Document analysis

Reverse sorting

Nhat Nam Ha

1. Write the reverse function.

```
19 references
public class ReverseSorting
{
    private IList<int> discs;
    private int length;

    8 references | 0/6 passing
    public ReverseSorting(int[] discs)
    {
        this.discs = new List<int>(discs);
        this.length = this.discs.Count;
    }

    6 references | 0/3 passing
    public ReverseSorting reverse(int index)
    {
        if (index < 1 || index > length)
            throw new IndexOutOfRangeException();

        int left = 0, right = index - 1;
        while (left < right)
        {
            int temp = discs[right];
            discs[right] = discs[left];
            discs[left] = temp;
            left++;
            right--;
        }
        return this;
    }
```

Check the condition if index input is in [1...length of discs], then reversing stack of discs by 2 pointers indicate the top and bottom of stack, swapping elements and narrow the range of swapping.

2. Design and implement your reverse sorting algorithm.

```
2 references | 0/1 passing
public ReverseSorting sort()
{
    for (int i = length - 1; i >= 0; i--)
    {
        int currentMax = discs[i];
        int currentMaxIndex = i;

        for (int j = 0; j < i; j++)
        {
            int maxValue = Math.Max(currentMax, discs[j]);
            if (maxValue > currentMax)
            {
                currentMax = maxValue;
                currentMaxIndex = j;
            }
        }

        if (currentMaxIndex == i) continue;
        if (currentMaxIndex != 0)
            reverse(currentMaxIndex + 1);

        reverse(i + 1);
    }
    return this;
}

3 references | 0/3 passing
public int[] getDiscs()
{
    return discs.ToArray();
}
```

For every iteration in outer loop, finding the max elements, if the max element is not in the correct position, it will be sorted by checking that element not in the top, then make a reverse put in the top, run another reverse based on outer loop index to put correct position.

3. Guarding correctness with at least 5-unit tests

```
0 references
public class ReverseSortingTests
{
    [Test]
    0 references
    public void Initialize()
    {
        int[] discs = new int[] { 4, 5, 7, 8 };
        ReverseSorting sorting = new(discs);

        Assert.IsNotNull(discs);
    }

    [Test]
    0 references
    public void CheckPrint()
    {
        int[] discs = new int[] { 4, 5, 7, 8 };

        ReverseSorting sorting = new(discs);
        int[] newDiscs = sorting.getDiscs();

        Assert.AreEqual(discs, newDiscs);
    }
```

```
[Test]
0 references
public void CheckReverse()
{
    int[] discs = new int[] { 4, 5, 7, 8 };
    int[] expected = new int[] { 7, 5, 4, 8 };

    ReverseSorting sorting = new(discs);
    var afterReverse = sorting.reverse(3);
    var newDiscs = afterReverse.getDiscs();

    Assert.AreEqual(expected, newDiscs);
}

[Test]
0 references
public void ReverseGetOutOfRangeException()
{
    int[] discs = new int[] { 4, 5, 7, 8 };
    ReverseSorting sorting = new(discs);
    Assert.Catch<IndexOutOfRangeException>(() => sorting.reverse(0));
}

[Test]
0 references
public void ReverseGetOutOfRangeException2()
{
    int[] discs = new int[] { 4, 5, 7, 8 };
    ReverseSorting sorting = new(discs);
    Assert.Catch<IndexOutOfRangeException>(() => sorting.reverse(5));
}
```

```
[Test]
0 references
public void CheckSort()
{
    int[] discs = new int[] { 8, 7, 5, 4 };
    int[] expected = new int[] { 4, 5, 7, 8 };

    ReverseSorting sorting = new(discs);
    var afterSort = sorting.sort();
    var newDiscs = afterSort.getDiscs();

    Assert.AreEqual(expected, newDiscs);
}
```

4. Performance Tests

```
[Params(1000, 2500, 5000, 10_000, 20_000, 30_000)]
public int N;

private int[] reverses;
private int[] sorts;

[GlobalSetup]
0 references
public void Setup()
{
    reverses = new int[N];
    sorts = new int[N];

    for (int i = 0; i < N; i++)
    {
        reverses[i] = N - i;
        sorts[i] = N - i;
    }
}
```

