Name of 1st Member : Majid Rahman

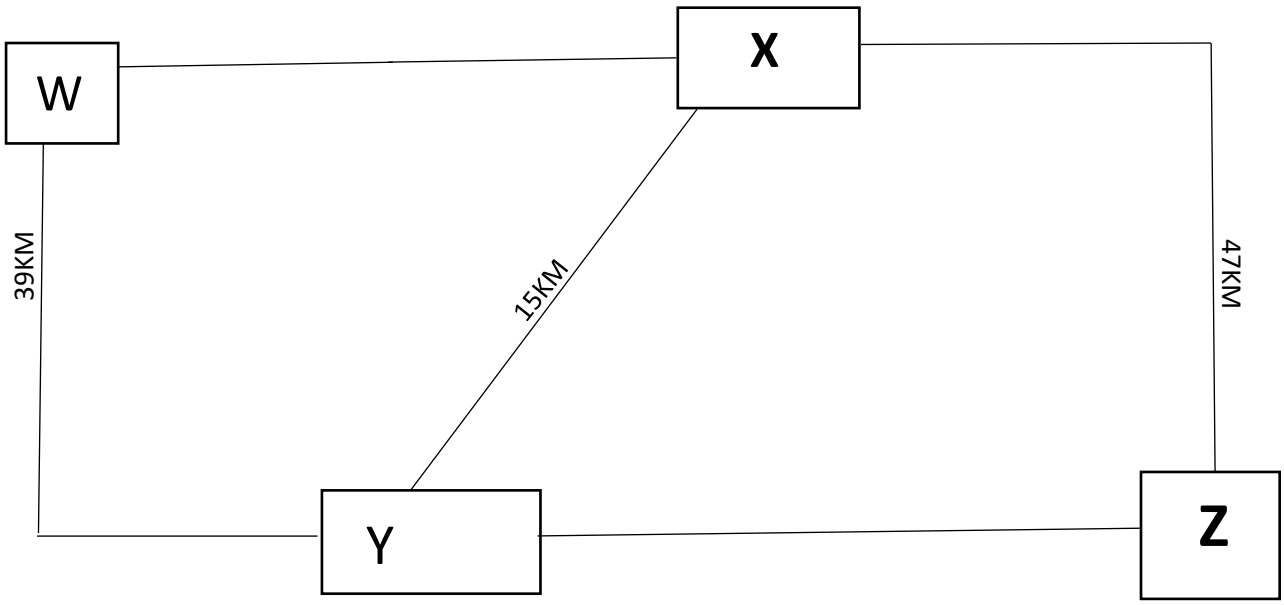
Enrollment No : 01-131232-041

Name of 2nd Member : Huzaifa Abbasi

Enrollment No : 01-131232-103

Class : BSE1(A)

TASK 1:



Step1: Start

Step2: Chose the start and destination location

Step3: Find all the roots connect to start and destination location

Step4: Find distances of each route

Step5: Compare distances of each route

Step6: You will get Shortest Location

Step7: End

;

Task 2 :

Sorting a List of Numbers

Algorithm:

Let Suppose our List is (76,4,91,12,2,11,3,1,24,84)

Step 1: Start

Step 2: Enter the List of Numbers. (76,4,91,12,2,11,3,1,24,84) {Input}

Step 3: Finding Ascending Order through **Quick sorting method.** {Function}

Step 4: Print the Array. {Output}

Step 5: End.

Explanation:

Different Sorting method.

Quick Sorting:

- Sort Quick is the best choice for this list.it has an average case time complexity of O (n log n) and usually perform well for various types of data.
- Quick Sort is particularly efficient when the list is relatively large, and its worst-case scenario is ess likely to occur on random input list.

Merge Sort:

- Merge Sort is also a good option.it consistently has a time complexity of O(n log n),making it efficient for sorting lists, regardless of the initial order.
- Merge Sort is stable, meaning it preserves the relative order of equal elements, which can be a benefit in some scenarios.

Bubble method:

- Bubble Sort is not recommended for this list, especially if the list is large. It has a worst-case time complexity of O (n^2) and is generally less efficient than Quick Sort or Merge Sort.

Counting Sorting:

- Counting Sort is not suitable for this list because it's designed for lists with small integer values and a limited range. The range in this list is relatively large, so Counting Sort isn't appropriate.

So, the best choice for Given List is **Quick Sorting**. As it will be more efficient and versatile for various input scenarios.

Task 3:

Step1: Start

Step2: Set 2 variables and assign them values like num1=0, num2=1 (first two Fibonacci Numbers).

Step3: Input number (required nth term of Fibonacci sequence) to find the nth Fibonacci number.
[Input]

Step4: As we are having the first two Fibonacci numbers, we will be using a loop to calculate the required Fibonacci number and the loop will be started from 3 to the number (nth term).
[Step4 -Step8=Processing]

Step5: Now we will calculate the Fibonacci number by adding num1 and num2, e.g., value=num1+num2

Step6: Now we will swap the values e.g., num1=num2

Step7: Then swap the value num2=value

Step8: The loop (Step5 – Step7) will be working until the desired nth Fibonacci number is calculated.

Step9: The final number which will be printed is num2 which will be the required nth Fibonacci number.
[Output]

Step10: Stop

Task 4:

Step1: Start

Step2: We will be creating an inventory list which will include the items already there and a place for new entries.

Step3: Input will be taken from the user and see if the item already exists then update the quantity otherwise add the item to the list. [Input]

Step4: If an item is having quantity '0' or has reached a particular quantity limit, then you have two options, whether you want to remove it from the list or re-order it. [Processing]

Step5: The report of the updated items list will be generated and printed on the screen.
[Output]

Step6: Stop.