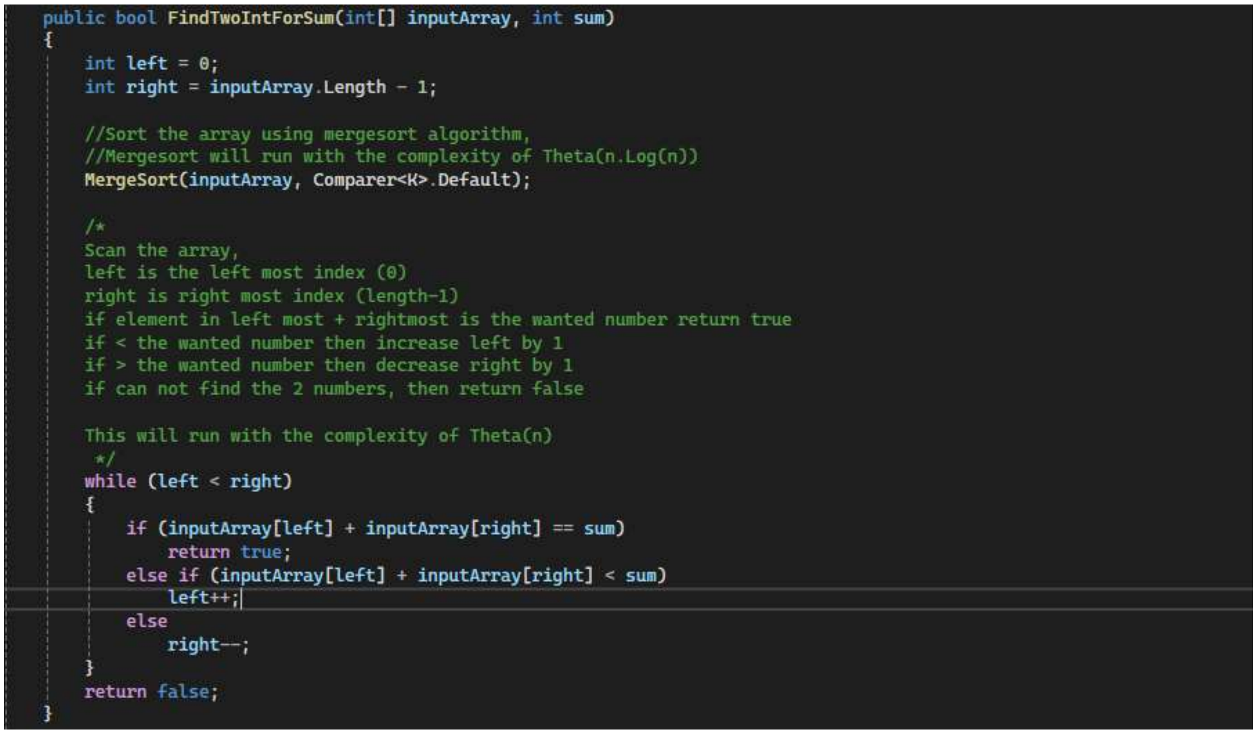
Task 6.1C

Answer the Questions

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
   - Utilizing merge sort, order the array
   - Utilize the “low” and “high” pointers, allocating them to the array’s first and last elements.
   - Examine the array one by one.
     - ✓ Return true if the desired number (x) is equal to the elements at “low” plus the elements at “high”
     - ✓ In the event where the sum of the elements at “low” and “high” is less than x, move the “low” pointer to the right by (+1)
     - ✓ Otherwise, move the “high” pointer to the left (-1) if the sum of the elements at “low” and “high” is greater than x
   - Return false if the array has been scanned and no pair of elements is found to be eligible.



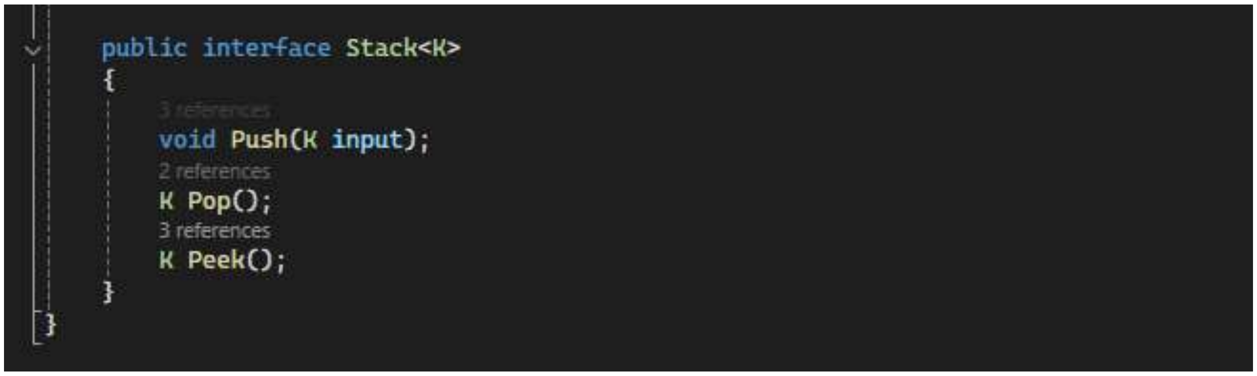
```
public bool FindTwoIntForSum(int[] inputArray, int sum)
{
    int left = 0;
    int right = inputArray.Length - 1;

    //Sort the array using mergesort algorithm,
    //Mergesort will run with the complexity of Theta(n.Log(n))
    MergeSort(inputArray, Comparer<K>.Default);

    /*
    Scan the array,
    left is the left most index (0)
    right is right most index (length-1)
    if element in left most + rightmost is the wanted number return true
    if < the wanted number then increase left by 1
    if > the wanted number then decrease right by 1
    if can not find the 2 numbers, then return false

    This will run with the complexity of Theta(n)
     */
    while (left < right)
    {
        if (inputArray[left] + inputArray[right] == sum)
            return true;
        else if (inputArray[left] + inputArray[right] < sum)
            left++;
        else
            right--;
    }
    return false;
}
```

