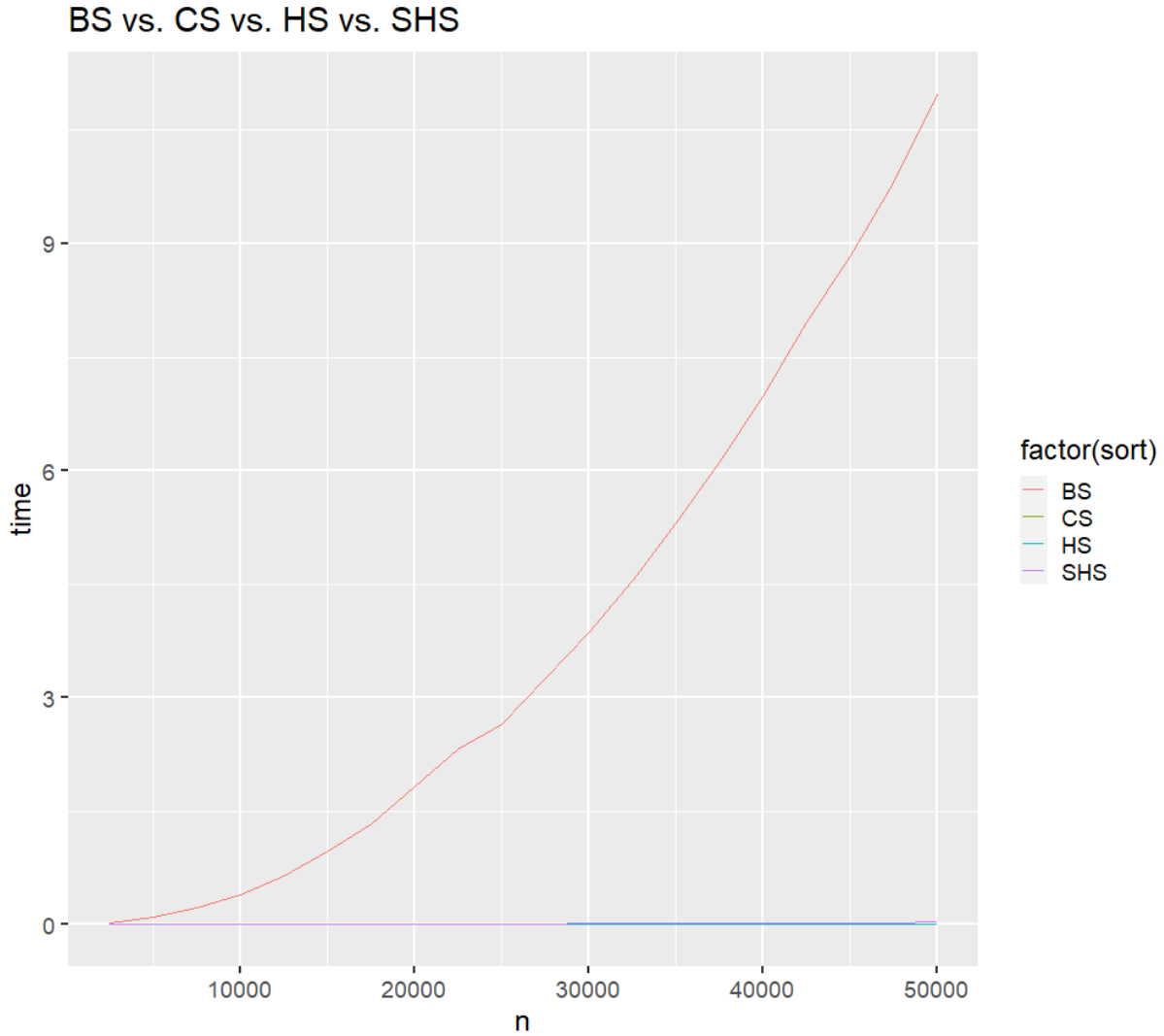
Report 1 - Sorting Algorithms - Bogna Kilanowska 148252

Comparison of speed of Bubble Sort, Heap Sort, Counting Sort and Shell Sort

Data:
Randomly generated integers by the rand() function from the “random” library.
20 measuring points (2500, 5000, 7500, …, 50000)

