Sorting algorithms

Bohdan Yeromenko
Faculty of Mathematics and Computer Science, University of Lodz

1. Introduction

We consider, implement and compare some famous sorting algorithms.

2. Sorting algorithms

Let's review three sorting algorithms:

2.1 Merge sort is a good choice if you want a stable sorting algorithm. Also, merge sort can easily be extended to handle data sets that can't fit in RAM, where the bottleneck cost is reading and writing the input on disk, not comparing and swapping individual items.

Time Complexity: Best :- O(n log(n))

Time Complexity: Average:- O(n log(n))

Time Complexity: Worst:-O(n log(n))

2.2 Quick sort is a good default choice. It tends to be fast in practice, and with some small tweaks its dreaded O(n^2)O(n2) worst-case time complexity becomes very unlikely. A tried and true favorite.

Time Complexity: Best : O(n log(n))

Time Complexity: Average: O(n log(n))

Time Complexity: Worst: O(n2)

2.3 Heap Sort is a good choice if you can't tolerate a worst-case time complexity of O(n^2)O(n2) or need low space costs. The Linux kernel uses heapsort instead of quicksort for both of those reasons.

Time Complexity: Best : O(n log(n))

Time Complexity: Average: O(n log(n))

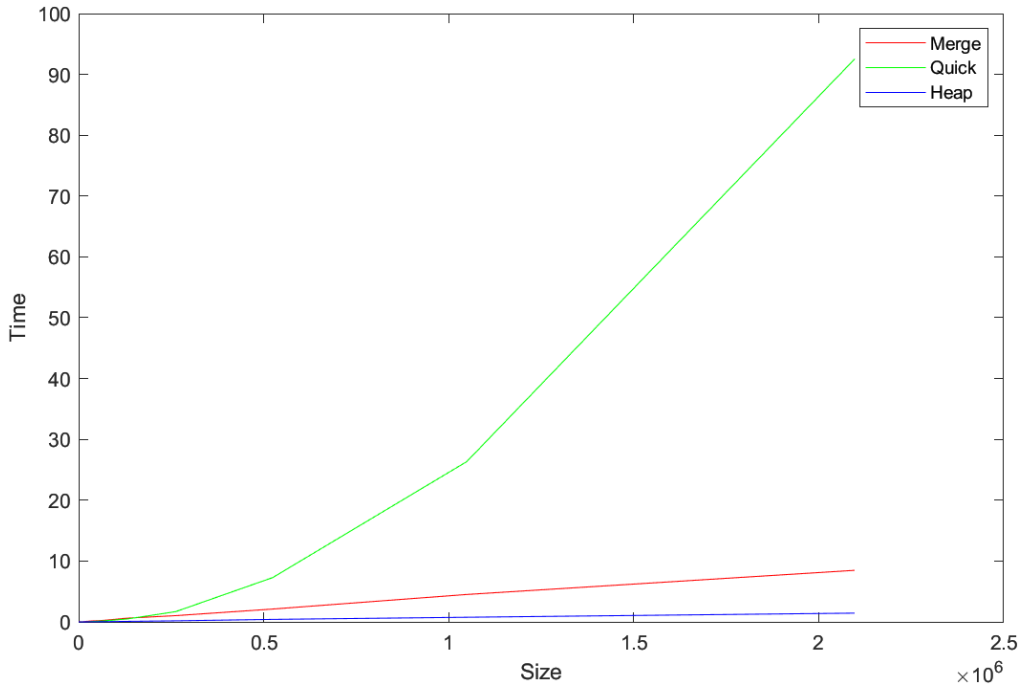
Time Complexity: Worst: O(n log n)

3. Result & Graph

"C:\Users\Bohdan Yeromenko\Desktop\Project1\CalculateTime.exe"

