Gabriel Brehm
CS361
May 5, 2019

Lab 2 Report

Part 1

```
public static void RadixSort(int[] list)
{
	for (int i = 1; list.Max() / i > 0; i *= 10)
	{
		CountingSort(list, i);
	}
}
```

This first function is Radix sort which will sort a given list of integers. It does so by walking though a for loop until the maximum value found in the list divided by the current i value is not greater than 0. Each iteration, the i value is multiplied by 10, and the counting sort function is called on the list, with i as the second parameter.

```
public static void CountingSort(int[] listA, int digit)
{
	int[] listB = new int[listA.Length];
	int[] listC = new int[10];
	for (int i = 0; i < 10; i++)
	{
		listC[i] = 0;
	}
	for (int i = 0; i < listA.Length; i++)
	{
		listC[(listA[i] / digit) % 10]++;
	}
	for (int i = 1; i < 10; i++)
	{
		listC[i] += listC[i - 1];
	}
	for (int i = listA.Length - 1; i >= 0; i--)
```

```
        {
            listB[listC[(listA[i] / digit) % 10] - 1] = listA[i];
            listC[(listA[i] / digit) % 10]--;
        }
        for (int i = 0; i < listA.Length; i++)
        {
            listA[i] = listB[i];
        }
    }
```

This method is Counting sort, which works by counting the occurrences of each value in the given array of integers called listA. The second parameter is the digit we are concerned with, as Radix sort sorts by column. First, we create two new arrays on the heap, one that is the same length as the given array, and another that is of length 10. The first of those is called listB while the second is called listC. The first for loop initializes every element of listC to 0. The next loop counts the number of occurrences of each number in listA, based on the digit we are currently looking at. Then each value of listC is set to be the sum of all values smaller than it. Next the method sets each element of listA to the appropriate location in listB based on some fun mathematics. Finally, the contents of listB are copied back into listA.

Part 2

```
    public static void BucketSort(int[] listA)
    {
        int min = listA.Min();
        List<int>[] listB = new List<int>[listA.Max() - min + 1];
        int n = listA.Length;
        for(int i = 0; i < listB.Length; i++)
        {
            listB[i] = new List<int>();
        }
        for(int i = 0; i < n; i++)
        {
            listB[listA[i] - min].Add(listA[i]);
        }
        for(int i = 0; i < listB.Length; i++)
```

```
        {
            listB[i].Sort();
        }
        int k = 0;
        for (int i = 0; i < listB.Length; i++)
        {
            for(int j = 0; j < listB[i].Count(); j++)
            {
                listA[k] = listB[i][j];
                k++;
            }
        }
    }
```

This next method is Bucket sort, also known as Bin sort. It functions by sorting the given array into an array of linked lists. First it creates an integer and sets it to the minimum value in the given list, then it creates an array of linked lists with a length equal to the maximum value minus the minimum value plus 1. Then it initializes each item in the array to a new empty linked list. Then each element of the given array is added to the new array at an index based on the value in the given list. Once each bucket is full of numbers, they are individually sorted. Finally, all of the list are concatenated back together in order and put back into the given list.

Part 3

Please refer to the Tables and Graphs, and Screen Dump sections towards the end of the report.

Part 4

Please refer to the Tables and Graphs sections towards the end of the report. Note that there is data regarding the execution time of Merge Sort. This data was collected during the previous lab, and thus, based on that lab's completeness, is verified to be correctly sorted.

Part 5

Please refer to the Tables and Graphs section towards the end of the report.

Part 6

```
    public static void RemoveMax(int[] list)
    {
        int max = list.Max();
```

```
        for (int i = 0; i < list.Length; i++)
        {
            if (list[i] == max)
            {
                list[i] = 0;
                break;
            }
        }
    }
```

This method removes the maximum value in the given array. Since the array can have multiple instances of that value, it simply removes the first one it sees. That would mean that this method uses a greedy approach. First it sets an integer to the maximum value in the array, then it walks down the array until it finds a matching value. It then sets that value to 0, and breaks out of the loop. Since it sets the value to 0, this method assumes the array contains only nonnegative integers.

```
    public static void FindTenLargest(int[] listA, int[] listB, int index)
    {
        if (index < 10)
        {
            listB[index] = listA.Max();
            RemoveMax(listA);
            FindTenLargest(listA, listB, index + 1);
        }
    }
```

This method finds the 10 largest elements in a given array of integers. The parameter listA is the array we are looking for the value in, listB is the array we are storing the largest elements in, and index is the index. This is a recursive function, so our exit condition is that index is index is greater than or equal to 10. If it is not, then the max value in the array is added to listB, the RemoveMax function is called on listA, and then FindTenLargest is recursively called on the same 2 arrays, and the index plus 1.

Part 7

To see the ten largest values of each array being outputted, please refer toe the Screen Dumps section towards the end of the report. The average times can also be seen in the Tables and Graphs section.

