3D5A Lab6&7

Abstract

The objective of this experiment was to implement some sorting algorithm, learn how to implement quicksort, learn how to evaluate the performance of a sorting algorithm and use sorting in a practical application.

Results:

1. In task 1, I implemented quicksort to sort an array of integers. My implementation counts the number of times values are swapped in the array and the number of times two values in the array are compared and reports the total number of swaps and comparisons at the end of the program. I began by demonstrating my implementation on array of 10 values for each of the 5 types of data given below and reported the total number of swaps and comparisons for type of data. I then demonstrated my implementation of quicksort of these 5 types of data given reported the total number of swaps and comparisons again, as shown in Figure 1 and 2.

    - Unique Random Values
    - Random Values
    - Ascending Sorted List
    - Descending Sorted List
    - Uniform List

    I observe that my implementation of quicksort performed the best for the two randomly shuffled arrays. It performed better for the array of random values with duplicates than it did for the array without duplicates because my implementation of the partition function didn't swap two elements in the array if those two elements had the same value. It didn't perform as well for the two array that were already sorted in ascending or descending because my implementation used the last element of the array as the pivot and the arrays were already sorted so that all the elements left of the pivot were lesser than or greater than the pivot. This meant when quicksort called itself recursively it was only calling an array that was one element smaller instead of half of the size of the array.

```
Running profile tests with 10 elements

TEST: Unique Random Values
SORTED: Y
SWAPS: 6
COMPS: 64

TEST: Random Values
SORTED: Y
SWAPS: 4
COMPS: 67

TEST: Ascending Sorted List
SORTED: Y
SWAPS: 0
COMPS: 90

TEST: Descending Sorted List
SORTED: Y
SWAPS: 5
COMPS: 85

TEST: Uniform List
SORTED: Y
SWAPS: 0
COMPS: 58
```

Figure 1.

```
Running profile tests with 10000 elements

TEST: Unique Random Values
SORTED: Y
SWAPS: 30127
COMPS: 238140

TEST: Random Values
SORTED: Y
SWAPS: 28945
COMPS: 234628

TEST: Ascending Sorted List
SORTED: Y
SWAPS: 0
COMPS: 50044995

TEST: Descending Sorted List
SORTED: Y
SWAPS: 5000
COMPS: 50039995

TEST: Uniform List
SORTED: Y
SWAPS: 0
COMPS: 247972
```

Figure 2.

2. In task 2, I implemented insertion sort and evaluated it in the same manner as I did quicksort as well as printed the swaps and comparisons for each of the 5 different types of data, as shown in Figure 3 and 4. I observe that my implementation of insertion sort performed the best for the ascending sorted array and the array where every value was the same because these two arrays were already in ascending order. This meant that my implementation of insertion sort only iterated through the array once. It did not perform as well for the other three arrays because it must swap an element with every element that has a greater value than it that is already sorted to sort the list. This means that insertion sort is bad for big files but it good for nearly sorted data whereas quicksort is good for big files and bad for nearly sorted data.

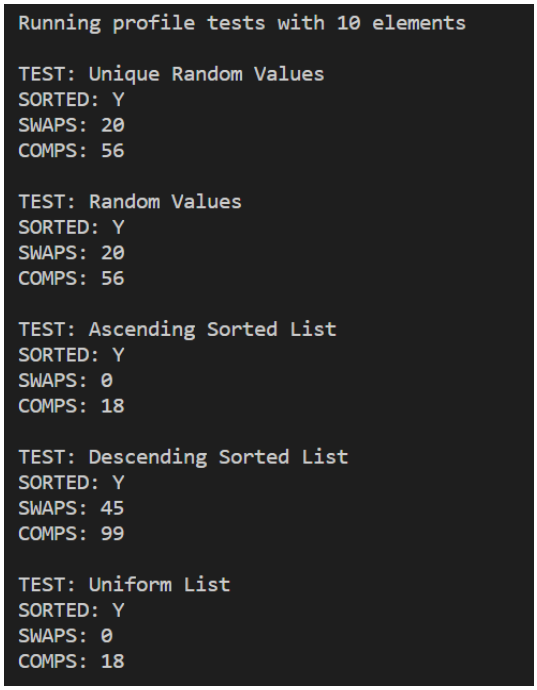

Figure 3

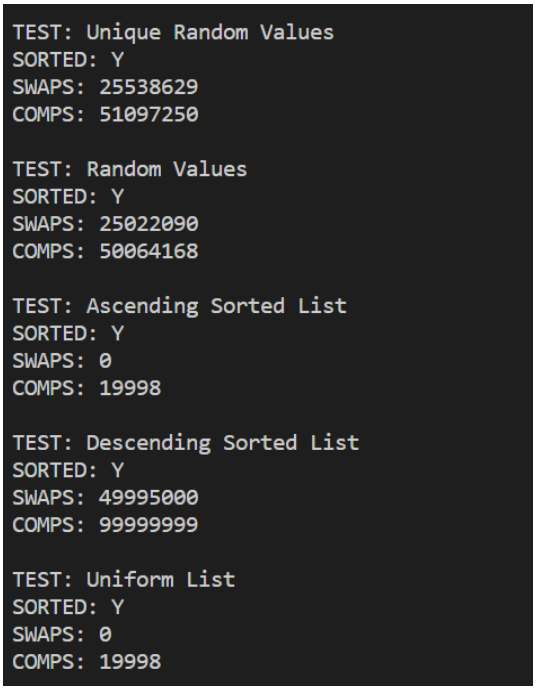

Figure 4.

3. In task 3, I had to sort the reviews based on games' scores and find out what the most popular games of the last 20 years were. I already stated in task 2 that quicksort is good for big files and bad for nearly sorted data whereas insertion sort is bad for big files but it good for nearly sorted data so, with in mind, I combined these two sorts. If the array was greater than 10 elements it would sort games using quicksort and if the was less than 10 elements it would sort it using insertion sort. I sorted the list by score and sub-sorted by title using this implementation of these two sorts to find the most popular games in the last 20 years. I printed the Top 10 and excluded duplicates, as shown in Figure 5. Finally, if I wanted to get the top 5 games for each of the last 20 years I would sub-sort by year instead and then by title if need be and print the top 5 games for each of the last 20 years.

Title	Platform	Score	Release Year
Zero Escape: Virtue's Last Reward	Nintendo 3DS	10	2012
Zen Bound	iPhone	10	2009
Xenogears	PlayStation	10	1998
World of Goo	iPhone	10	2011
Wave Race 64	Nintendo 64	10	1996
Wario Ware Twisted!	Game Boy Advance	10	2005
Virtua Fighter 4: Evolution	PlayStation 2	10	2003
Viewtiful Joe	GameCube	10	2003
Vagrant Story	PlayStation	10	2000
Unreal Tournament [1999]	PC	10	1999

Figure 5.

References

1. https://www.tutorialspoint.com/c_standard_library/c_function_rand.htm
2. https://www.youtube.com/watch?v=WaNLJf8xzC4
3. https://www.youtube.com/watch?v=JU767SDMDvA
4. https://megocode3.wordpress.com/2008/01/28/8/