Task 2.2C

Name- Anneshu Nag
Student ID- 2210994760

1. The last four digits of my roll number: **4760**

Algorithm	**Operations count per line**	**Total**
int count = 0;	1	**B = $3N + 4$**
for (int i = 0; i < N; i++) {	1 + (N + 1) + N	**W = $3N^3/2 + 7N^2/2 + 5N + 6$**
if (rand() < 0.5) {	N x (1)	**A = $3N^3/4 + 7N^2/4 + 4N + 5$**
for (int k = N; k > N/2; k--) {	N x (1 + ((N - N/2) + 1) + (N - N/2))	
count++;	N x (N - N/2) x (1)	
}		
}		
}		
if (rand() > 0.5) {	1	
int j = count;	1	
while(j > 0) {	(N) x (N - N/2) + 1	
for (int k = 0; k < N; k++){	(N) x (N - N/2) x (1 + (N + 1) + N)	
count++;	(N) x (N - N/2) x N x (1)	
}		
j--;	(N) x (N - N/2) x (1)	
}		
}		
int count = 0;	1	**B = $34N^3/2 – 31N^2/2 + 5N/2 + 2$**
for (int i = 0; i < N-1; i++) {	1 + (N - 1 + 1) + (N – 1)	**W = $34N^3/2 – 31N^2/2 + 5N/2 + 2$**
for (int j = i+1; j < N; j++) {	(N - 1) x (1 + (N/2 + 1) + N/2)	**A = $34N^3/2 – 31N^2/2 + 5N/2 + 2$**
count+=N;	(N – 1) x N/2	
}		
}		
int j = 0;	1	
int num = count;	1	
while(j < num){	N x ((N – 1) x N/2) + 1	
int k = 10;	N x ((N – 1) x N/2)	
while (k < 20) {	N x ((N – 1) x N/2) x (11)	
k++;	N x ((N – 1) x N/2) x (10)	
count++;	N x ((N – 1) x N/2) x (10)	
}		
j++;	N x ((N – 1) x N/2)	
}		

2. Task 1.1P asked you to develop / provided you with a number of the Vector class's methods (and properties), such as Count, Capacity, Add, IndexOf, Insert, Clear, Contains, Remove, and RemoveAt. What is the algorithmic complexity of each of these operations?

Count `public int Count { get; private set; } = 0;`	**Worst Case Complexity = O (1)** **Best Case Complexity = Ω (1)** **Average Case Complexity = Θ (1)** **Microsoft Complexity = O (1)** **Hence, it is the same.**

Capacity

```
public int Capacity { get; private set; } = 0;
```

Worst Case Complexity = O (1)
Best Case Complexity = Ω (1)
Average Case Complexity = Θ (1)

Microsoft Complexity = O (1)
Hence, it is the same.

ExtendedData

```
private void ExtendData(int extraCapacity)
{
   T[] newData = new T[data.Length + extraCapacity];
   for (int i = 0; i < Count; i++)
       {
           newData[i] = data[i];
       }
       data = newData;
}
```

Worst Case Complexity = O (N)
Best Case Complexity = Ω (N)
Average Case Complexity = Θ (N)

Microsoft Complexity = Not given

*** Not needed. Done for context.**

Add

```
public void Add(T element)
{
    if (Count == data.Length){
        ExtendData(DEFAULT_CAPACITY);
    }
    data[Count++] = element;
}
```

Worst Case Complexity = O (N)
Best Case Complexity = Ω (1)
Average Case Complexity = N/A

Microsoft Complexity = O (1) [for Count less than Capacity] or O (N) [if Capacity is increased]
Hence, it is the same.

IndexOf

```
public int IndexOf(T element)
{
     for (var i=0; i<Count; i++)
     {
         if (data[i].Equals(element))
         return i;
     }
     return -1;
}
```

Worst Case Complexity = O (N)
Best Case Complexity = Ω (1)
Average Case Complexity = N/A