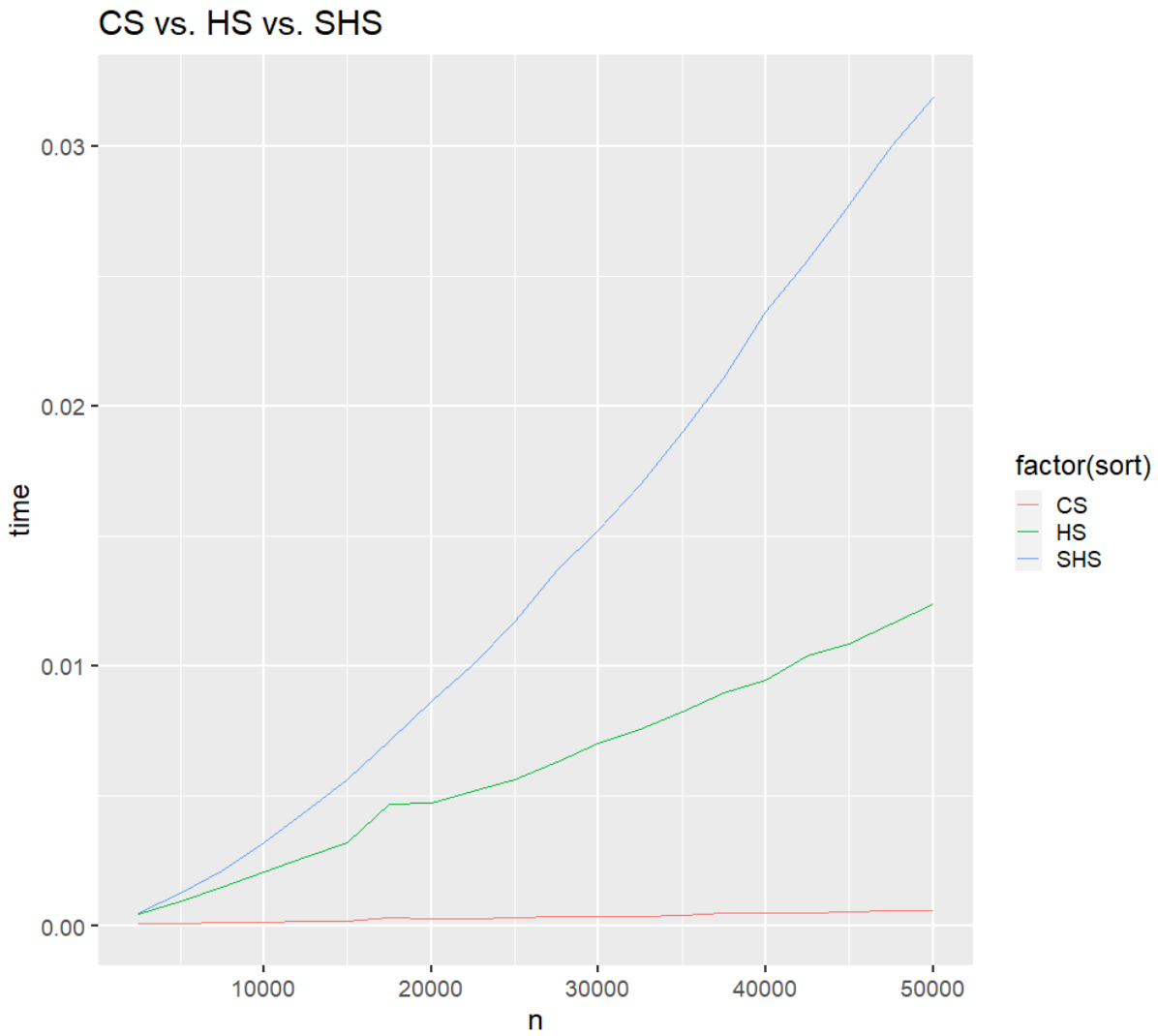
Chart:



Conclusions:
The average complexity of the BS is much worse than that of the rest of the sorting methods presented on this chart. From the shape of the red curve, it can be concluded that the complexity of the Bubble Sort equals O(n^2).