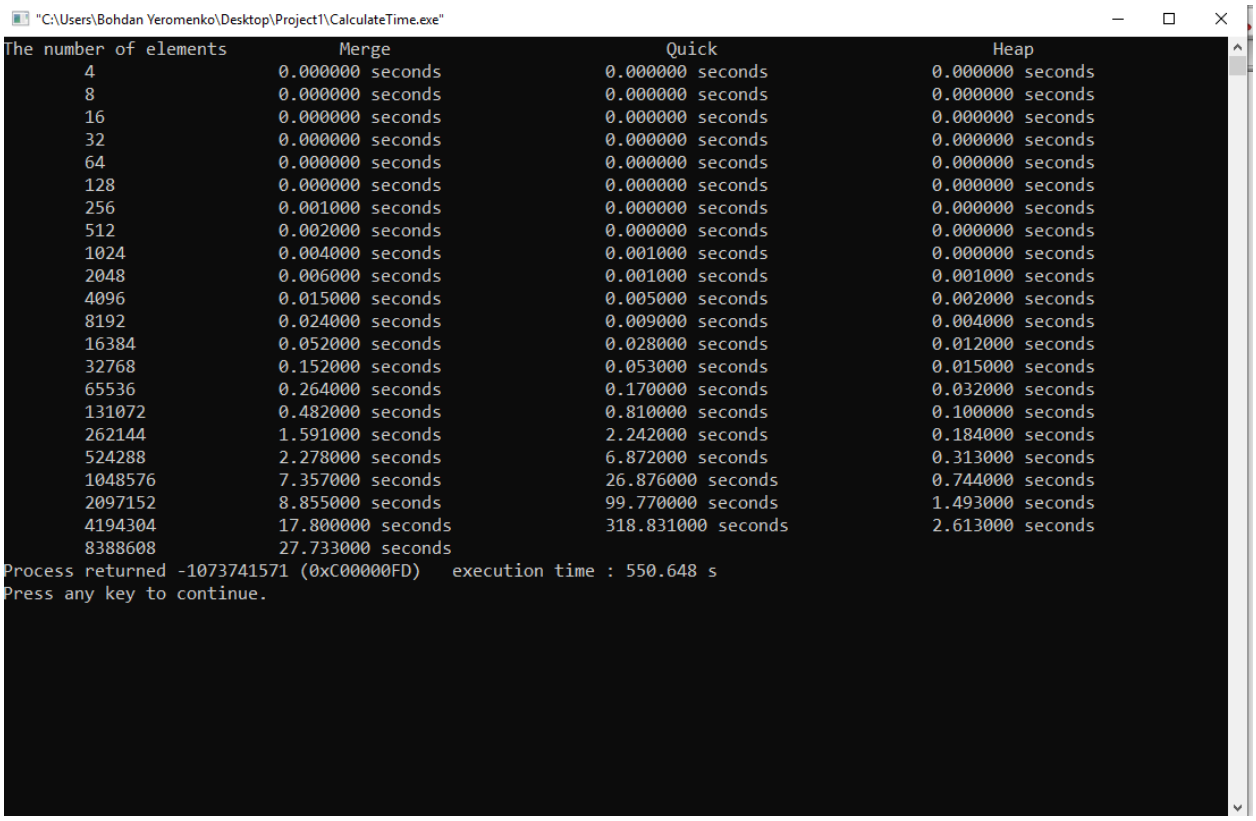
The number of elements	Merge	Quick	Heap
4	0.000000 seconds	0.000000 seconds	0.000000 seconds
8	0.000000 seconds	0.000000 seconds	0.000000 seconds
16	0.000000 seconds	0.000000 seconds	0.000000 seconds
32	0.001000 seconds	0.000000 seconds	0.000000 seconds
64	0.000000 seconds	0.000000 seconds	0.000000 seconds
128	0.000000 seconds	0.000000 seconds	0.000000 seconds
256	0.000000 seconds	0.000000 seconds	0.000000 seconds
512	0.001000 seconds	0.000000 seconds	0.000000 seconds
1024	0.003000 seconds	0.000000 seconds	0.000000 seconds
2048	0.005000 seconds	0.002000 seconds	0.001000 seconds
4096	0.011000 seconds	0.004000 seconds	0.001000 seconds
8192	0.024000 seconds	0.009000 seconds	0.003000 seconds
16384	0.054000 seconds	0.018000 seconds	0.006000 seconds
32768	0.095000 seconds	0.046000 seconds	0.013000 seconds
65536	0.199000 seconds	0.134000 seconds	0.029000 seconds
131072	0.398000 seconds	0.389000 seconds	0.066000 seconds
262144	0.790000 seconds	1.338000 seconds	0.130000 seconds
524288	1.620000 seconds	4.838000 seconds	0.281000 seconds
1048576	4.153000 seconds	21.370000 seconds	0.609000 seconds
2097152	7.090000 seconds	88.446000 seconds	1.296000 seconds

Process returned 0 (0x0)   execution time : 135.490 s
Press any key to continue.