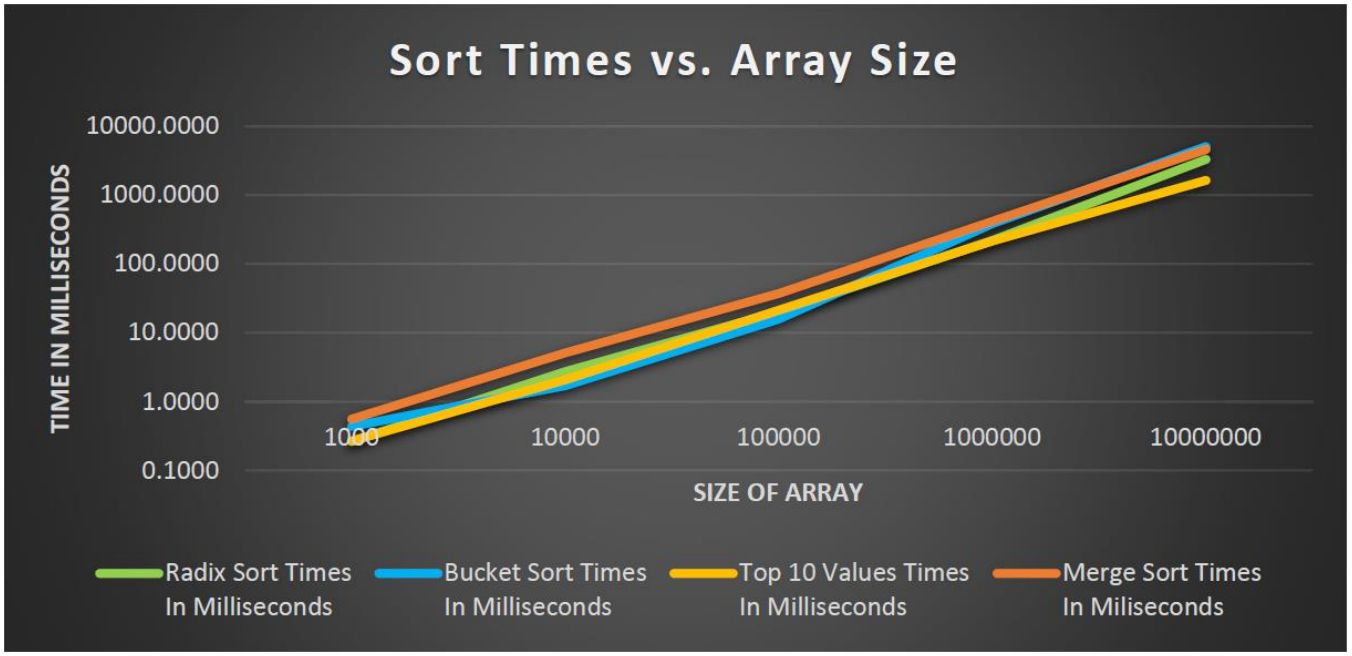
Part 8

Please refer to the Tables and Graphs, and Screen Dumps section of the report.