Comparison of speed of Heap Sort, Counting Sort and Shell Sort

To look closer at the shape of the rest of the curves we have to compare them separately from the BS.



The performance of the CS, HS, and SHS is much better than that of the BS (for 50000 elements Shell Sort was 300 times faster than Bubble Sort). Of these 4 methods, Counting Sort seems to be the best, but the reality is slightly different.

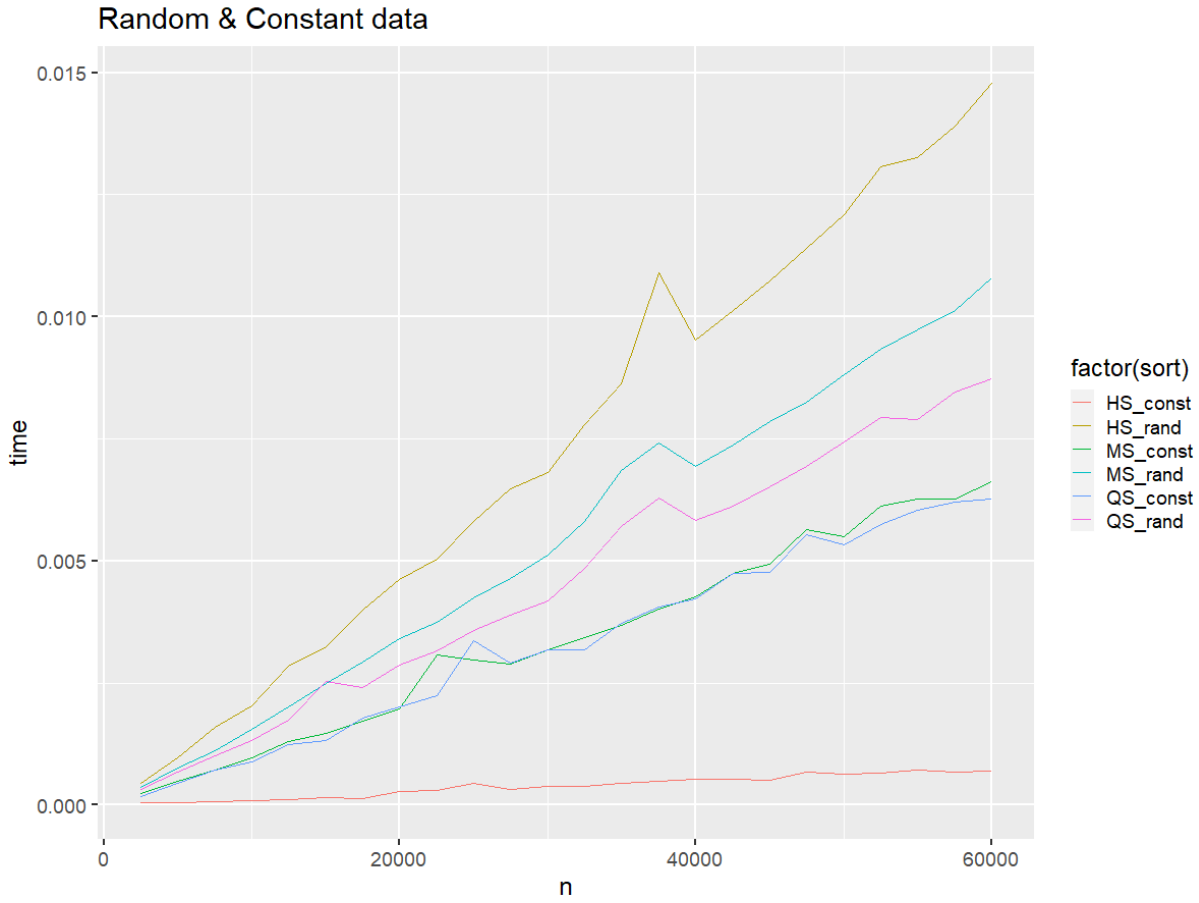
The average complexity of Heap Sort and Shell Sort is O(n*logn), so it only depends on the number of elements to sort. The average complexity of the Counting Sort is O(n + k), where k is the range of the elements. In this experiment, the range of the data was [0, 10000], while the number of elements to sort was up to 5 times bigger. In this situation, CS performed very well. There would be a different story when the range of the data was [0, 2^100].

