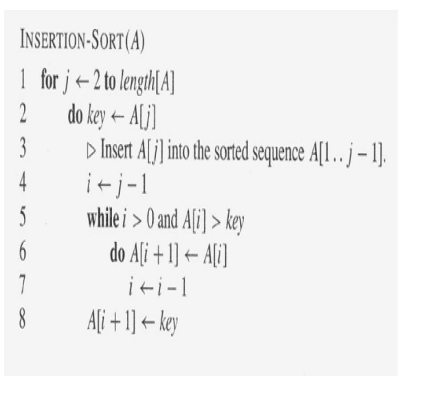
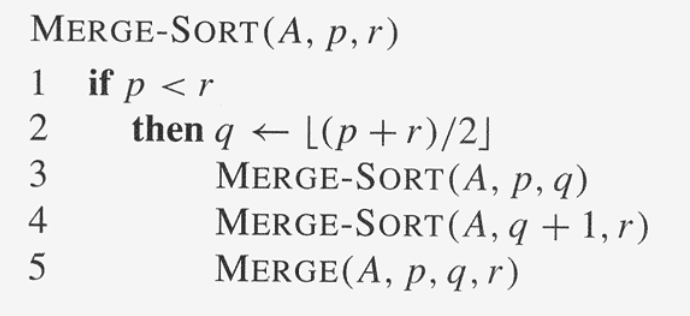
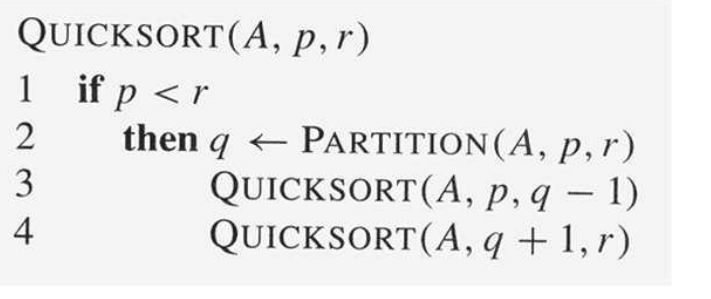
Juan Bermudez  
10-18-2018  
CMPE 3333  
Project 3

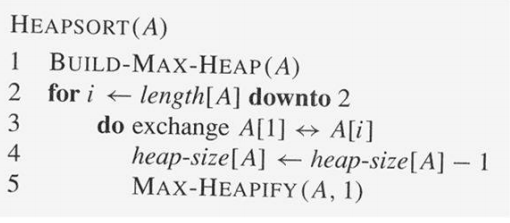
Report For Project 3

The program’s code may be executed on Visual Studio or on an online C++ compiler. All the sorting algorithms work accordingly in sorting integers. The program will sort 3 randomly generated arrays with values from 0 to 1000. The ReverseArray function will sort the arrays in reverse order, from largest to smallest. The comparisons made in each array by an algorithm are also displayed, along with the running times. A persistent issue that kept crashing the program was the size of the arrays. For example, if an array’s length was of 106, then the program would end. Another issue encountered with extraordinarily large values in an array was the output of negative numbers. For this reason the sizes of the arrays were kept at 103, 104, and 105. The output of the program is given in order for Insertion Sort, Merge Sort, Heap Sort, and Quick Sort. The algorithms were designed by following the pseudocode on the power of chapter 7

