Task 6.1P

Name- Anneshu Nag
Student ID- 2210995760

1. Design a Θ (n log n) time algorithm that, given a set S of n integer numbers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x.

<u>Ans:</u> Steps to achieve the required solution:

1. **First of all, sort all the elements in the array using either Merge Sort or Quick Sort as they have average time complexity of Θ (n log n).**

2. **To find the actual sum of two elements, do the following-**
    - **Take two pointers at the lowest index (say lowerIdx) and at the highest index (higherIdx).**

    - **Traverse the array and check the conditions that are required i.e. if the sum of element at the 'lowerIdx' and the element at the 'higherIdx' is equal to x then return true.**

      If their sum is less than x then shift the 'lowerIdx' pointer to the right side i.e. 'lowerIdx++'.

      If their sum is greater than x then shift the 'higherIdx' pointer to the left side i.e. 'higherIdx--'

      Finally, if even after the traversal is complete and no pair of elements has the sum of x then return false.

Code for the given explaination-

```
/* This code is made by Anneshu Nag, Student ID- 2210994760  */
/*                    Dated- 25/09/2023                       */

public bool SumOfElements (int[] array, int x)
{
    int lowerIdx = 0;
    int higherIdx = array.Length - 1;

    // Sort the array first using either Merge or Quick Sort
    MergeOrQuickSort(array);

    // Traverse the sorted array and check for the conditions explained
    while (lowerIdx < higherIdx)
    {
        if (array[lowerIdx] + array[higherIdx] == x)
        {
            return true;
        }
        else if (array[lowerIdx] + array[higherIdx] < x)
        {
            lowerIdx++;
        }
        else //  For the condition where sum is greater than x
        {
            higherIdx--;
        }
    }
    return false;
}
```

2. A Stack data structure provides Push and Pop, the two operations to write and read data, respectively. Using the Stack as a starting point design a data structure that, in addition to these two operations, also provides the Min operation to return the smallest element of the stack. Remember that the new data structure must operate in a constant Θ (1) time for all three operations.

Ans: Steps for the implementation of the data structure required-

1. **I have created a 'MinimumStackImplementation' class where I have first declared to private Stack 'myStack' to store the integer elements and another private Stack 'minEleStoreStack' to track the minimum element.**

2. **The Push method will push the 'pushVal' to the main stack, it will push element to the 'minEleStoreStack' if it the smallest element and is also smaller than the element present in the 'minEleStoreStack'. Also, it will store the element to both the stacks if it is the first element.**

   It is working in O(1) complexity.

3. **The Pop method will first check if the stack is empty or not. If it is not empty then it will remove the latest element i.e. the topmost element of the stack. If the removed element from the main stack is equal to the topmost element in the 'minEleStoreStack' then remove it too.**

   It is working in O(1) complexity.

4. **Finally the 'MinimumElement' method will return the latest minimum element in the when it is called.**

   It is working in O(1) complexity.

```csharp
/* This code is made by Anneshu Nag, Student ID- 2210994760  */
/*                      Dated- 25/09/2023                    */

/* Implementation of the data structure */
public class MinimimStackImplementation
{
    private Stack<int> myStack;
    //Another stack for storing the minimum element
    private Stack<int> minEleStoreStack;

    public MinimimStackImplementation()
    {
        myStack = new Stack<int>();
        minEleStoreStack = new Stack<int>();
    }

    /* Implementation of Push method */
    public void Push(int pushVal)
    {
        myStack.Push(pushVal); // Insert Element to Stack

        // Push to the minEleStoreStack if it is the smallest element
        if (minEleStoreStack.Count == 0)
        {
            minEleStoreStack.Push(pushVal);
        }
        else if (minEleStoreStack.Peek() >= pushVal)
        {
            minEleStoreStack.Push(pushVal);
        }
    }

    /* Implementation of the Pop method */
    public int Pop()
    {
        // Error if no elements in the Stack to pop
        if (myStack.Count == 0)
        {
            throw new InvalidOperationException("Stack Underflow : Empty!");
        }

        // Removing element from the Stack i.e. the topmost element
        int popFromStack = myStack.Pop();

        // If the popped element is the equal to the element in
minEleStoreStack then remove it too
        if (popFromStack == minEleStoreStack.Peek())
        {
            minEleStoreStack.Pop();
        }

        //Return the popped element
        return popFromStack;
    }
    /* Finding the minimum element*/
    public int MinimumElement()
    {

        if (minEleStoreStack.Count == 0)
        {
            throw new InvalidOperationException("Stack Underflow : Empty!");
        }
        // The top element of the minEleStoreStack will contain the minimum
element
        return minEleStoreStack.Peek();
    }
}
```