2019AIM1014 CS506

Kirtimaan Gogna **Assignment-1** 6 August 2019

---

**OBJECTIVE:** To compare time complexities and time taken by different sorting algorithms on different kinds of input and visualize the results.

**SORTING ALGORITHMS USED:**

1.) Bubble sort
2.) Insertion sort
3.) Selection sort
4.) Merge sort
5.) Quick sort

**TYPES OF INPUT GIVEN TO THE SORTING ALGORITHMS:**

1.) Random array.
2.) Sorted array.
3.) Reverse sorted array.

**OUTPUTS AND OBSERVATIONS:**

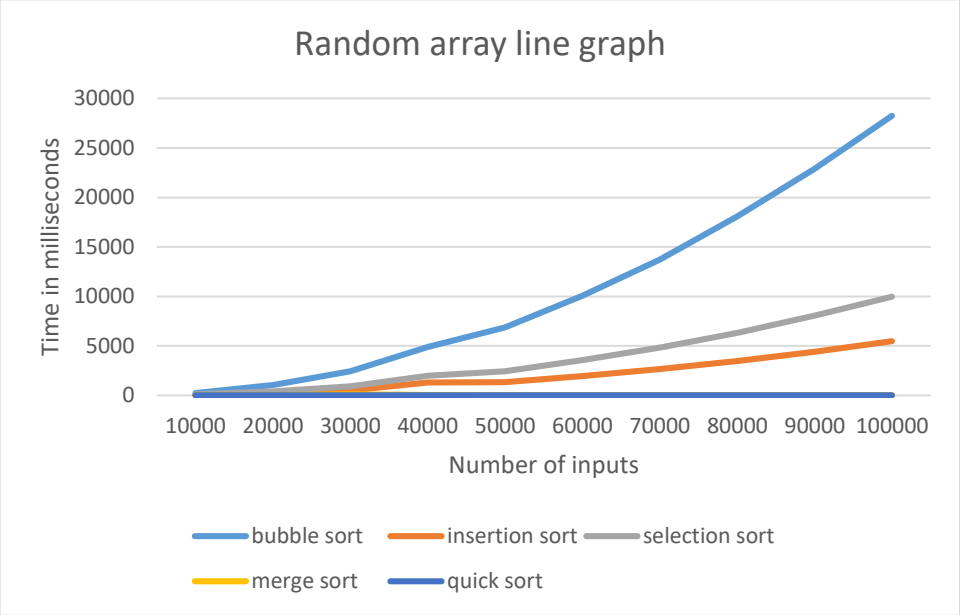
We randomly generate the inputs for random array by using rand() function for given input by user (between 0 to n-1).We define three different arrays and allocate them memory according to the size of the user input .We generate the sorted array data by directly feeding the random array with any sorting algorithm or by simply giving it input in increasing order. We generate reverse sorted array by copying the sorted data in reverse order . The subroutine called for doing this task is generate_num(int n) which takes the argument of the size of numbers to be generated. After the input arrays are generated these arrays are fed to different sorting algorithms to test their runtimes and analyse time complexities.

**CASE 1: When we feed random array to different sorting algorithms.**

**OUTPUT:**

| Input | 10000 | 20000 | 30000 | 40000 | 50000 | 60000 | 70000 | 80000 | 90000 | 100000 |
|---|---|---|---|---|---|---|---|---|---|---|
| Bubble sort | 234 | 1030 | 2437 | 4877 | 6877 | 10070 | 13716 | 18098 | 22900 | 28257 |
| Insertion sort | 47 | 204 | 485 | 1299 | 1344 | 1943 | 2652 | 3461 | 4390 | 5471 |
| Selection sort | 110 | 406 | 905 | 1992 | 2437 | 3567 | 4833 | 6322 | 8076 | 9968 |
| Merge sort | 0.001 | 18 | 90 | 50 | 0 | 0.009 | 0.011 | 0.012 | 0.008 | 0.016 |
| Quick sort | 1 | 0 | 20 | 15 | 0 | 7 | 9 | 11 | 12 | 13 |

Table 1: Time taken (in milliseconds) by different algorithms on random array.