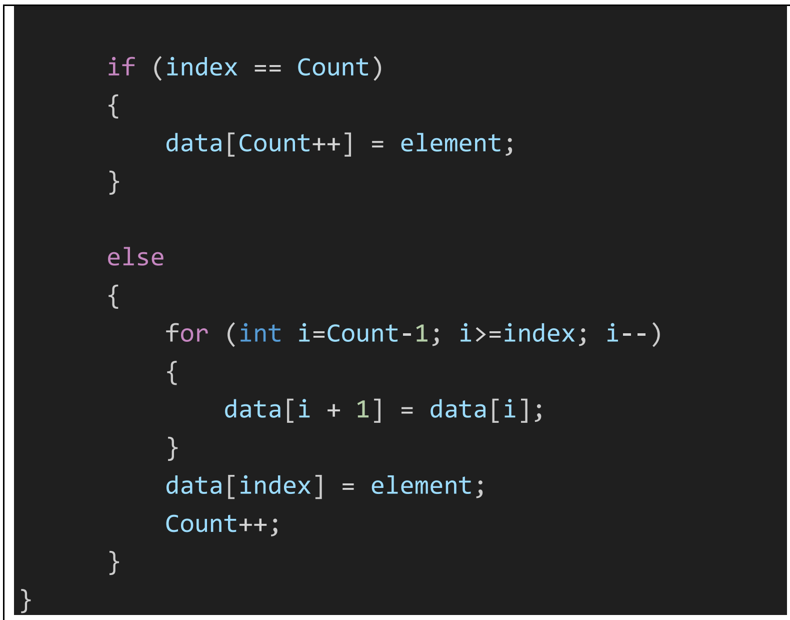
Microsoft Complexity = O (N)
Hence, it is the same.

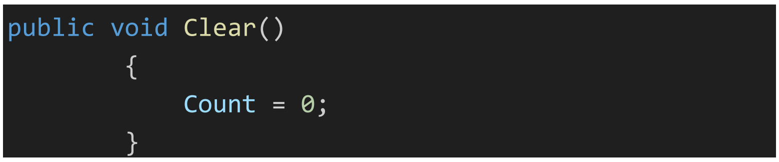
Insert

```
public void Insert(int index, T element)
{
      if (index < 0 || index > Count)
      {
          throw new IndexOutOfRangeException("The
index given is out of the range.");
      }

      if (Count == Capacity)
      {
          ExtendData(DEFAULT_CAPACITY);
      }
```

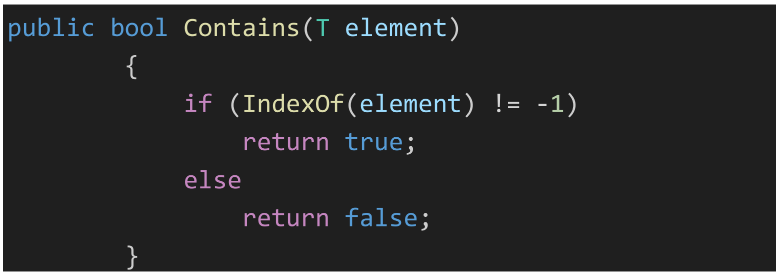
Worst Case Complexity = O (N)
Best Case Complexity = Ω (1)
Average Case Complexity = N/A

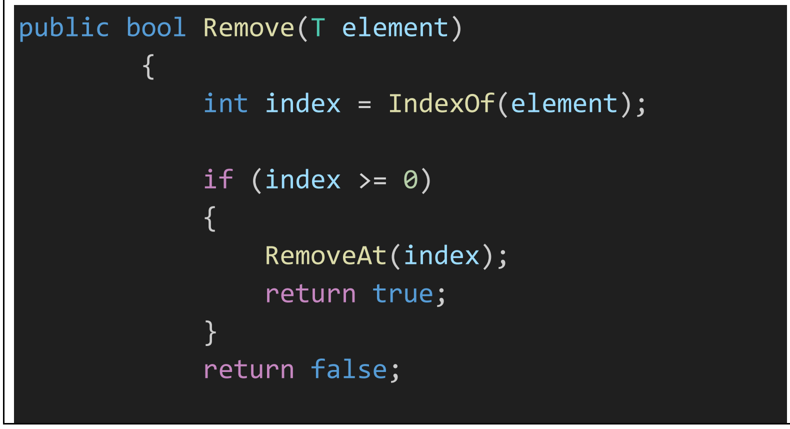
Microsoft Complexity = O (N)
Hence, it is the same.

<table>
<tr>
<td>

<pre>
      if (index == Count)
      {
          data[Count++] = element;
      }

      else
      {
          for (int i=Count-1; i>=index; i--)
          {
              data[i + 1] = data[i];
          }
          data[index] = element;
          Count++;
      }
}
</pre>
</td>
<td></td>
</tr>
<tr>
<td>
Clear

<pre>
public void Clear()
        {
            Count = 0;
        }
</pre>
</td>
<td>
Worst Case Complexity = O (1)
Best Case Complexity = Ω (1)
Average Case Complexity = Θ (1)

Microsoft Complexity = O (N)
It is not the same. They must have used a different logic.