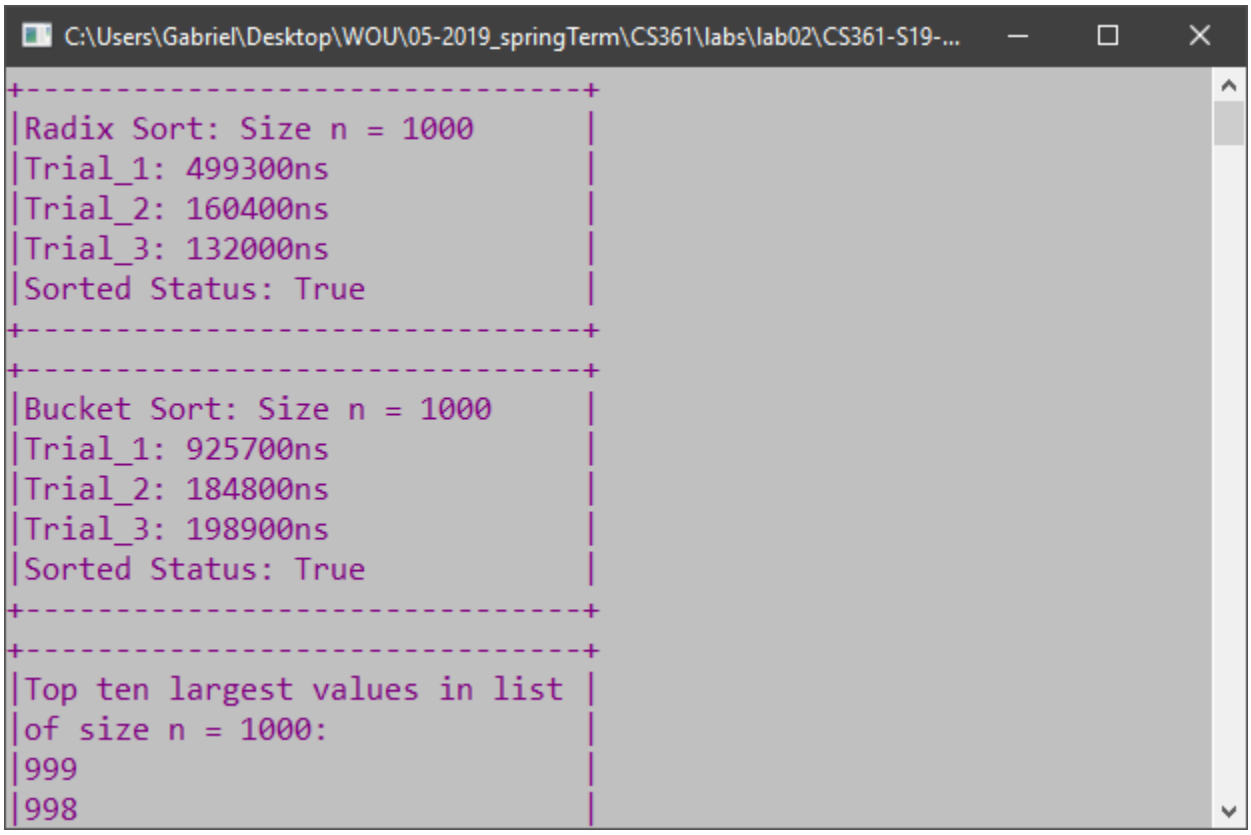
Tables and Graphs

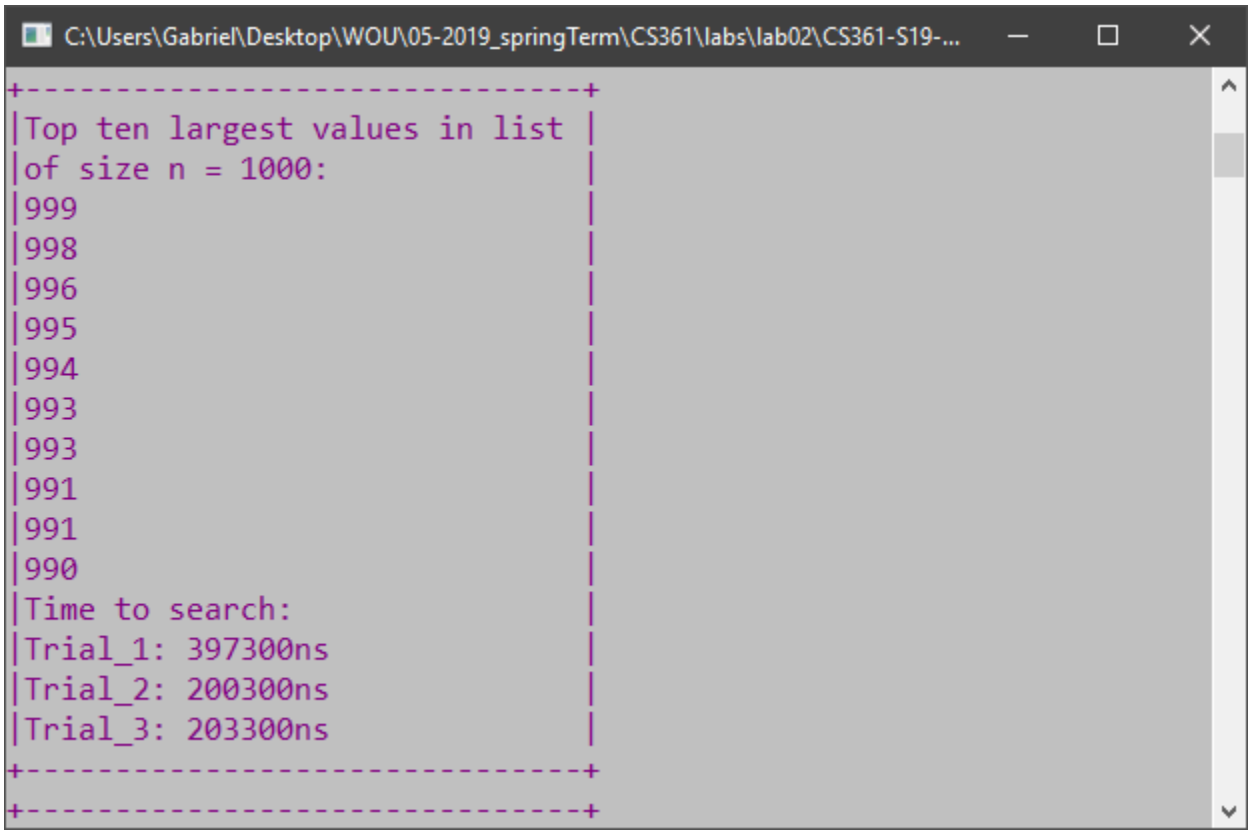
Size	Radix Sort Times In Nanoseconds	Bucket Sort Times In Nanoseconds	Top 10 Values Times In Nanoseconds
1000	499300	925700	397300
	160400	184800	200300
	132000	198900	203300
10000	1470700	1377600	2124800
	2109500	1985400	2072900
	4657200	1740600	2074000
100000	22600500	11962500	22618800
	18360200	19767400	20748200
	19849500	15648300	20857600
1000000	219257500	422238900	214299900
	219983900	402700400	213615400
	220999700	335226200	216933100
10000000	2826372900	4381116600	1324073800
	3741697600	5513999600	1771584400
	3356266200	5244999800	1797010700