Graph 1: Line graph for random input array vs time by sorting algorithms (in millisecs)

**OBSERVATIONS:**

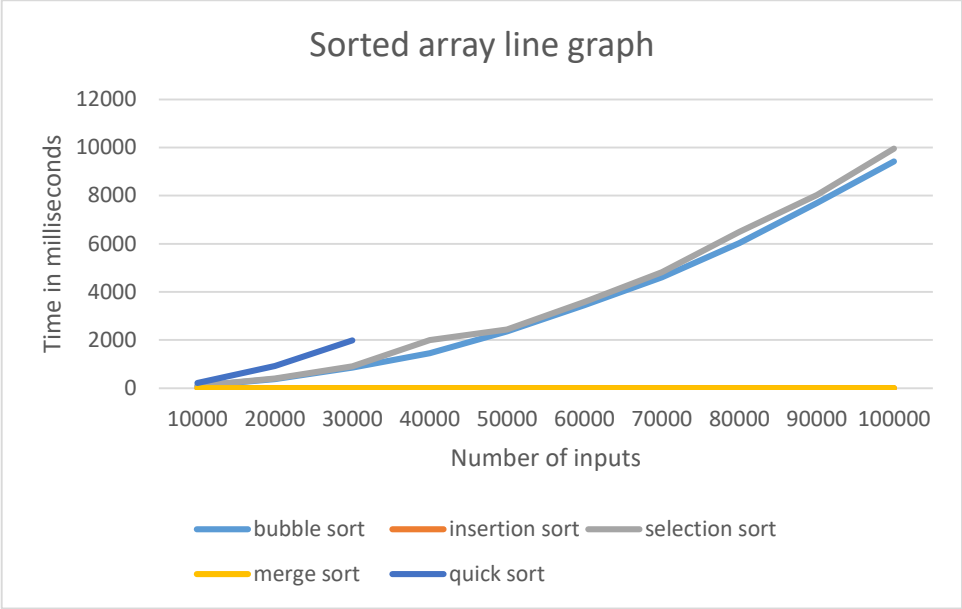
From the graph we can observe that bubble sort is performing very inefficiently even on significantly smaller inputs. It is because it has average and worst case time complexity of order of n squared where n is the input size. Insertion sort and selection sort has better efficiency compared to bubble sort. Merge sort performs very well on random integer data but it takes up huge auxiliary space of the order of the input size given. Quicksort is also performing well and it takes up just constant extra space.

**CASE 2: When we feed sorted array to different sorting algorithms.**

**OUTPUT:**

| input | 10000 | 20000 | 30000 | 40000 | 50000 | 60000 | 70000 | 80000 | 90000 | 100000 |
|---|---|---|---|---|---|---|---|---|---|---|
| bubble sort | 94 | 375 | 859 | 1458 | 2356 | 3452 | 4609 | 6026 | 7696 | 9418 |
| insertion sort | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| selection sort | 94 | 406 | 908 | 1993 | 2437 | 3586 | 4818 | 6491 | 8017 | 9946 |
| merge sort | 0 | 0 | 0 | 0.016 | 0.43 | 0.005 | 0.007 | 0.007 | 0.014 | 0.009 |
| quick sort | 217 | 922 | 1985 | | | | | | | |

Table 2: Time taken (in milliseconds) by different algorithms on sorted array input.




Graph 2: Line graph for sorted input array vs time by sorting algorithms (in millisecs)

