Algorithm – Sorting and Searching

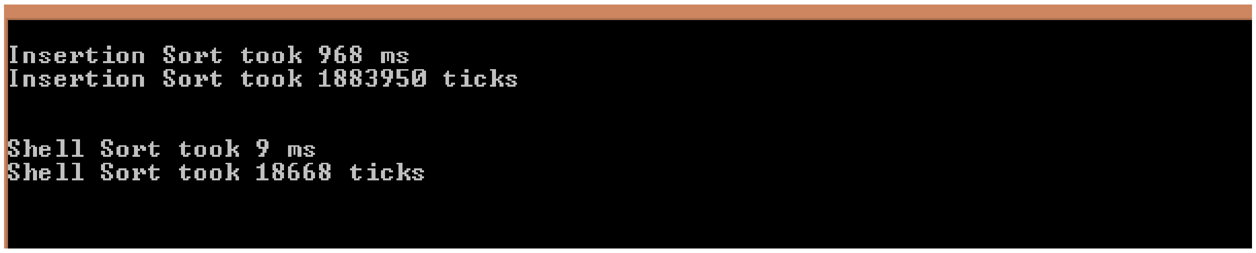
Student / Class Details

Full name: Madhupurna Dutta
Student ID: 102190136
Teacher: Tim Baird
Date / Time started: 29/06/2019

Tasks

1. A variation of Insertion Sort is Shell Sort.
How is it better than Insertion?
Implement both Insertion and Shell.
Use these implementations to sort the 15,000 numbers contained in unsorted_numbers.csv.

Ans:
Shell sort is faster than Insertion Sort. As Shell sort unlike insertion sort does not sort the entire array at once. Instead, it divides the array into noncontiguous segments, which are separately sorted by using Insertion Sort.
Once the segments are sorted, Shell Sort re-divides the array into less segments and repeat the algorithm until at last that the number of segments equals one, and the segment is sorted.

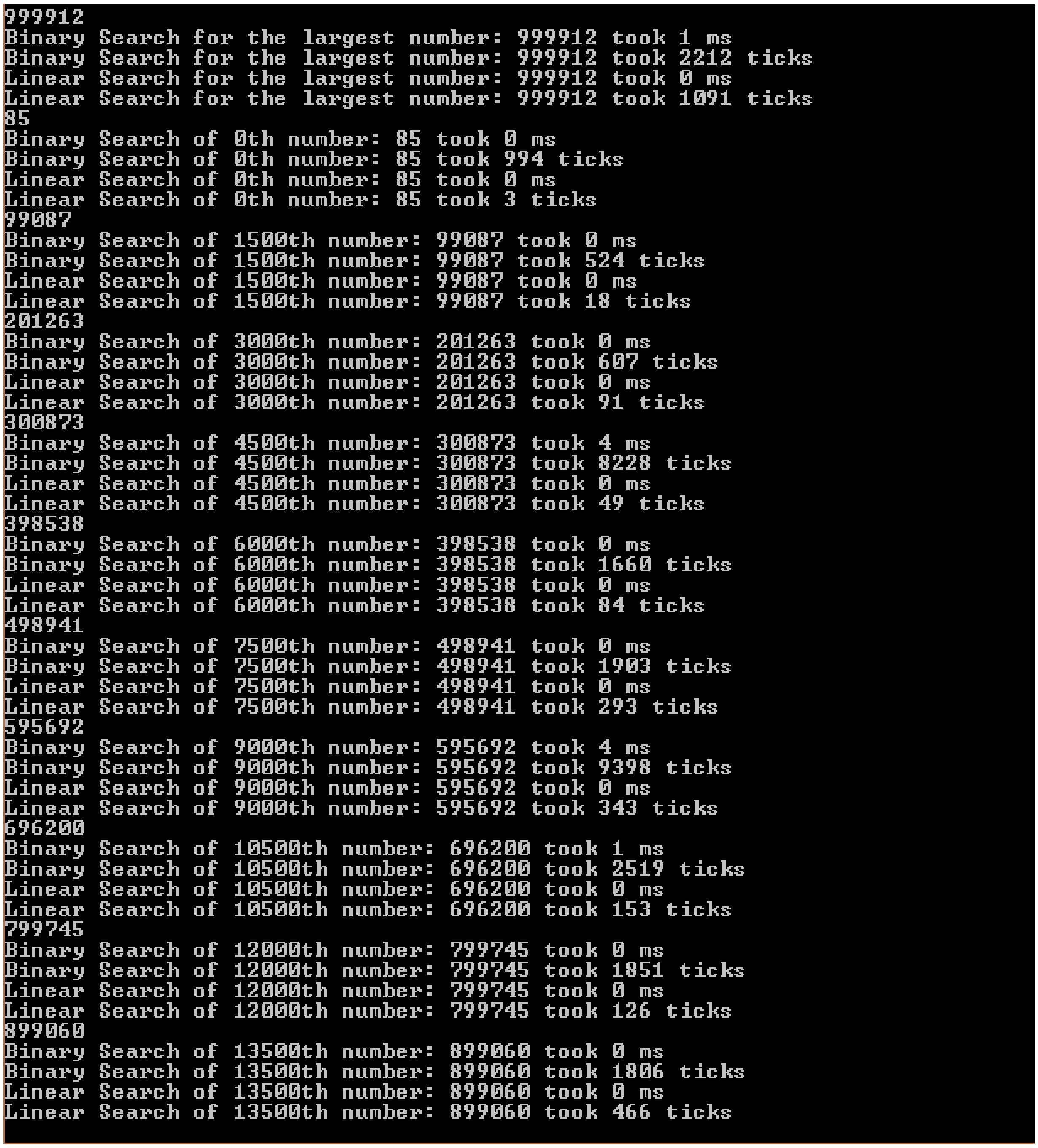


```
Insertion Sort took 968 ms
Insertion Sort took 1883950 ticks


Shell Sort took 9 ms
Shell Sort took 18668 ticks
```