3.1 Experiment

Let's slightly increase the number of elements for sorting:

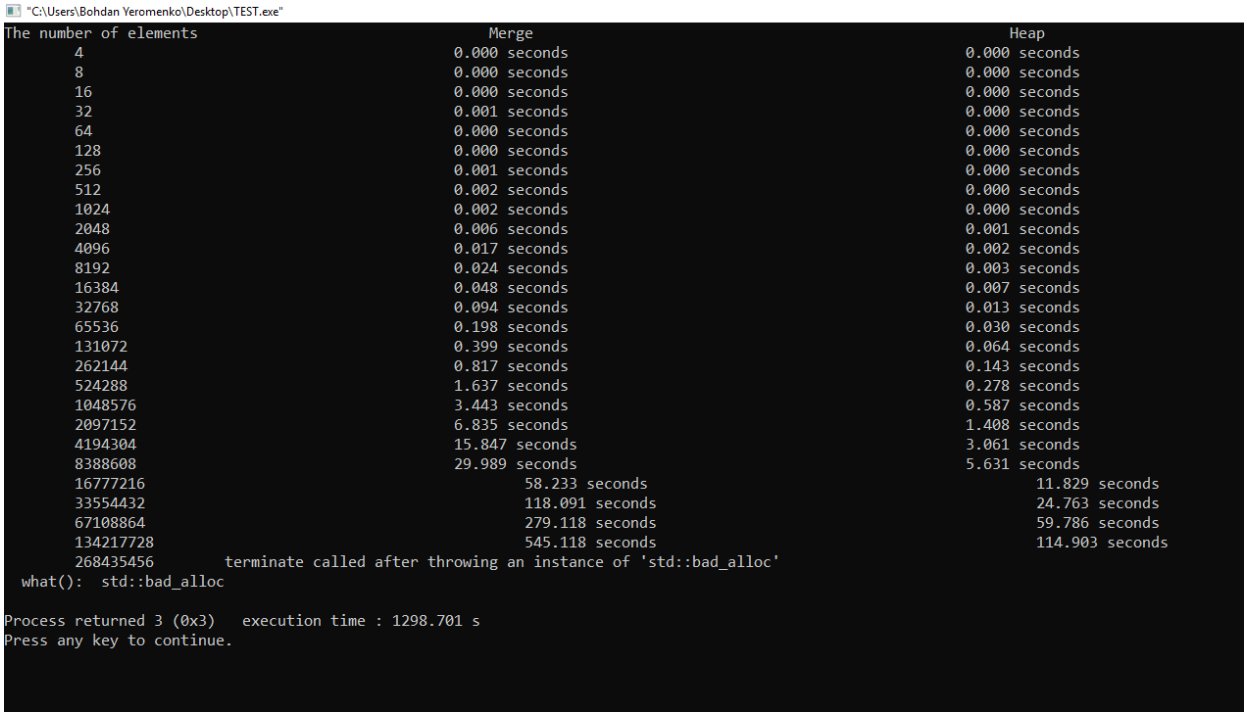


Here we clearly see the fact that brave quick sort algorithm will no longer be able to continue our Samurai Path. It is worth noting his courage and speed of work at the very beginning.