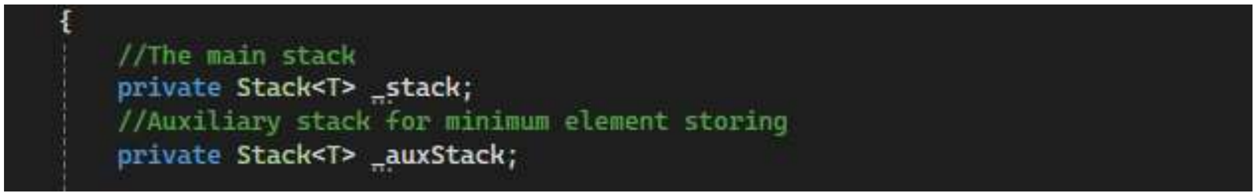
2. A Stack data structure provides Push and Pop, the two operations to write and read data, respectively. Using the Stack as a starting point design a data structure that, in addition to these two operations, also provides the Min operation to return the smallest element of the stack. Remember that the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



```
public interface Stack<K>
{
    void Push(K input);
    K Pop();
    K Peek();
}
}
```

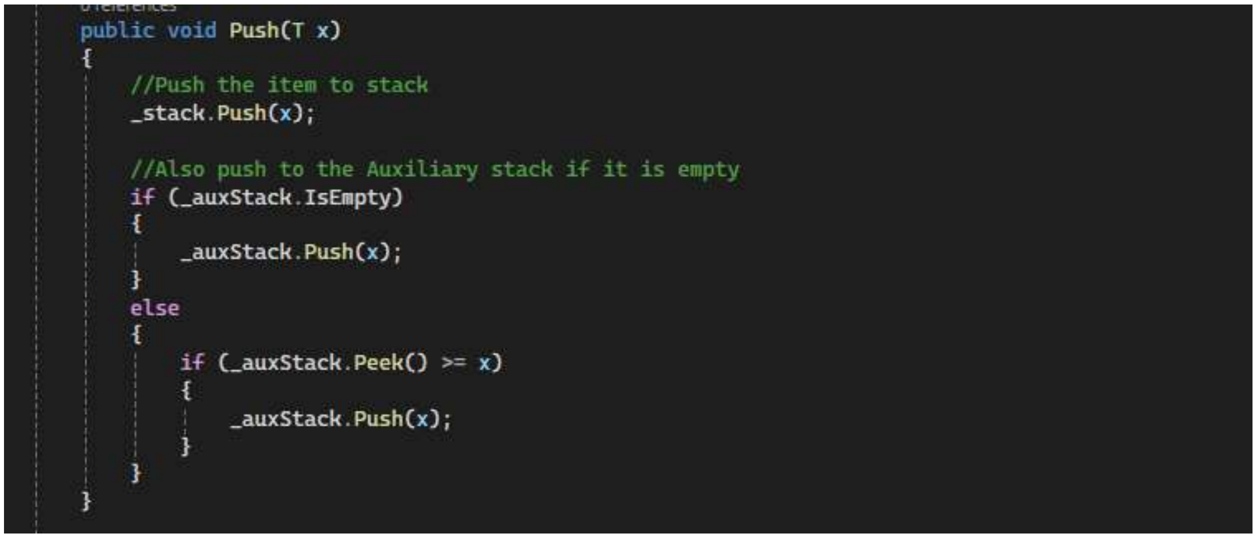
   - This interface has also been implemented by the "Stack" class, which also has attributes like "IsEmpty" and so forth.

