

7205 HW4

Name: Xuebao Zhao NUID: 002108354

Q1:

Algorithm analysis:

Insertion Sort:

In inserting sort, we first consider the first element as a sorted sequence and consider the second element to the last as an unsorted sequence. We scan the unsorted sequence from beginning to end, inserting each unsorted element into the appropriate position in the ordered sequence.

In the implementation process, a double-layers cycle is used, and the outer cycle go through all elements other than the first element, the inner layer cycle is to insert the current element into the sequence in front of the element and moves.

In insert sort, when the array is ordered, that's the best case, we just compare the current number with the previous one, and that's $n - 1$ times. Time complexity is $O(n)$. Worst case is that if the array is in reverse order, that's the most comparisons, and worst case, Time complexity is $O(n^2)$.

Heap Sort:

In heap sort, we first build a max heap. The way to build a max heap is to start at the last non-leaf node, compare the node with its children and swap parent node and maximum child node. Move to the child node and continue comparing with its child node until the heap become the max heap. Then change the head (maximum) and end. After that, we reduce the size of the heap by 1 and build a max heap again. Repeat the above steps until the size of the heap become 1.

Time complexity is always $O(n \log n)$. Because every time we get the maximum element we need to re max-heapify it.

Quick Sort:

In quick sort, we start by taking a number from the sequence as the base number and put the numbers which is larger to its right, less than or equal to its left side. Repeat the second step for the left and right intervals until each interval has only one number.

In the best case, partition evenly divided the array every time. The depth of the recursion tree is $\log n$. In the best case, the time complexity of the quicksort algorithm is $O(n \log n)$. In the worst case, each partition yields only a subsequence with one less than the previous partition, and the other subsequence is empty. So, we need to perform $n - 1$ recursive call. The time complexity is $O(n^2)$.

Approach:

First, define two groups of arrays. One is used to store the original data. This group includes BST, AVG, and WST. The other group is used to store the test data which need to be sorted. Generate data for group one:

BST: 1000 integers already sorted in ascending order.

AVG: 1000 randomly generated integers between 0 and 100,000.

WST: 1000 integers sorted in descending order.

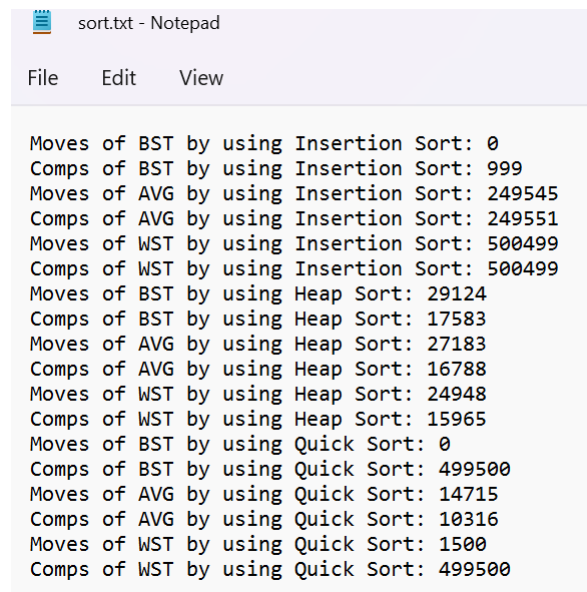
Reinitialize arrays in group two with arrays in group one before sorting the arrays by using above algorithms. Count the numbers of moves and comparisons for each sort and write values into 'sort.txt'.

Results:

Print the result:

```
The array is sorted!
Moves of BST by using Insertion Sort: 0
Comps of BST by using Insertion Sort: 999
The array is sorted!
Moves of AVG by using Insertion Sort: 249545
Comps of AVG by using Insertion Sort: 249551
The array is sorted!
Moves of WST by using Insertion Sort: 500499
Comps of WST by using Insertion Sort: 500499
The array is sorted!
Moves of BST by using Heap Sort: 29124
Comps of BST by using Heap Sort: 17583
The array is sorted!
Moves of AVG by using Heap Sort: 27183
Comps of AVG by using Heap Sort: 16788
The array is sorted!
Moves of WST by using Heap Sort: 24948
Comps of WST by using Heap Sort: 15965
The array is sorted!
Moves of BST by using Quick Sort: 0
Comps of BST by using Quick Sort: 499500
The array is sorted!
Moves of AVG by using Quick Sort: 14715
Comps of AVG by using Quick Sort: 10316
The array is sorted!
Moves of WST by using Quick Sort: 1500
Comps of WST by using Quick Sort: 499500
```