3.2 Experiment v2.0

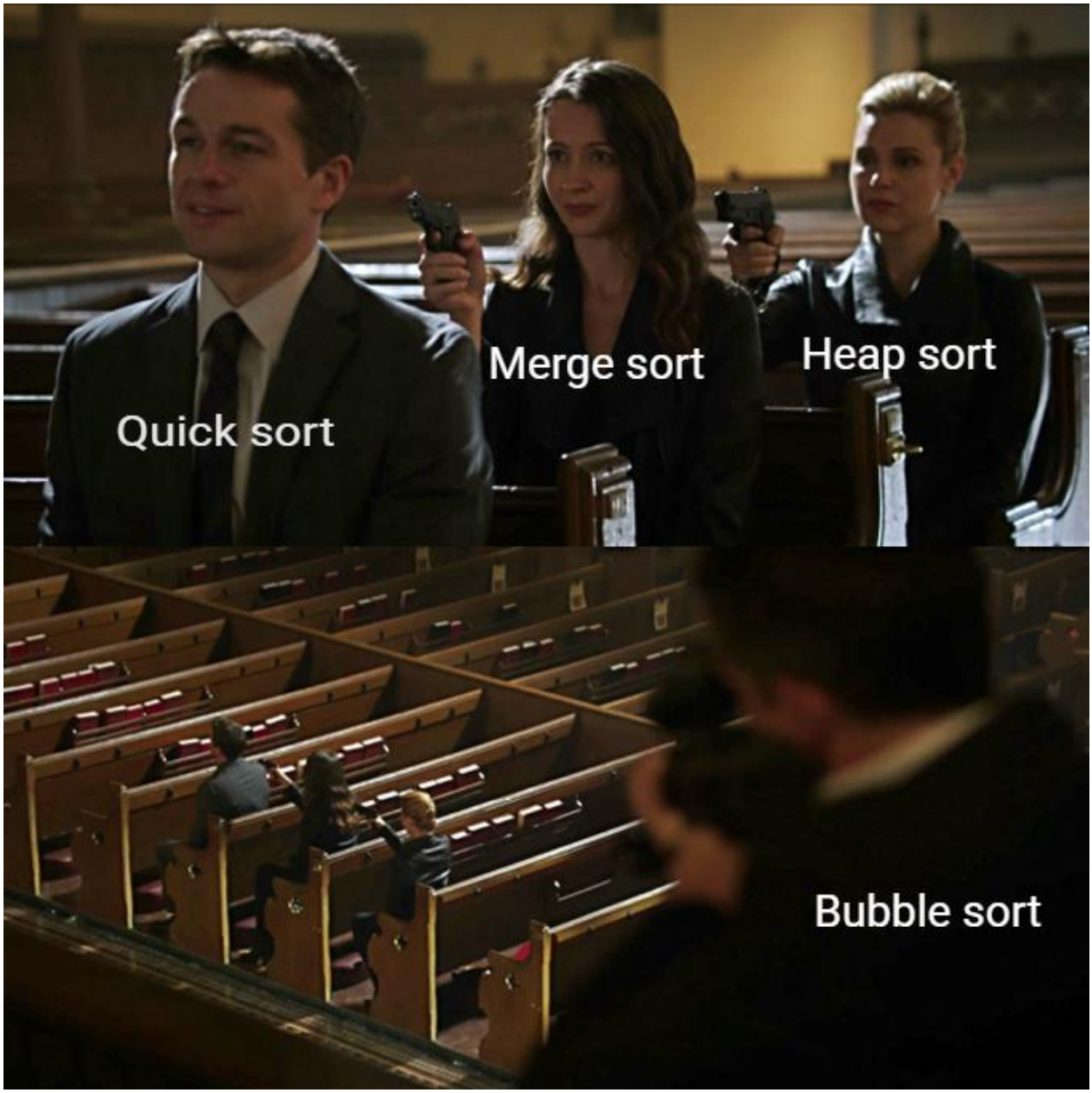
But why should we stop there, if we can drop the i5 - 7200U 2.5GHz with 8 GB RAM by just a sorting algorithm…

Unfortunately, our old friend will no longer be able to accompany us, so we will continue without him:



Merge sort fought hard, but fell as a victim to lack of RAM.

3. Conclusion



Based on the information received, we can conclude that *quick sort*, although it has a high speed and the relative simplicity of the algorithm, greatly degrades the speed of execution over time and shows up as an unstable. In the same time *merge sort* works slower than the previous one but ultimately overtakes quick sort on the second half of the array. Also its advantage is stability. And finally here comes a winner of our race – *heap sort*. This algorithm showed up as relatively fast and hardy, but it has such a disadvantage as instability.