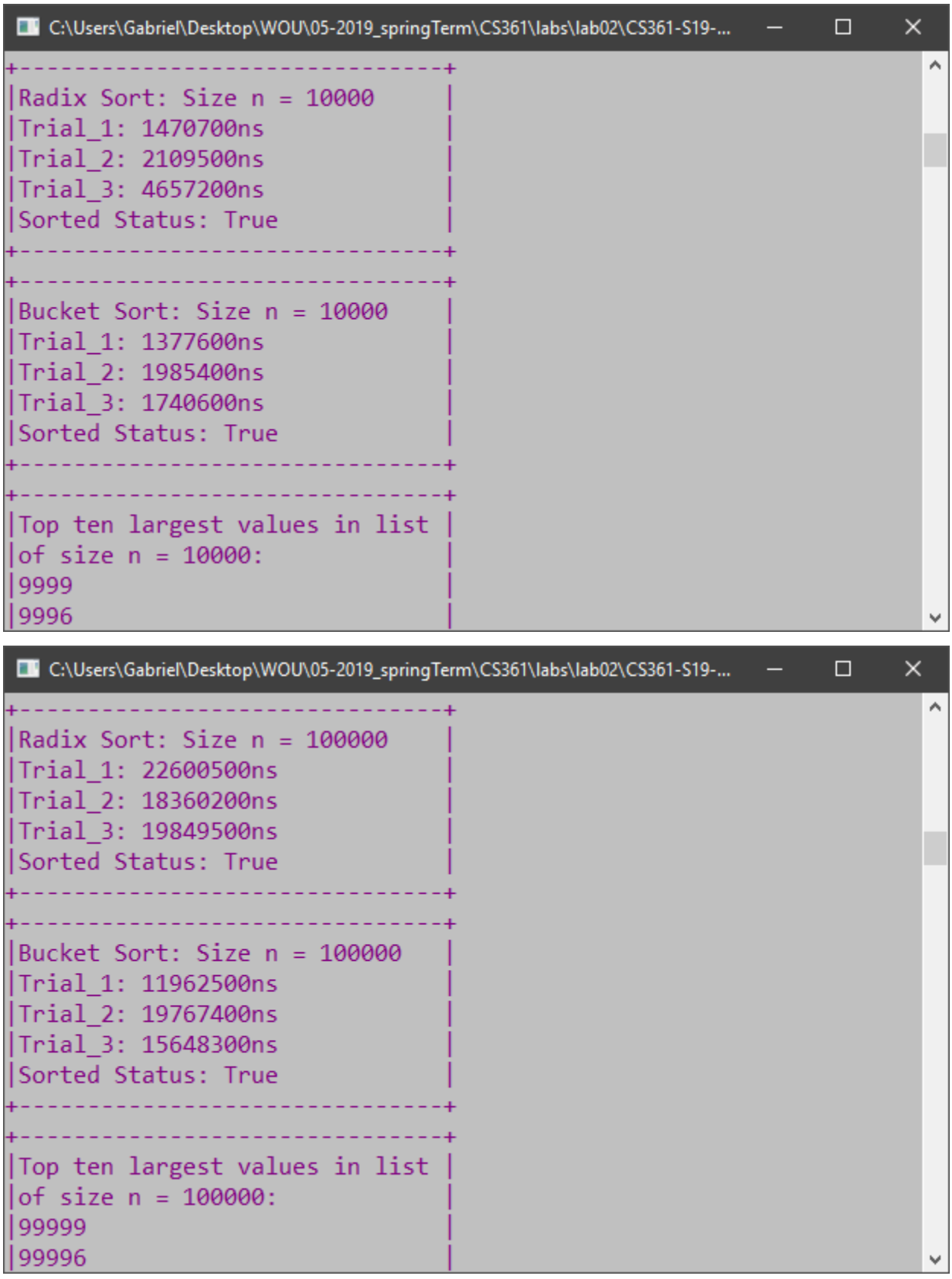
Size	Radix Sort Times In Milliseconds	Bucket Sort Times In Milliseconds	Top 10 Values Times In Milliseconds	Merge Sort Times In Miliseconds
1000	0.2639	0.4365	0.2670	0.5570
10000	2.7458	1.7012	2.0906	5.1134
100000	20.2701	15.7927	21.4082	37.0481
1000000	220.0804	386.7218	214.9495	415.9392
10000000	3308.1122	5046.7053	1630.8896	4607.0562