The contents in the 'sort.txt':



```
sort.txt - Notepad

File Edit View

Moves of BST by using Insertion Sort: 0
Comps of BST by using Insertion Sort: 999
Moves of AVG by using Insertion Sort: 249545
Comps of AVG by using Insertion Sort: 249551
Moves of WST by using Insertion Sort: 500499
Comps of WST by using Insertion Sort: 500499
Moves of BST by using Heap Sort: 29124
Comps of BST by using Heap Sort: 17583
Moves of AVG by using Heap Sort: 27183
Comps of AVG by using Heap Sort: 16788
Moves of WST by using Heap Sort: 24948
Comps of WST by using Heap Sort: 15965
Moves of BST by using Quick Sort: 0
Comps of BST by using Quick Sort: 499500
Moves of AVG by using Quick Sort: 14715
Comps of AVG by using Quick Sort: 10316
Moves of WST by using Quick Sort: 1500
Comps of WST by using Quick Sort: 499500
```

Graphs:

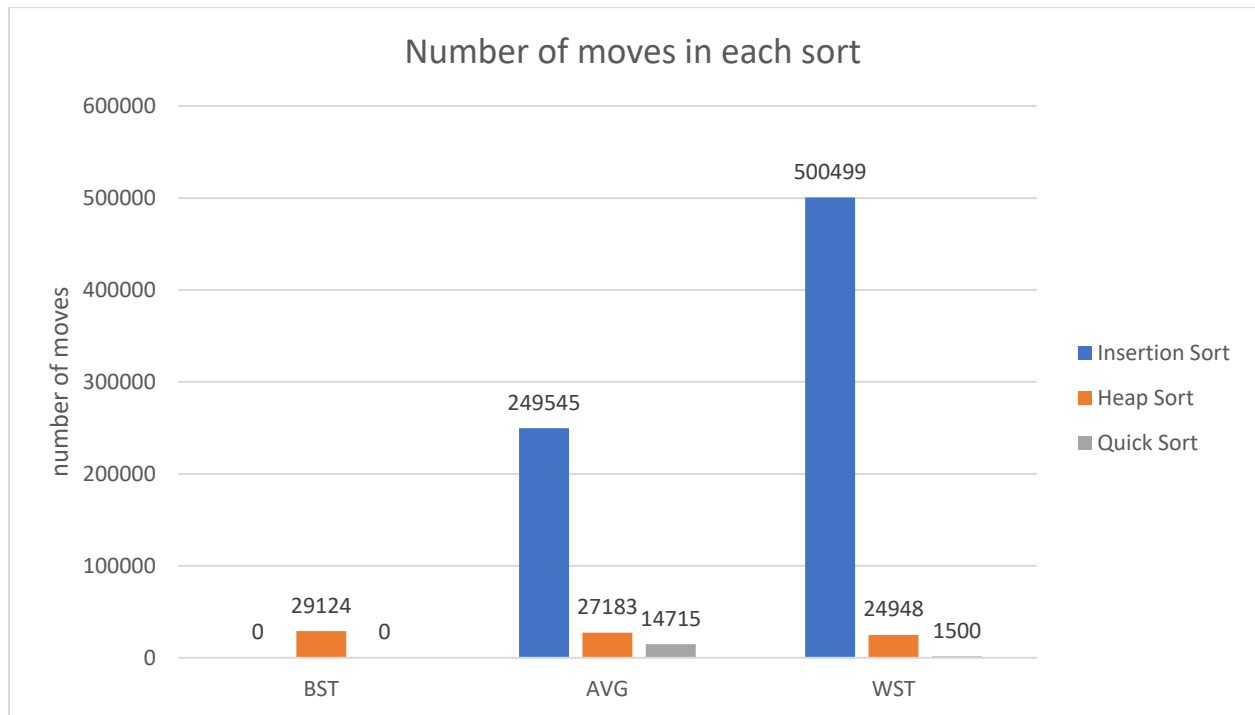


Figure 1 Number of moves in each sort

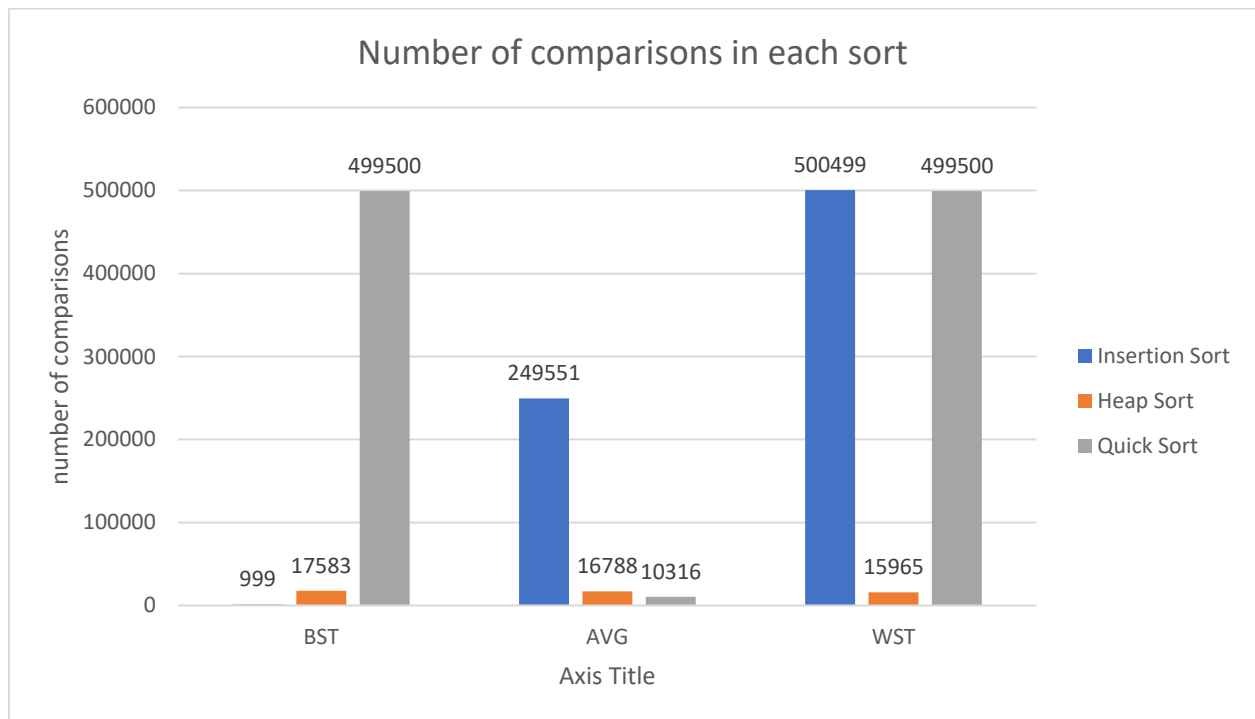


Figure 2 Number of comparisons in each sort

Why does the Insertion sort algorithm result in zero moves when sorting an already sorted array (the best case)?

A 'move' happens when the current element is smaller than its previous elements or when insertion action takes place. When the array is already sorted, the latter element is bigger than all the preceding elements, so there is no need to move.

Why does the Insertion sort algorithm result in 999 comparisons when sorting an already sorted array (the best case)?

Each element except the first element need to compare with its previous element, so there are at least $n - 1$ comparisons. Because the array is already sorted, the current element only compares with its previous element and will not run for loop.

Based on the two excel graphs, how does each algorithm perform under different scenarios (best, average, and worst)?

Insertion Sort: In insertion sort, both the number of moves and the number of comps increase from the best case to the worst case, and the increasing trend is huge, from zero to about 500k. Compared to the other two algorithms, the best case has the best result, and the worst case has the worst result.

Heap Sort: In heap sorting, all the results are average, moves are maintained at around 25k, and comps are maintained at around 15k. It performs best in number of comps among the algorithms when the array is reversed while performs worst in number of moves when the array is already sorted.

Quick Sort: It has the best perform in number of moves compared with other two algorithms. The number of moves is all very closed to 0. However, when it comes to the number of comparisons, it only performs well when the array is in the average situation. This sort get worst preforms both under best scenario and under worst scenario. The number of comps is around 500k.