*** They first removed all the elements from the array before making the Count 0.**
</td>
</tr>
<tr>
<td>
Contains

<pre>
public bool Contains(T element)
        {
            if (IndexOf(element) != -1)
                return true;
            else
                return false;
        }
</pre>
</td>
<td>
Worst Case Complexity = O (N)
Best Case Complexity = Ω (1)
Average Case Complexity = N/A

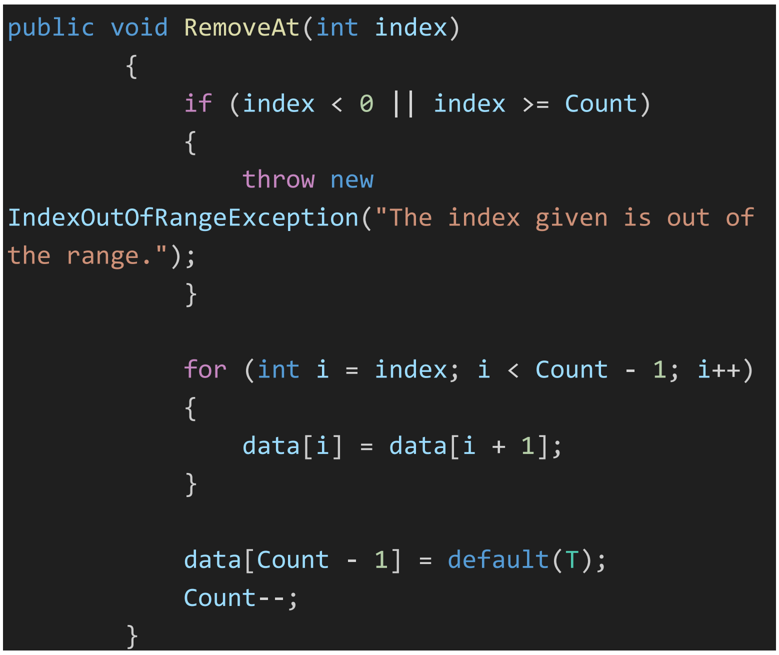
Microsoft Complexity = O (N)
Hence, it is the same.
</td>
</tr>
<tr>
<td>
Remove

<pre>
public bool Remove(T element)
        {
            int index = IndexOf(element);

            if (index >= 0)
            {
                RemoveAt(index);
                return true;
            }
            return false;
</pre>
</td>
<td>
Worst Case Complexity = O (N)
Best Case Complexity = Ω (1)
Average Case Complexity = N/A

Microsoft Complexity = O (N)
Hence, it is the same.
</td>
</tr>
</table>

```
        }
```

RemoveAt

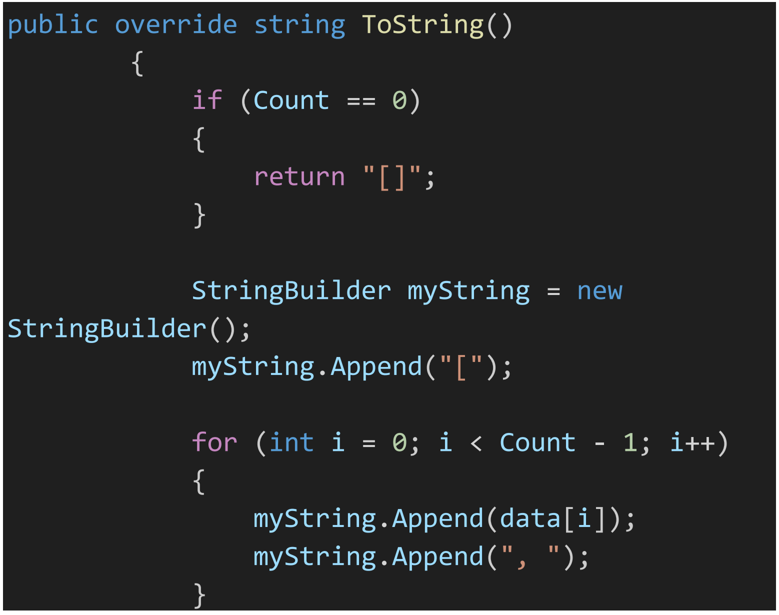


```
public void RemoveAt(int index)
        {
            if (index < 0 || index >= Count)
            {
                throw new 
IndexOutOfRangeException("The index given is out of 
the range.");
            }

            for (int i = index; i < Count - 1; i++)
            {
                data[i] = data[i + 1];
            }

            data[Count - 1] = default(T);
            Count--;
        }
```

Worst Case Complexity = O (N)
Best Case Complexity = Ω (1)
Average Case Complexity = N/A

Microsoft Complexity = O (N)
Hence, it is the same.