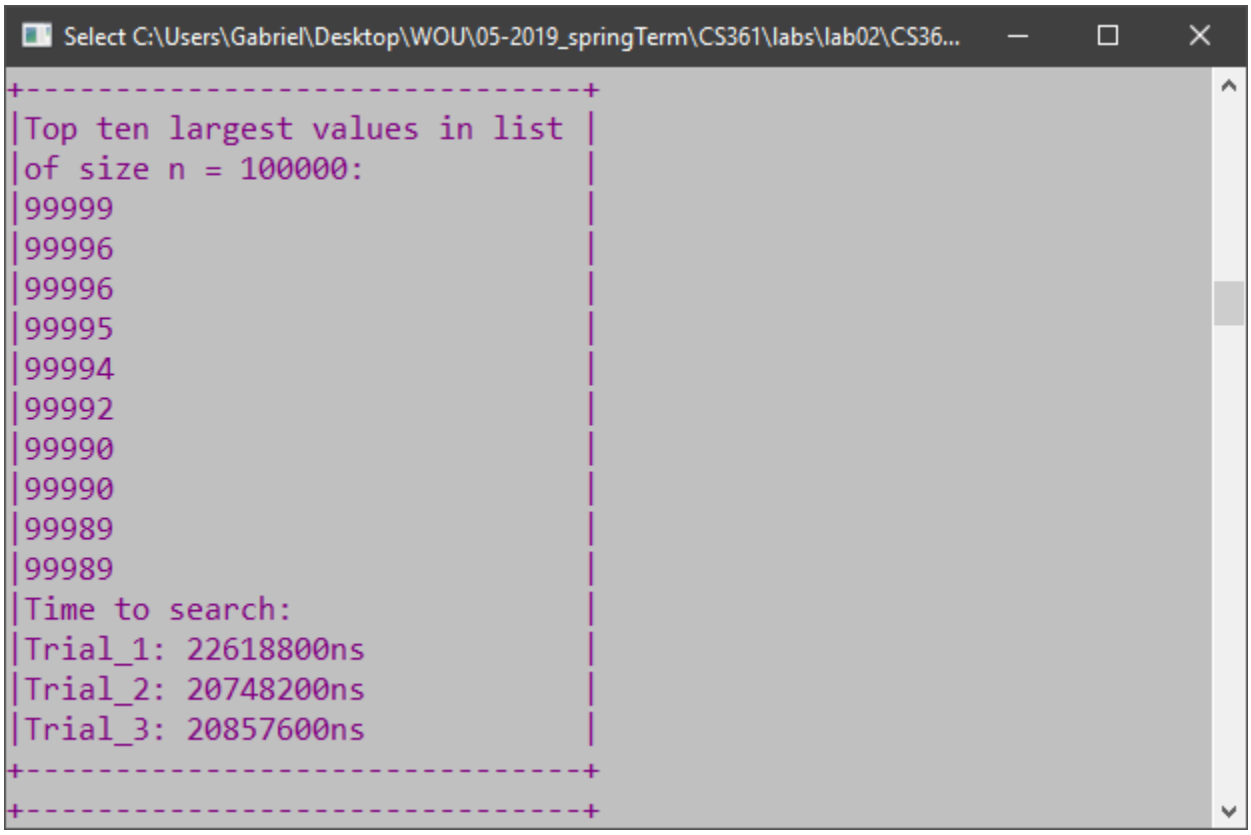


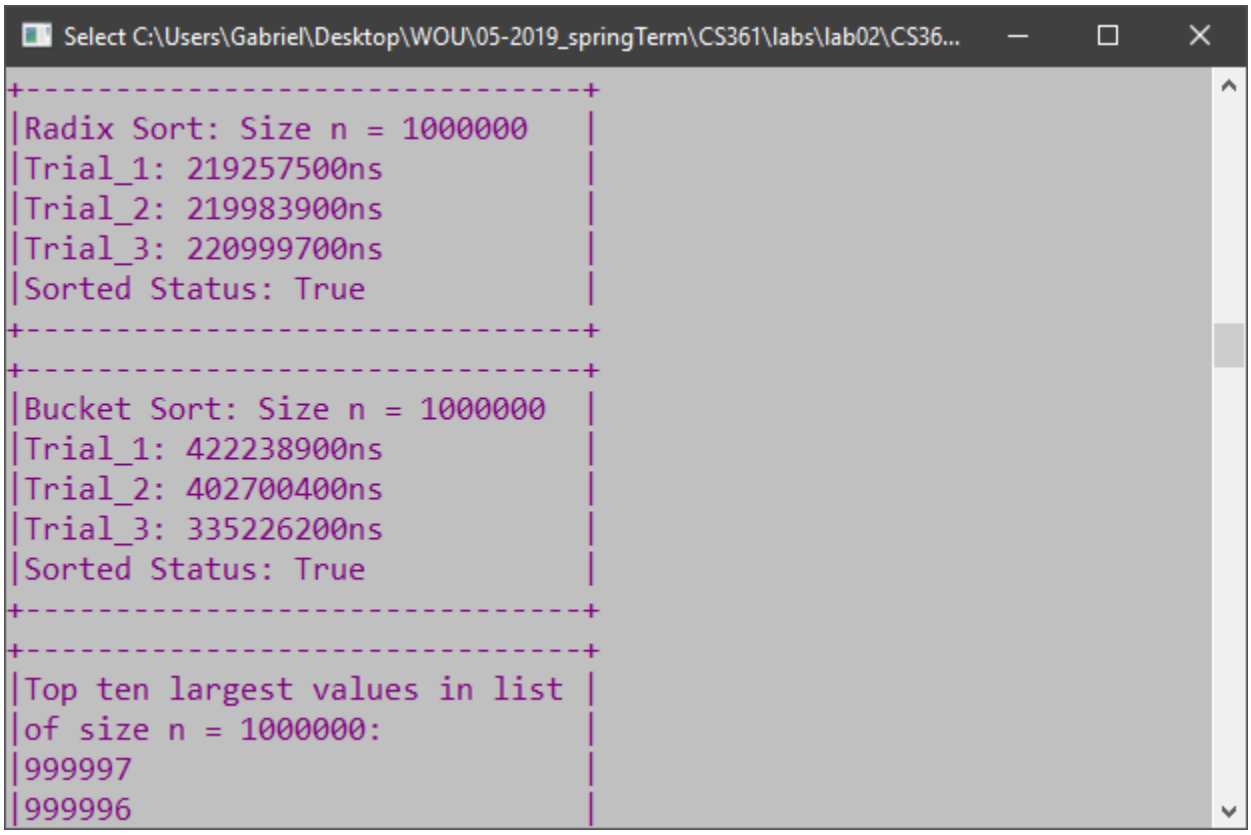
Screen Dumps

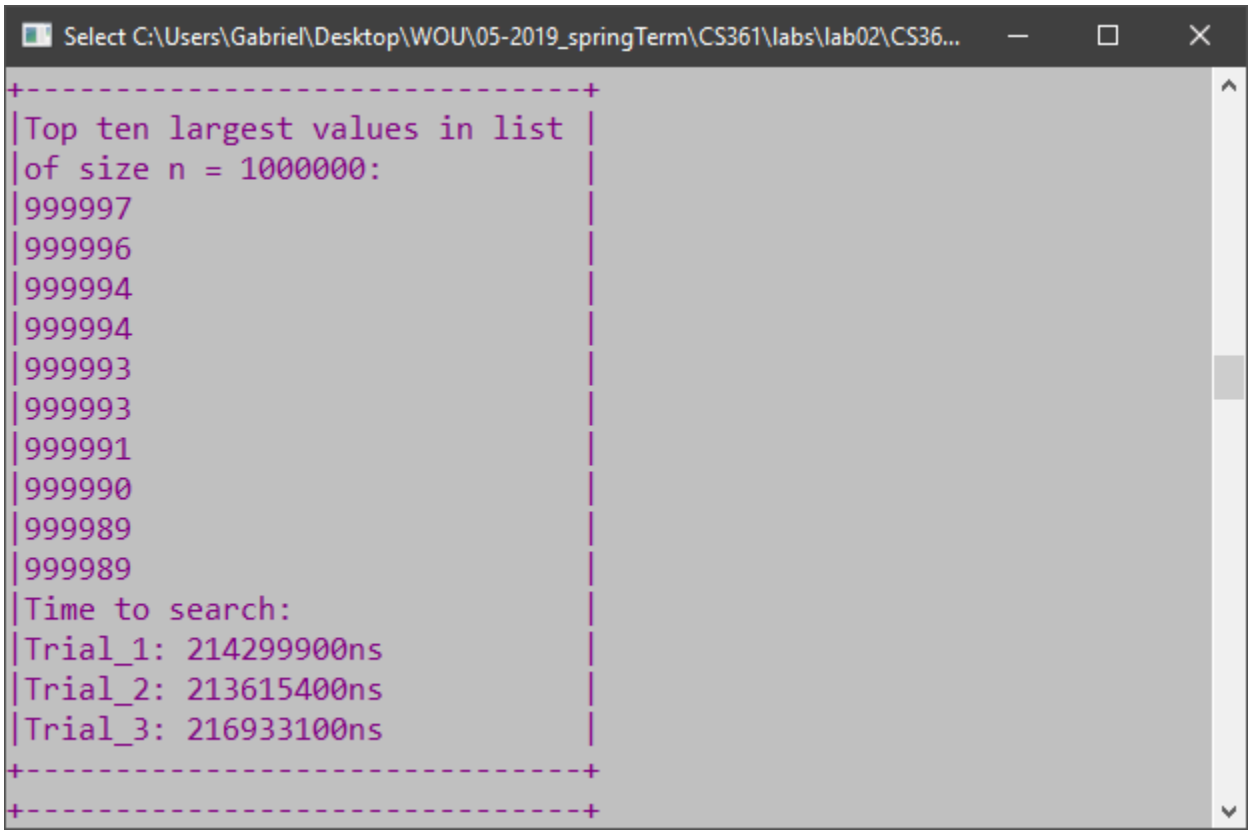


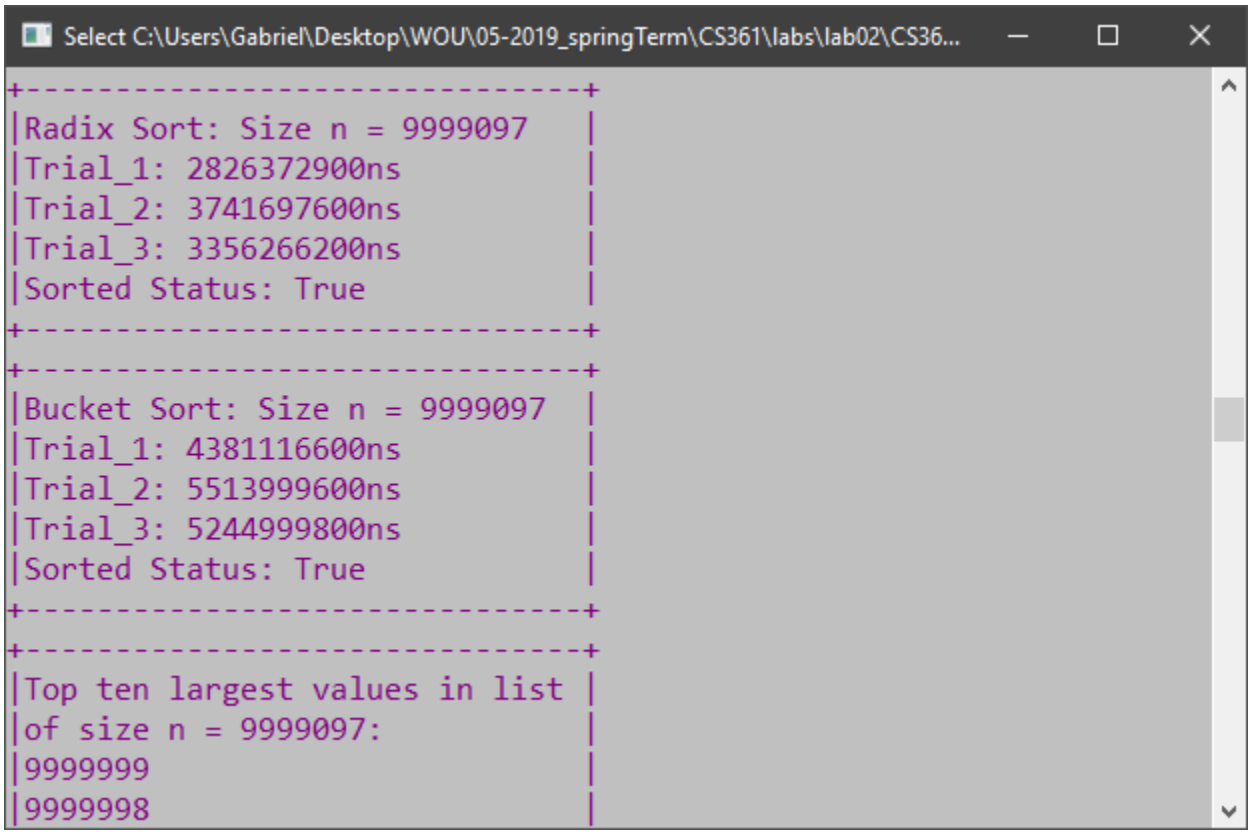


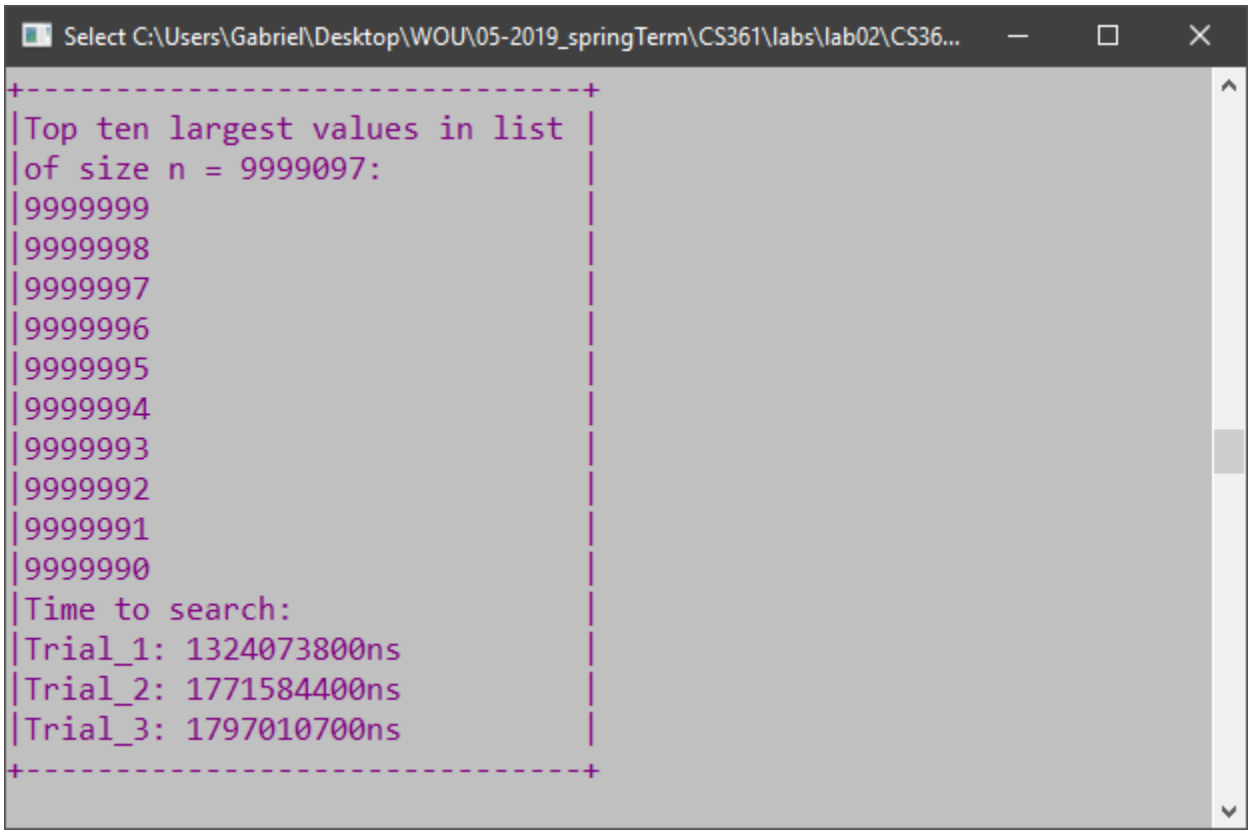

C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\labs\lab02\CS361-S19-...
+--------------------------------+
|Radix Sort: Size n = 10000 |
|Trial_1: 1470700ns |
|Trial_2: 2109500ns |
|Trial_3: 4657200ns |
|Sorted Status: True |
+--------------------------------+
+--------------------------------+
|Bucket Sort: Size n = 10000 |
|Trial_1: 1377600ns |
|Trial_2: 1985400ns |
|Trial_3: 1740600ns |
|Sorted Status: True |
+--------------------------------+
+--------------------------------+
|Top ten largest values in list |
|of size n = 10000: |
|9999 |
|9996 |
C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\labs\lab02\CS361-S19-...
+--------------------------------+
|Radix Sort: Size n = 100000 |
|Trial_1: 22600500ns |
|Trial_2: 18360200ns |
|Trial_3: 19849500ns |
|Sorted Status: True |
+--------------------------------+