2. Code a linear search and a binary search.
Perform a search on the largest number and every 1,500th number from the previous task.
Time the searches and compare against Big O notation for the searches.

Ans:



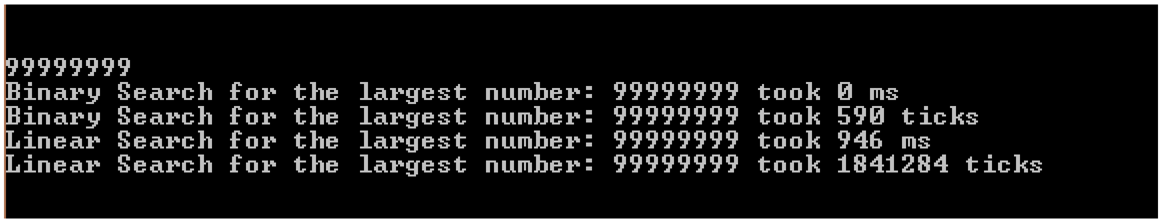
```
999912
Binary Search for the largest number: 999912 took 1 ms
Binary Search for the largest number: 999912 took 2212 ticks
Linear Search for the largest number: 999912 took 0 ms
Linear Search for the largest number: 999912 took 1091 ticks
85
Binary Search of 0th number: 85 took 0 ms
Binary Search of 0th number: 85 took 994 ticks
Linear Search of 0th number: 85 took 0 ms
Linear Search of 0th number: 85 took 3 ticks
99087
Binary Search of 1500th number: 99087 took 0 ms
Binary Search of 1500th number: 99087 took 524 ticks
Linear Search of 1500th number: 99087 took 0 ms
Linear Search of 1500th number: 99087 took 18 ticks
201263
Binary Search of 3000th number: 201263 took 0 ms
Binary Search of 3000th number: 201263 took 607 ticks
Linear Search of 3000th number: 201263 took 0 ms
Linear Search of 3000th number: 201263 took 91 ticks
300873
Binary Search of 4500th number: 300873 took 4 ms
Binary Search of 4500th number: 300873 took 8228 ticks
Linear Search of 4500th number: 300873 took 0 ms
Linear Search of 4500th number: 300873 took 49 ticks
398538
Binary Search of 6000th number: 398538 took 0 ms
Binary Search of 6000th number: 398538 took 1660 ticks
Linear Search of 6000th number: 398538 took 0 ms
Linear Search of 6000th number: 398538 took 84 ticks
498941
Binary Search of 7500th number: 498941 took 0 ms
Binary Search of 7500th number: 498941 took 1903 ticks
Linear Search of 7500th number: 498941 took 0 ms
Linear Search of 7500th number: 498941 took 293 ticks
595692
Binary Search of 9000th number: 595692 took 4 ms
Binary Search of 9000th number: 595692 took 9398 ticks
Linear Search of 9000th number: 595692 took 0 ms
Linear Search of 9000th number: 595692 took 343 ticks
696200
Binary Search of 10500th number: 696200 took 1 ms
Binary Search of 10500th number: 696200 took 2519 ticks
Linear Search of 10500th number: 696200 took 0 ms
Linear Search of 10500th number: 696200 took 153 ticks
799745
Binary Search of 12000th number: 799745 took 0 ms
Binary Search of 12000th number: 799745 took 1851 ticks
Linear Search of 12000th number: 799745 took 0 ms
Linear Search of 12000th number: 799745 took 126 ticks
899060
Binary Search of 13500th number: 899060 took 0 ms
Binary Search of 13500th number: 899060 took 1806 ticks
Linear Search of 13500th number: 899060 took 0 ms
Linear Search of 13500th number: 899060 took 466 ticks
```