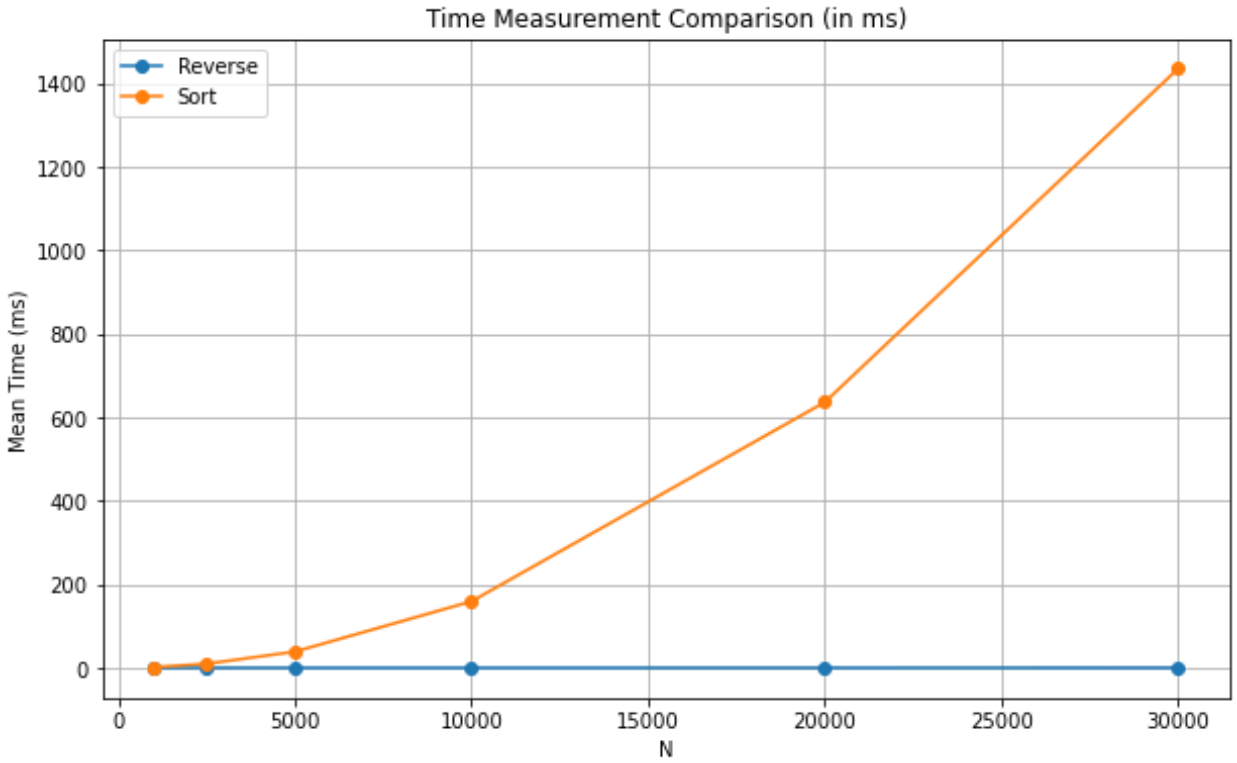
```
[Benchmark]
0 references
public void Reverse()
{
    ReverseSorting sort = new(reverses);
    sort.reverse(N);
}

[Benchmark]
0 references
public void Sort()
{
    ReverseSorting sorting = new(sorts);
    sorting.sort();
}
```

Run the benchmark code and generating report for time measurement of 2 tasks reverse and sorting.

Method	N	Mean	Error	StdDev	Gen0	Gen1	Gen2	Allocated
Reverse	1000	3.893 µs	0.0508 µs	0.0475 µs	0.8621	0.0076	-	3.99 KB
Sort	1000	1,572.903 µs	5.8501 µs	5.4722 µs	-	-	-	3.99 KB
Reverse	2500	9.421 µs	0.0452 µs	0.0401 µs	2.1362	0.0763	-	9.85 KB
Sort	2500	9,708.229 µs	35.7158 µs	33.4085 µs	-	-	-	9.85 KB
Reverse	5000	18.859 µs	0.1185 µs	0.1108 µs	4.2419	0.2747	-	19.62 KB
Sort	5000	38,738.661 µs	162.8398 µs	152.3205 µs	-	-	-	19.63 KB
Reverse	10000	37.734 µs	0.2365 µs	0.2212 µs	8.4229	1.0376	-	39.15 KB
Sort	10000	159,404.078 µs	712.4417 µs	666.4184 µs	-	-	-	39.98 KB
Reverse	20000	75.638 µs	0.5303 µs	0.4961 µs	16.8457	2.8076	-	78.21 KB
Sort	20000	636,513.487 µs	2,712.7964 µs	2,537.5515 µs	-	-	-	87.08 KB
Reverse	30000	169.166 µs	1.6894 µs	1.4976 µs	36.8652	36.8652	36.8652	117.27 KB
Sort	30000	1,435,400.493 µs	5,423.8120 µs	5,073.4371 µs	-	-	-	126.14 KB

Plotting report to graph for visualizing.



5. Analysis Complexity

In my implementation, my reverse sorting implementation consists of 2 nested for loop:

- The outer for-loop iterates backward elements in the discs which complexity is O(N).
- The body inside are inner-loop finds the max element within the range based on the outer-loop, also in O(N) and at most 2 reverse operations.
    - 1st: iteration in outer-loop, inner loop iterates N-1 elements.
    - 2nd iteration in outer-loop, inner loop iterates N-2 elements.
    - 3rd iteration in outer-loop, inner loop iterates N-3 elements.
    - N-1 iteration in outer-loop, inner loop iterates 1 element.

Total operations for the finding max elements based on outer-loop index, will be 1 + 2 + 3+…+N-1 = (N-1) *N / 2.

Reverse is called at most 2 times inside outer loop, with assume time complexity is O (1) then it will run at most 2*N operations.

So total operation will be (N-1) *N / 2 + 2*N = (N^2 + 3*N) / 2, time complexity is **O(N^2).**

However, the reverse function iterates whole list by using 2 pointers, which is at most O(N), then total operations will be same as finding the max elements above which is (N-1) *N / 2.

So total operation will be (N-1) *N / 2 + 2*(N-1) *N / 2 = 3*(N-1) N/2 = 3*(N^2 – N) /2, time complexity is **O(N^2).**