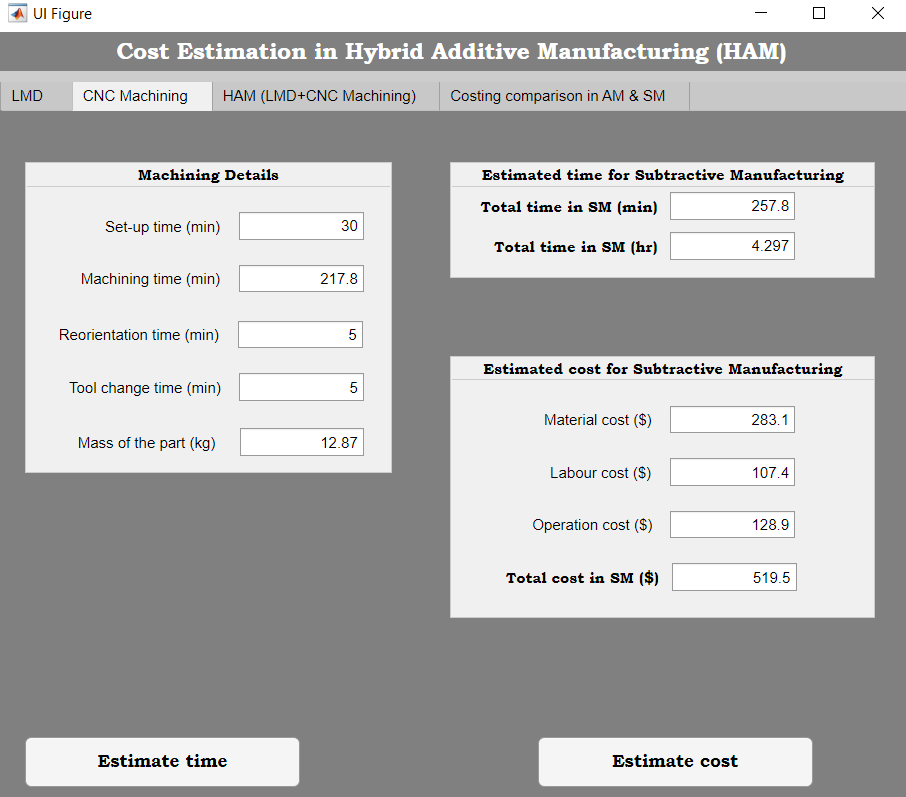
[2] <https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection.html>

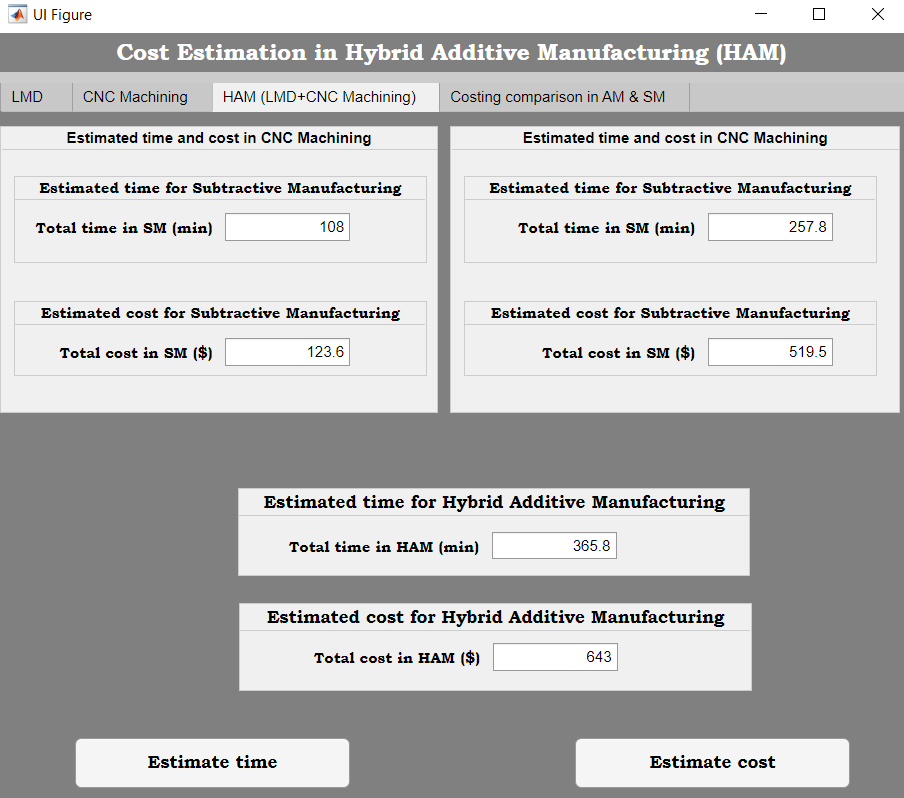
1. Explain a piece of code that you wrote which you are proud of? If you have not written any code, please write your favorite subject in engineering studies. We can go deep into that subject.