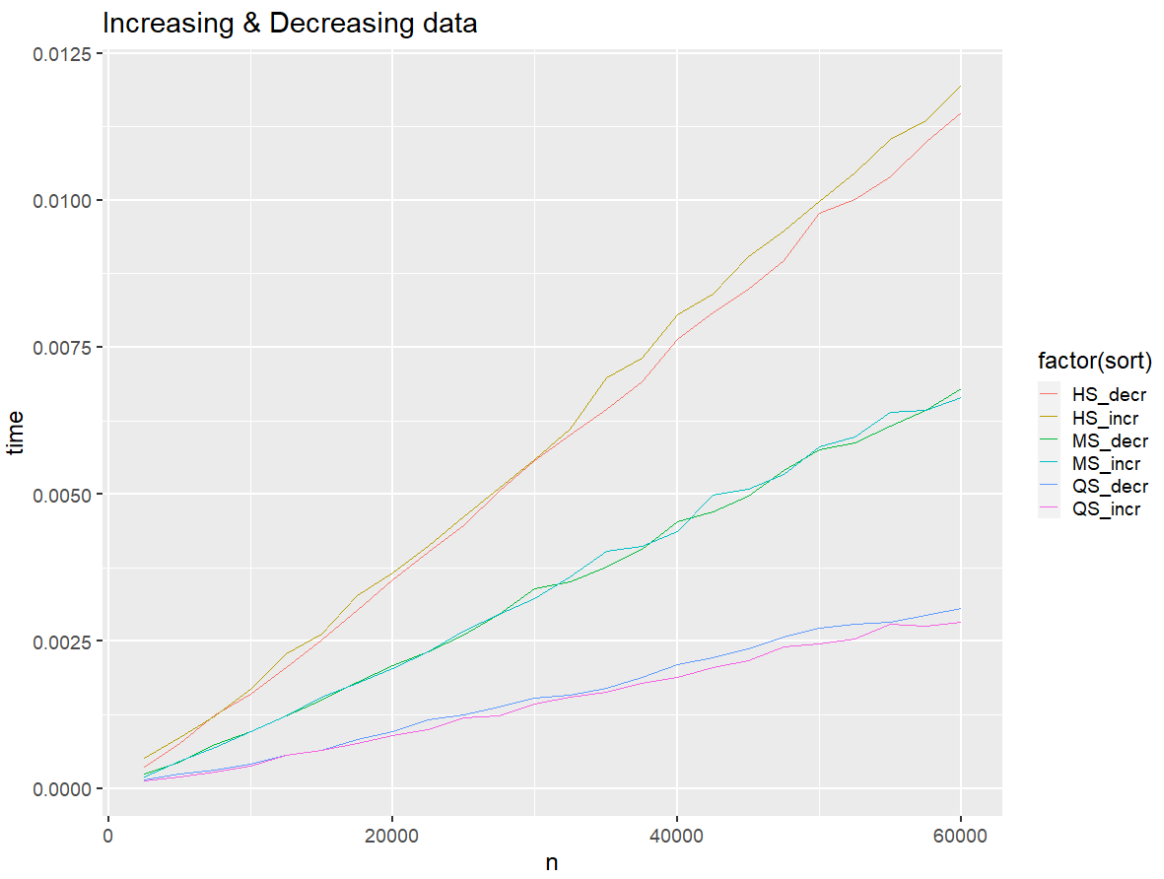
There is one more problem with Counting Sort. This method requires creating another array in which all occurrences of the elements will be counted - so it consumes more space in memory.