ToString



```
public override string ToString()
        {
            if (Count == 0)
            {
                return "[]";
            }

            StringBuilder myString = new 
StringBuilder();
            myString.Append("[");

            for (int i = 0; i < Count - 1; i++)
            {
                myString.Append(data[i]);
                myString.Append(", ");
            }
```

Worst Case Complexity = O (N)
Best Case Complexity = Ω (N)
Average Case Complexity = Θ (N)

Microsoft Complexity = Not given

*** Not needed. Done for context.**

3. f is a function that satisfies the following:
- f is in O (n^2),
- f is in Ω (n),
- f is neither in Θ (n) nor in Θ (n^2), but it can be represented with Θ.

Can you give an example of such a function f? Show that the function you name indeed satisfies all of the above. Also, name a well-known algorithm that meets these conditions for all situations (best, worst and average cases).

The function f (n) = n log n is the example that satisfies all the above conditions.

We know that f (n) = n log n grows slower than quadratic function so it is in O (n^2). Not only that, we also know that f (n) = n log n grows faster than linear function so it is in Ω (n).

Furthermore, we can clearly see that it lies in between Ω (n) (linear) and O (n^2) (quadratic) hence it is neither in Θ (n) nor in Θ (n^2). Finally, f (n) = n log n can be represented in Θ as it is bounded in between linear and quadratic function.

A well-known algorithm that meets these conditions for all the situations (best, worst and average) is the Merge Sort Algorithm.

Its worst case complexity is O (n log n) while its best case complexity is Ω (n log n). Hence, its overall complexity can be represented in Θ (n log n).

4. For each pair of functions given below, point out the asymptotic relationships that apply: f = O (g), f = Θ (g), and f = Ω (g).

a) f (n) = $n^{1/2}$ and g (n) = log n
'$n^{1/2}$' grows faster than 'log n'. Therefore, f (n) is not in O (g (n)) but it is in Ω (g (n)). Also '$n^{1/2}$' and 'log n' have different growth rates. Therefore, f (n) is not in Θ (g (n)).

b) f (n) = 1500 and g (n) = 2
'1500' and '2' both are constants and f (n) is bounded above and below by the constant function 2, so f (n) is in O (g (n)) and it is in Ω (g (n)).
Also as they have the same growth rates. Therefore, f (n) is also in Θ (g (n)).

c) f (n) = 800.2^n and g (n) = 3^n
'800.2^n' is bounded by a constant multiple of '3^n' for large 'n'. At the same time, (800.2 < 3) which means f (n) grows slower than g (n). Therefore, f (n) is in O (g (n)) but it is not in Ω (g (n)).
Also as they have different growth rates. Therefore, f (n) is not in Θ (g (n)).

d) f (n) = 4^{n+13} and g (n) = 2^{2n+2}
Here '4^{n+13} / 2^{2n+26}' we will get '2^{2n}. 2^{26} / 2^{2n}. 2^2' equal to 2^{24}, as 'n' approaches infinity.
We can clearly see that f (n) / g(n) approaches 2^{24} as 'n' becomes large which implies that f (n) is 2^{24} times larger than g (n) and they both have the same growth rate bounded by a constant. So, f (n) is in O (g (n)) and in Ω (g (n)).
Also as they have same growth rates. Therefore, f (n) is in Θ (g (n)).

e) f (n) = 9n.log n and g (n) = n.log 9n
Here '9n.log n / n.log 9n' we will get '9.log n/log 9n'. As 'n' becomes large log (9n) grows at the same rate as log n so we can write '9.log n/log 9n' ≈ '9.log n/log n' = 9. We can clearly see here that f (n) for a sufficiently large 'n' where n > $n_{0,}$ this satisfies both the upper bound and the lower bound condition.

Hence, we can say that f (n) and g (n) have the same logarithmic growth rate and f (n) is in O (g (n)) and in Ω (g (n)).
Also as they have same growth rates. Therefore, f (n) is in Θ (g (n)).

f) f (n) = n! and g (n) = (n + 1)!
Here 'n! / (n + 1)!' we will get '1 /n + 1' where 'n' increases as '1 /n + 1' approaches 0.
We can clearly see here that f (n) has slower growth rate than that of g (n). Hence, we can say that f (n) is in O (g (n)) and not in Ω (g (n)).
Also as they have different growth rates. Therefore, f (n) is not in Θ (g (n)).