   - Two distinct stacks objects are needed in order to create a stack data structure that satisfies the requirement:



```
{
    //The main stack
    private Stack<T> _stack;
    //Auxiliary stack for minimum element storing
    private Stack<T> _auxStack;
```

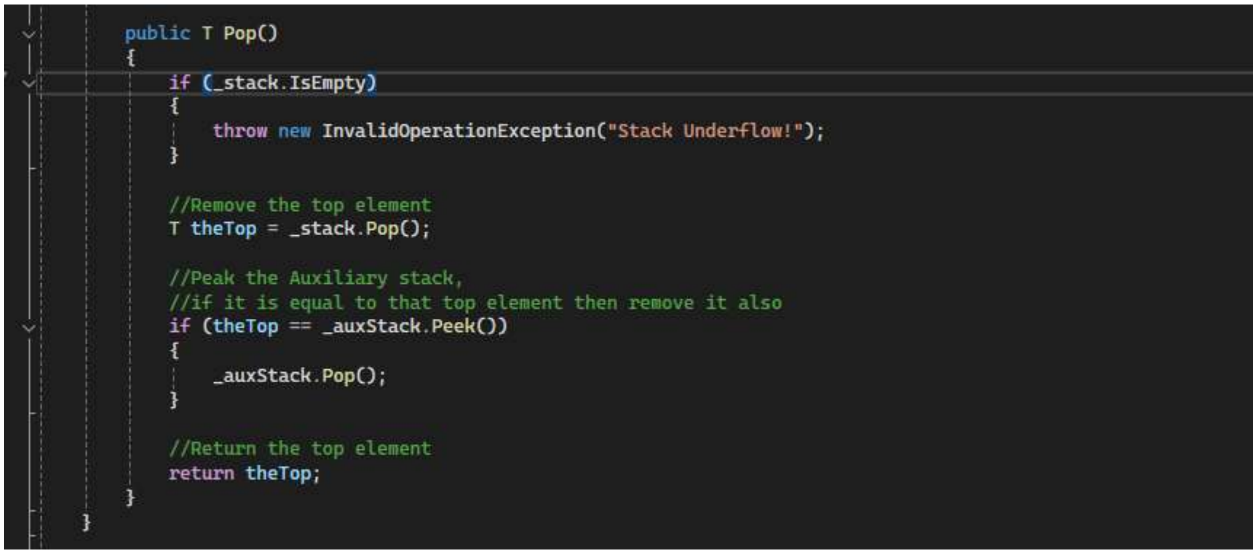
   - If the most recent element in the targeted item's stack is equal to or greater than the new element, the Push method must push it to both the main and auxiliary stacks:



```
public void Push(T x)
{
    //Push the item to stack
    _stack.Push(x);

    //Also push to the Auxiliary stack if it is empty
    if (_auxStack.IsEmpty)
    {
        _auxStack.Push(x);
    }
    else
    {
        if (_auxStack.Peek() >= x)
        {
            _auxStack.Push(x);
        }
    }
}
```

- In order to remove the most recent element from the stack, the Pop method first verifies that the stack is not empty. If it is, the most recent element in the auxiliary will also be removed.



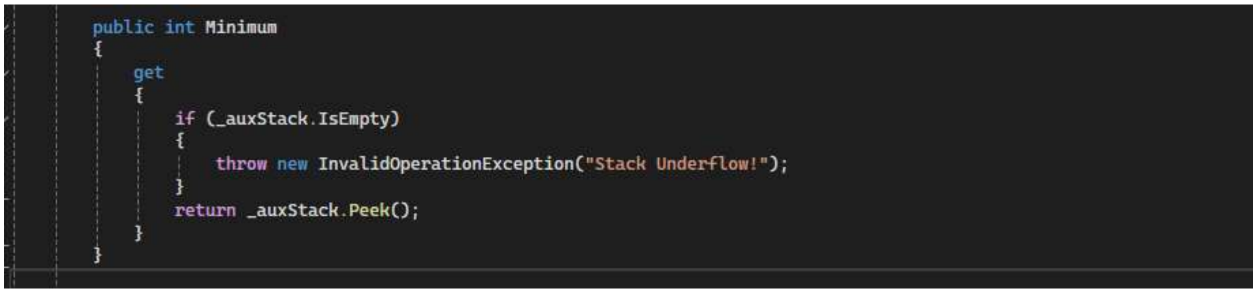
```
    public T Pop()
    {
        if (_stack.IsEmpty)
        {
            throw new InvalidOperationException("Stack Underflow!");
        }

        //Remove the top element
        T theTop = _stack.Pop();

        //Peak the Auxiliary stack,
        //if it is equal to that top element then remove it also
        if (theTop == _auxStack.Peek())
        {
            _auxStack.Pop();
        }

        //Return the top element
        return theTop;
    }
}
```

- Lastly, the Minimum property will ensure that there is an item in the Auxiliary stack by just peeking at the Auxiliary:



```
public int Minimum
{
    get
    {
        if (_auxStack.IsEmpty)
        {
            throw new InvalidOperationException("Stack Underflow!");
        }
        return _auxStack.Peek();
    }
}
```

- These processes will operate at a constant time complexity.