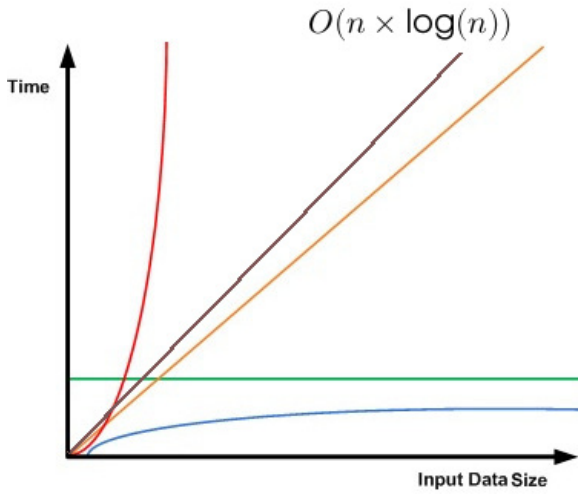
Instructions: This exam is to be taken in silence, without notes, books, or electronic devices (including "smart" watches or earbuds). The time limit to complete it is the duration of the class period (1 hour and 15 minutes). Answer the following questions and problems, trying to be as clear and as accurate as possible. Take your time to read the statements carefully before trying to answer them. If you need more space, write on the back of your test page and indicate it clearly. When writing code, make sure your special punctuation characters are legible, and your lowercase and uppercase letters are easy to distinguish. As usual, every statement or series of statement is assumed to be in a valid class and method, and you can use the `C.W( )` and `C.WL( )` abbreviations.

____ / 15 pts. **Problem 1** Label each function in the graph below with their "big-O" notation. The function $O(n \times \log(n))$ is labelled as an example.



____ / 25 pts. **Problem 2** Consider the implementation of "custom" lists `CList` shared on page 7.

1. What would be displayed by the following?

```
CList<int> myList1 = new CList<int>( );
myList1.AddF(1);
myList1.AddF(5);
myList1.AddF(2);
myList1.AddF(6);
Console.WriteLine(myList1);
```

2. Write the `IsEmpty( )` method, that returns **`true`** if the calling object is an empty `CList`, **`false`** otherwise.

3. The following `RemoveF` method that removes the first element from a `CList` is incorrect. Explain why, and give a series of statements that would throw the exception.

```
public void RemoveF( )
{
    first = first.Next;
}
```

4. Write the `AddL` method, that adds the element passed as an argument at **the end** of the calling `CList` object.

```
public void AddL(T dataP){



}
```

5. Explain in brief terms what would need to change to make this `CList` class implements *doubly* linked list, and the benefits and drawback of doubly versus singly linked lists.

____ / 20 pts. **Problem 3** Complete the following (partial) implementation of stacks using arrays.

```
class CAStack<T>
{
    private T[] mArray;

    private int top = 0;   // Greatest index unoccupied by a value.

    public CAStack(int sizeP = 10)
    {
        if (sizeP <= 0)
            throw new ApplicationException("Size must be positive.");
        else
            mArray = new T[sizeP];
    }

    public bool IsEmpty()
    {









    }

    public void Push(T value)
    {









    }

    public T Pop()
    {









    }
}
```

____ / 25 pts. **Problem 4** Consider the implementation of "custom" queue `CQueue` shared on page 8.

1. Give the value of `front`, `end` and `size` after the following has been executed:

```
CQueue<int> myQueue1 = new CQueue<int>(3);
myQueue1.Enqueue(1);
myQueue1.Enqueue(2);
myQueue1.Enqueue(3);
```

2. After the following statement has been executed

```
Console.WriteLine("Value dequeued: " + myQueue1.Dequeue());
```

Give

(a) What would be displayed and

(b) the value of the `front`, `end` and `size`

3. Draw `myQueue1`'s `mArray` and indicate `front` and `end` values after `myQueue1.Enqueue(4);` has been executed.

4. Write a `Capacity` property to be inserted in our `CQueue` class, whose getter gives the number of empty slots in the queue.

____ / 30 pts. **Problem 5** Write a `Find` method for binary search trees, knowing that

- The `Node` class is implemented as usual, with a `T Data` property, and `Node left`, `Node right` attributes,
- The binary search tree's only attribute is `Node root`,
- `T` instantiates `IComparable<T>`, so `CompareTo` can be used.

Remember that you may need to implement helper method(s) to complete this task.

```
public bool Find(T dataP)
{




}
```

1 on dictionary, following mike,

____ / 25 pts. **Problem 7** Consider the following algorithm:

```
public static void Sort(List<T> listP)
{
    T current;
    int slot;
    for (int bar = 1; bar < listP.Count; bar++)
    {
        current = listP[bar];
        for (
            slot = bar;
            slot > 0 && current.CompareTo(listP[slot - 1]) < 0;
            slot--
        )
        {
            listP[slot] = listP[slot - 1];
        }
        listP[slot] = current;
        // Here
    }
}
```

1. Explain how it would operate by indicating how the following list would be updated each time `// Here` is reached with the following values for `bar`:

Index	0	1	2	3	4	5	6	7
Value	16	14	36	22	7	13	12	45
`bar` is 1								
`bar` is 2								
`bar` is 3								
`bar` is 4								
`bar` is 5								
`bar` is 6								
`bar` is 7								
`bar` is 8								

1. Give its best time complexity, with a description of list achieving this time.

2. Give its worst time complexity, with a description of list achieving this time.

```
public class CList<T>
{
    private class Cell
    {
        public T Data { get; set; }
        public Cell Next { get; set; }

        public Cell(T dataP, Cell nextP)
        {
            Data = dataP;
            Next = nextP;
        }
    }

    private Cell first;

    public CList()
    {
        first = null;
    }

    public void AddF(T dataP)
    {
        first = new Cell(dataP, first);
    }
}
```

```
class CQueue<T>
{
  private int front, end, size;

  private T[] mArray;

  public CQueue(int capacity = 10)
  {
    mArray = new T[capacity];
    front = 0;
    end = 0;
    size = 0;
  }

  public void Enqueue(T dataP)
  {
    mArray[end] = dataP;
    Increment(ref end);
    size++;
  }

  public T Dequeue( )
  {
    size--;
    T data = mArray[front];
    Increment(ref front);
    return data;
  }

  private void Increment(ref int value)
  {
    value++;
    value %= mArray.Length;
  }
}
```