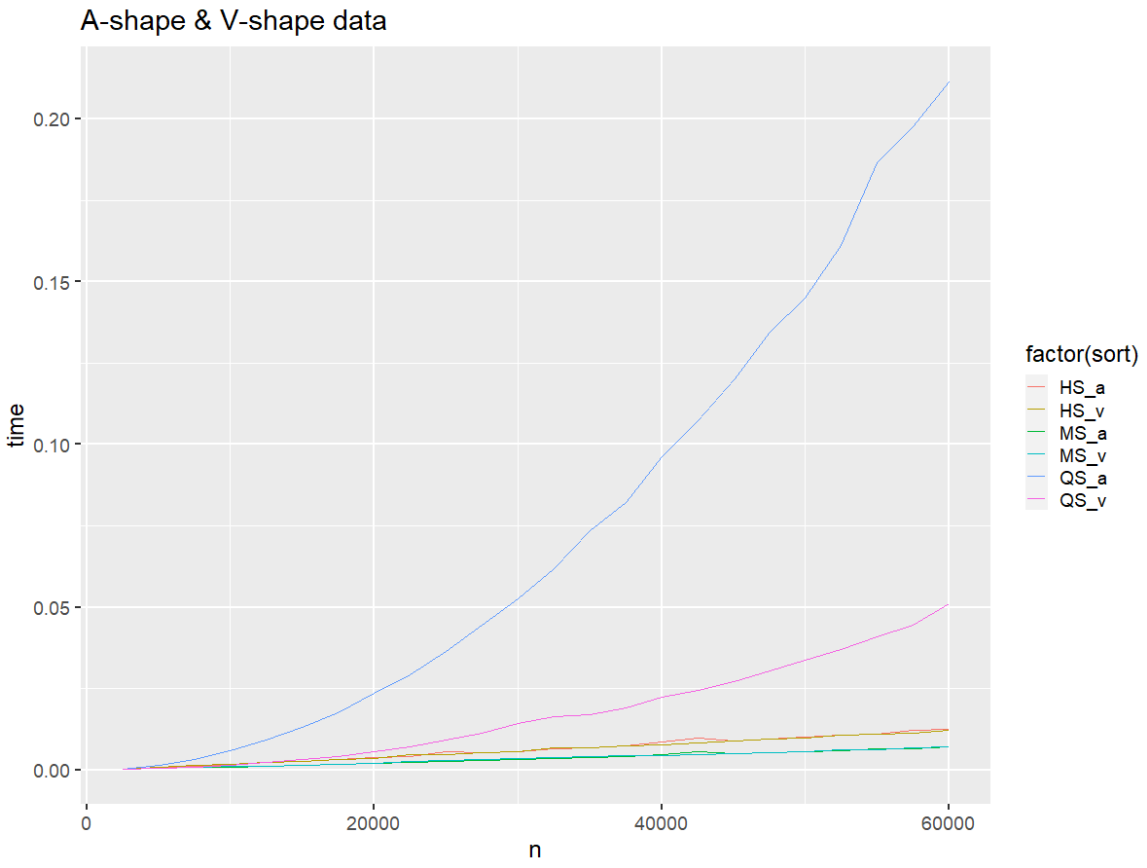
Comparison of the effectiveness of Quick Sort, Heap Sort and Merge Sort

Data:
Randomly generated integers by the rand() function from the “random” library.
24 measuring points (2500, 5000, 7500, …, 60000)

Charts:




Increasing & Decreasing data
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
time
0
20000
40000
60000
n
factor(sort)
HS_decr
HS_incr
MS_decr
MS_incr
QS_decr
QS_incr


A-shape & V-shape data
0.20
0.15
0.10
0.05
0.00
time
0
20000
40000
60000
n
factor(sort)
HS_a
HS_v
MS_a
MS_v
QS_a
QS_v

Conclusions:

From the charts, it can be concluded that Quick Sort performs much worse for A-shape and V-shape data sets than for the other ones. The shape of the curve in these 2 cases shows that the complexity of the QS in the worst scenario is O(n^2). But why is it the worst scenario? It is because the middle element which was taken as the pivot was the biggest one (A-shape) or the lowest (V-shape). It leads to dividing the array into two parts: one with only 1 element and the second one with n-1 elements and so on, so all of the elements will be taken as the pivot and the sorting procedure will require n divisions (in the best case scenario this number is log(2)n).

Although for the rest of the cases the complexity of the QS is much closer to its average, which equals O(n*logn). Quick Sort also doesn't require more memory (for example Merge Sort consumes some space to create additional arrays). Good complexity in the average case scenario, no need of consuming more memory, and quite simple procedure make the Quick Sort algorithm very useful - many libraries use this algorithm in their sort() functions.