+--------------------------------+
|Bucket Sort: Size n = 100000 |
|Trial_1: 11962500ns |
|Trial_2: 19767400ns |
|Trial_3: 15648300ns |
|Sorted Status: True |
+--------------------------------+
+--------------------------------+
|Top ten largest values in list |
|of size n = 100000: |
|99999 |
|99996 |


Select C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\labs\lab02\CS36...
+--------------------------------+
|Top ten largest values in list |
|of size n = 100000: |
|99999 |
|99996 |
|99996 |
|99995 |
|99994 |
|99992 |
|99990 |
|99990 |
|99989 |
|99989 |
|Time to search: |
|Trial_1: 22618800ns |
|Trial_2: 20748200ns |
|Trial_3: 20857600ns |
+--------------------------------+
+--------------------------------+


Select C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\labs\lab02\CS36...
+--------------------------------+
|Radix Sort: Size n = 1000000 |
|Trial_1: 219257500ns |
|Trial_2: 219983900ns |
|Trial_3: 220999700ns |
|Sorted Status: True |
+--------------------------------+
+--------------------------------+
|Bucket Sort: Size n = 1000000 |
|Trial_1: 422238900ns |
|Trial_2: 402700400ns |
|Trial_3: 335226200ns |
|Sorted Status: True |
+--------------------------------+
+--------------------------------+
|Top ten largest values in list |
|of size n = 1000000: |
|999997 |
|999996 |


Select C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\labs\lab02\CS36...
+--------------------------------+
|Top ten largest values in list |
|of size n = 1000000: |
|999997 |
|999996 |
|999994 |
|999994 |
|999993 |
|999993 |
|999991 |
|999990 |
|999989 |
|999989 |
|Time to search: |
|Trial_1: 214299900ns |
|Trial_2: 213615400ns |
|Trial_3: 216933100ns |
+--------------------------------+
+--------------------------------+


Select C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\labs\lab02\CS36...
+--------------------------------+
|Radix Sort: Size n = 9999097 |
|Trial_1: 2826372900ns |
|Trial_2: 3741697600ns |
|Trial_3: 3356266200ns |
|Sorted Status: True |
+--------------------------------+
+--------------------------------+
|Bucket Sort: Size n = 9999097 |
|Trial_1: 4381116600ns |
|Trial_2: 5513999600ns |
|Trial_3: 5244999800ns |
|Sorted Status: True |
+--------------------------------+
+--------------------------------+
|Top ten largest values in list |
|of size n = 9999097: |
|9999999 |
|9999998 |


Select C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\labs\lab02\CS36...
+--------------------------------+
|Top ten largest values in list |
|of size n = 9999097: |
|9999999 |
|9999998 |
|9999997 |
|9999996 |
|9999995 |
|9999994 |
|9999993 |
|9999992 |
|9999991 |
|9999990 |
|Time to search: |
|Trial_1: 1324073800ns |
|Trial_2: 1771584400ns |
|Trial_3: 1797010700ns |
+--------------------------------+

References

I worked with Jacob Malmstadt on this project.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). Cambridge, MA: The MIT Press.

Bucket Sort. (2015, September 27). Retrieved May 1, 2019, from https://www.programmingalgorithms.com/algorithm/bucket-sort.

Radix Sort. (2019, March 06). Retrieved May 2, 2019, from https://www.geeksforgeeks.org/radix-sort/.