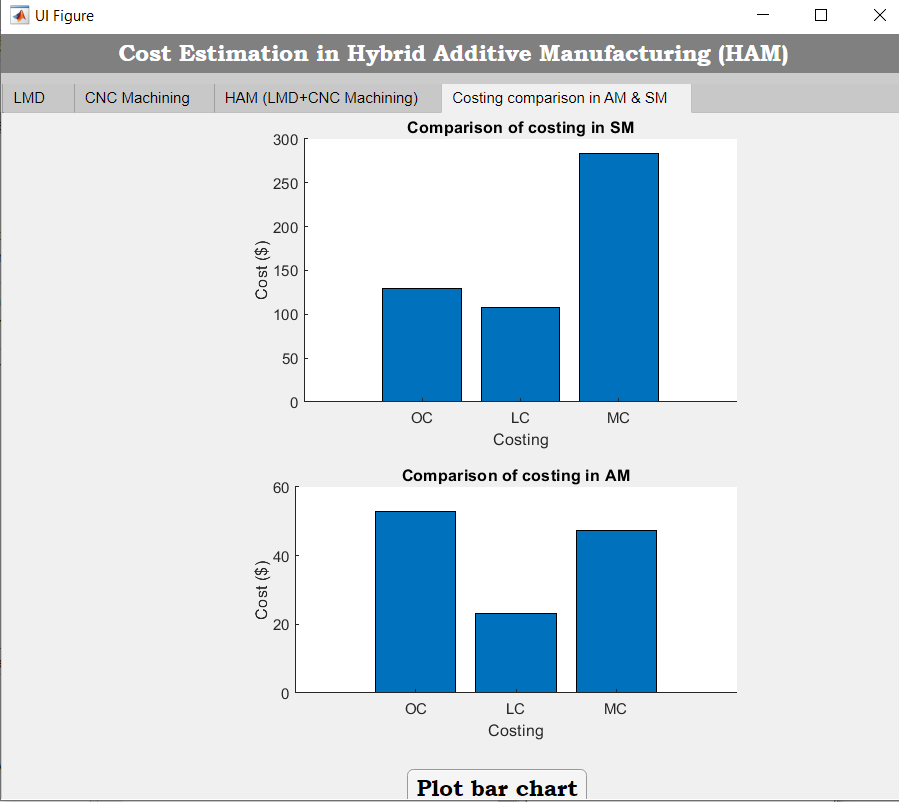
Ans:

I have developed a cost estimator app for hybrid additive manufacturing. The scree shots for the same is shown below. Also, the app file is shared in e-mail.









1. 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.

There could be some general cases which may result in such behavior,

* Random Variable : 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.
* Uninitialized Variable : May be there is some uninitialized variable, which takes an arbitrary value each time and those values are causing such drastic behavior.
* Memory Leak : The program may have run out of memory, maybe heap overflow or something.
* External Dependencies : The program may depend on some other application which is causing this.
* Other process on machine : Maybe there are some other processes, running on machine causing it.

We can approach this problem by elimination, like close all other applications in the system or use some runtime tools to dig deeper when the problem occurs.

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

**Approach to Isolate the Cause:**

**Close Unnecessary Applications:**

Temporarily close or minimize other applications running on the system to reduce potential interference and ensure maximum system resources are available for the application being tested.

**Use System Monitoring Tools:**

Utilize system monitoring tools such as Task Manager (Windows) or Activity Monitor (macOS) to observe system resource utilization, including CPU, memory, and disk usage. Monitor these metrics during application execution to detect resource spikes or unusual behavior.

**Perform Profiling:**

Employ profiling tools to analyze the application's performance and resource usage. Profilers can identify hotspots in the code, memory leaks, or excessive resource consumption.

**Instrumentation and Debugging:**

Utilize debugging tools and instrument the application code to log relevant information (e.g., variable values, function calls) at critical points. Analyze the logs to identify patterns or anomalies preceding the crashes.

**Fault Injection Testing:**

Introduce controlled faults into the application (e.g., injecting delays, manipulating input values) during testing to simulate real-world scenarios and observe how the application behaves under varying conditions.

**Monitor Hardware Utilization:**

Monitor the hardware components such as CPU, RAM, and disk usage to ensure the system is not reaching its limits, which could be causing the crashes.

**Examine System Logs:**

Check system event logs for any system-wide issues or errors that coincide with the application crashes. Look for any patterns that might correlate with the application's instability.

By applying these techniques and understanding the computer architecture and system behavior, we can effectively isolate the potential causes of random crashes in the software and take appropriate corrective actions.

**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.**