The time complexity of a Linear Search is O(n). Which means to find an element in the worst-case computer will have to examine every single item in the array. This makes the Linear search not performant in sorting bigger arrays.

Binary Search is more efficient than linear search in sorting big arrays. It has a time complexity of O(log n). The Binary search only works on sorted lists. A binary search works by finding the middle element of a sorted array and comparing it to your target element.

Binary Search is faster while sorting arrays with large number of elements. For Smaller arrays (like the above example) Linear search works faster.

Example of Binary and Linear search performance on big arrays (like an array containing 100000000 elements) :



```
99999999
Binary Search for the largest number: 99999999 took 0 ms
Binary Search for the largest number: 99999999 took 590 ticks
Linear Search for the largest number: 99999999 took 946 ms
Linear Search for the largest number: 99999999 took 1841284 ticks
```

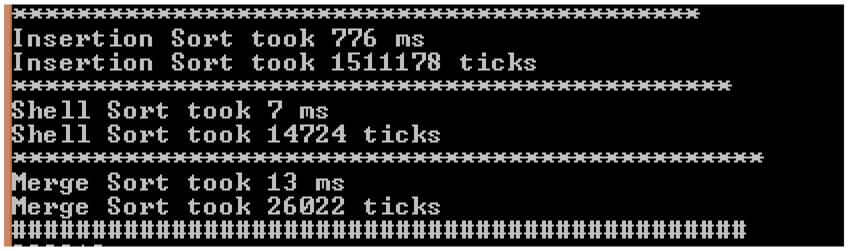
3. Could Merge Sort be run as a multi-threaded application?
Would there be likely to be a performance gain in doing so?
Why/Why not?

Ans: Multithreading is useful when the problem size is big. Threading is an expensive process, so it may not be useful to use multi-threading for sorting smaller problem sizes.

One possible approach to multi-threading in merge sort is to divide the task or the problem itself in multiple chunks or subsets and then sort each subset parallelly using threads and then combine the individual results at the end into one single sorted array.

Another possible approach can be enhancing the merge sort algorithm itself and introduce threading mechanism inside the sorting algorithm such that when a problem is given, the algorithm divides the task and assigns it to separate threads that will work parallelly and combines the result into a sorted array.

An Example from the previous task to show time comparison between Insertion, shell and merge sorts.



```
*******************************************
Insertion Sort took 776 ms
Insertion Sort took 1511178 ticks
*********************************************
Shell Sort took 7 ms
Shell Sort took 14724 ticks
***********************************************
Merge Sort took 13 ms
Merge Sort took 26022 ticks
##############################################
```

GitHub Link :

https://github.com/Madhu-Dutta/SortingSearchingAlgorithm