**OBSERVATIONS:**

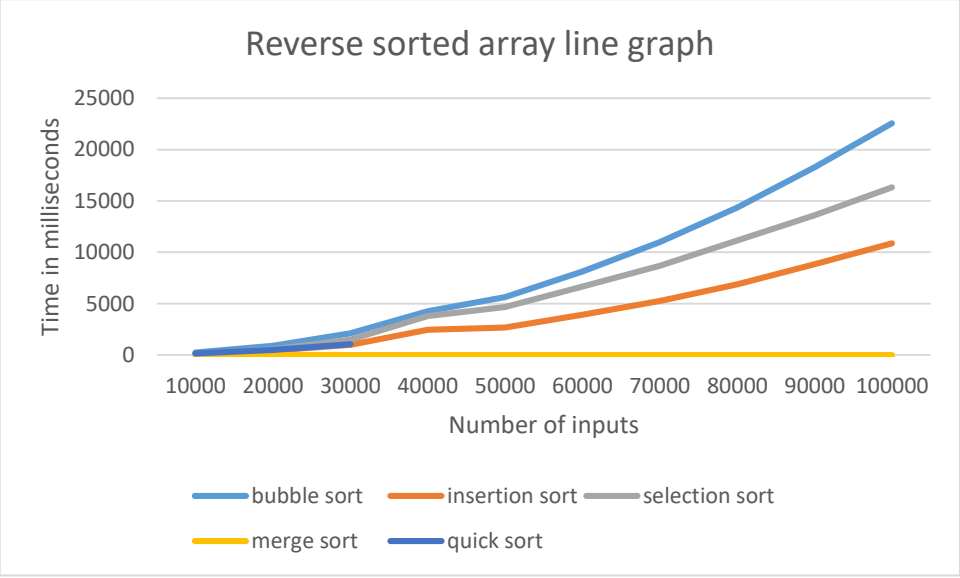
The fact that the input array is already sorted, it has no effect in decreasing the time complexity of bubble sort. Bubble sort still checks for each element, if it is “bubbled up” to its correct position. However bubble sort performance can be improved if we use a flag that is set if an exchange is made after an entire pass over the array. If no swapping of elements is being done than it certainly show that the array is already sorted, which gives best time complexity of linear time. Insertion sort performs the best in case of already sorted array. Selection sort takes almost the same time in almost all the input cases. Merge sort performs exceptionally well compared to above three sorting algorithms. Quicksort average case time complexity is order of nlogn , however in case of already sorted input array it triggers its worst case scenario in which its time complexity shoots up to the order of n squared. Hence it is the worst case for quicksort. However this can almost always be avoided if we use randomised quicksort or if the data is unsorted. It should also be observed that beyond certain limit of inputs our quicksort doesn’t give any result .It is because of the stack overflow caused by the recursive calling of the helping subroutines partition and quicksort itself.

**CASE 3: When we feed reverse sorted array to different sorting algorithms.**

**OUTPUT:**

| input | 10000 | 20000 | 30000 | 40000 | 50000 | 60000 | 70000 | 80000 | 90000 | 100000 |
|---|---|---|---|---|---|---|---|---|---|---|
| bubble sort | 234 | 906 | 2102 | 4255 | 5624 | 8128 | 10984 | 14363 | 18288 | 22558 |
| insertion sort | 109 | 437 | 984 | 2448 | 2671 | 3914 | 5273 | 6898 | 8853 | 10877 |
| selection sort | 105 | 547 | 1485 | 3800 | 4655 | 6677 | 8680 | 11143 | 13626 | 16317 |
| merge sort | 0 | 0 | 0 | 0 | 0.16 | 0.005 | 0.006 | 0.007 | 0.007 | 0.009 |
| quick sort | 140 | 500 | 1041 | | | | | | | |

Table 3: Time taken (in milliseconds) by different algorithms on reverse sorted array input.




Graph 3: Line graph for reverse sorted input array vs time taken (in milliseconds) by different sorting algorithms.

**OBSERVATIONS:**

It should be the worst case scenario for the different sorting algorithms because the order in which the elements are stored is in the exact opposite order. But this is not the case for all the sorting algorithms. It is observed that for bubble sort and insertion sort it is the worst case. For selection sort it doesn't really change the runtime for any time of input. Merge sort performance is the best. In quicksort it is the worst case scenario. It should be

observed that quicksort fails to give output after some large input array because of stack overflow.

**CONCLUSION:**

- Bubble sort performs badly in all the types of input arrays.
- Insertion sort performs best in case of already sorted input arrays.
- Selection sort performs comparably similar in all the types of input.
- Merge sort performs very well but takes huge auxiliary memory.
- Quicksort performs best in case of random array and performs badly in case of already sorted and reverse sorted arrays.

**REFERENCES:**


- Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein (2009). *Introduction to algorithms.*MIT press.
- Selection sort-GeeksforGeeks. Online available at: https://www.geeksforgeeks.org/selection-sort/
- Quicksort-GeeksforGeeks. Online available at: https://www.geeksforgeeks.org/quick-sort/
- Merge sort-GeeksforGeeks. Online available at: https://www.geeksforgeeks.org/merge-sort/