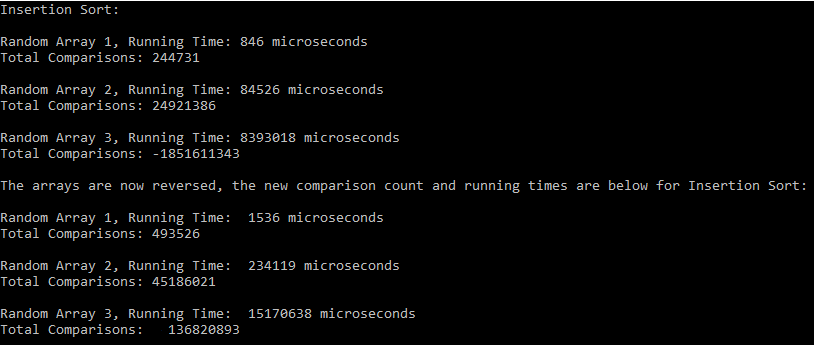


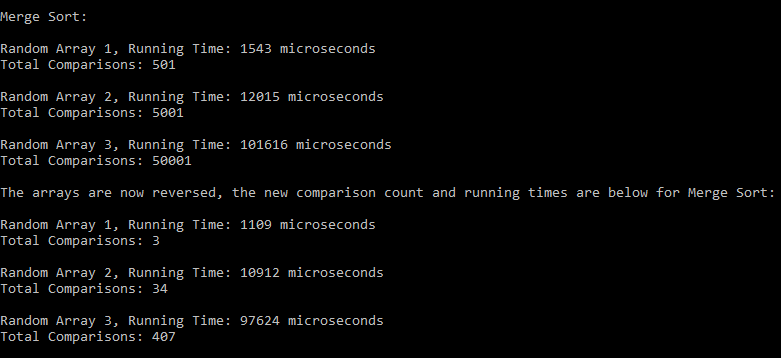


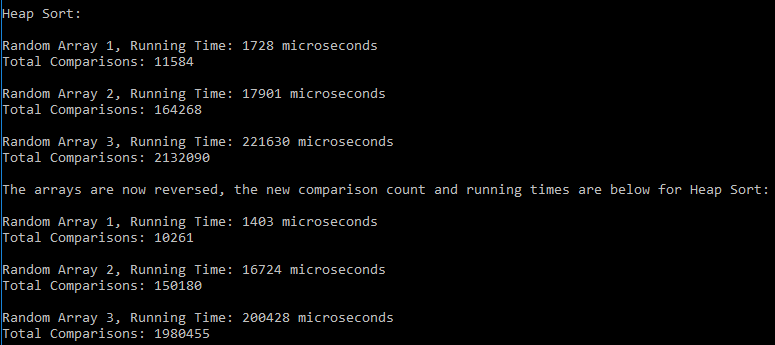


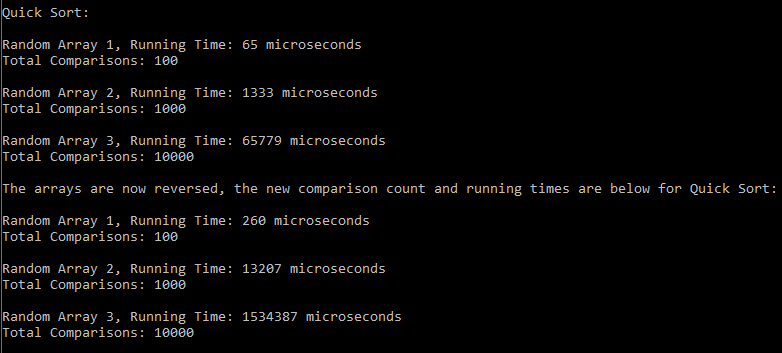


Below are the number of Comparisons for each Algorithm, along with the running times:









(Reverse Sorted Array means the array was arranged from largest to smallest, and then sorted again)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Insertion Sort** | | | | |
| **Array Size** | **Regular Sorted Array** | | **Reverse Sorted Array** | |
|  | **Total Comparisons** | **Running Time(micro sec.)** | **Total Comparisons** | **Running Time (micro sec.)** |
| **103** | **244731** | **846** | **493526** | **1536** |
| **104** | **24921386** | **84526** | **45186024** | **234119** |
| **105** | **181611343** | **8393018** | **136820893** | **15170638** |

The Insertion Sort algorithm iterates through an entire array, removing an element from the array, and putting it where it belongs from left to right. In it’s best running time, insertion sort runs on O(N).  
The worst and average case is when an array is in reverse order, and that will be O(N2). For comparisons, Insertion Sort will have (N-1) on best case, and worst and average case (1/2(*N*2 - *N*)) .  
 The results are consistent with the theoretical analysis. The total comparisons and running times for the code represent the average case, showing a linear increase in the output in comparisons and running time. The worst case is represented by the reverse sorted array, which shows the values in comparisons doubling in size.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Merge Sort** | | | | |
| **Array Size** | **Regular Sorted Array** | | **Reverse Sorted Array** | |
|  | **Total Comparisons** | **Running Time(micro sec.)** | **Total Comparisons** | **Running Time (micro sec.)** |
| **103** | **501** | **1543** | **3** | **1109** |
| **104** | **5001** | **12015** | **34** | **10912** |
| **105** | **50001** | **101616** | **407** | **97624** |

In Merge Sort, the array is split in half, then each half is recursively sorted, finally the merge algorithm combines the two halves. The Merge Sort algorithm makes most comparisons in the merge procedure where the size of the arrays are N/2. Thus, the equation for comparisons may be linearly (N/2 + N). For both best case and worst case, the Merge Sort algorithm will have a running time of O(N Log N), that is because Merge Sort splits any array in half, so regardless, every element in the array is compared at least once.  
 Here, the results are consistent with the theoretical part of the Merge Sort algorithm. The comparisons show linear consistency, and running time O(NLOGN )

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Heap Sort** | | | | |
| **Array Size** | **Regular Sorted Array** | | **Reverse Sorted Array** | |
|  | **Total Comparisons** | **Running Time(micro sec.)** | **Total Comparisons** | **Running Time (micro sec.)** |
| **103** | **11587** | **1728** | **10261** | **1403** |
| **104** | **164268** | **17901** | **150180** | **16724** |
| **105** | **2132090** | **221630** | **1980455** | **200428** |

The Heap Sort algorithm separates the array into a sorted and unsorted region by placing the largest element on the bottom and the smallest to the top. In Heap Sort, the number of comparisons made depends on the way elements are ordered. Regardless, the running time is the same, with O(N LOG N).   
 Because of this, the results show consistency. The comparisons and the running times in both regular and reversed arrays show similar results, of course the comparisons and times are different, but if seen broadly, then they are consistent. Other trials can be executed to show that the results will remain within range of each other.

For the quick sort array, the values used are **102, 103 , and 104**. Numbers larger than these would cause the program to exit.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Quick Sort** | | | | |
| **Array Size** | **Regular Sorted Array** | | **Reverse Sorted Array** | |
|  | **Total Comparisons** | **Running Time(micro sec.)** | **Total Comparisons** | **Running Time (micro sec.)** |
| **102** | **100** | **65** | **100** | **260** |
| **103** | **1000** | **1333** | **1000** | **13207** |
| **104** | **10000** | **65779** | **10000** | **1534387** |

For Quick Sort, the worst case is when the pivot is the largest or smallest element in the array. The worst running time is O(N2), and the best case is O(N Log N). For this program the pivot was not randomized. So for the Regular Sorted Array, we can assume the best case, and since the reversed array is descending from largest to small in order, then the reverse case must have the worst case.  
 The comparisons in the quicksort array are mostly done in partition, when they are compared to the pivot. The reason the comparisons are equal to the array is because all elements are compared to the pivot. The results are consistent with the theoretical analysis. The regular sorted array has a faster running time of O(N LOG N) (Best case) and the reversed order array has a running time that doubles when compared to the regular sorted array running time.

Other Runs:

