CONCLUSIONS Rafał Stachowiak

EXERCISE I

Our knowledge of computational complexity confirms the reality about sorting alghoritms.

In small arrays of data all methods' sorting time is similiar, but when we increase the

number of elements in the array, bubble sort is the slowest one.

According to complexity in worstcase scenario:

bubblesort(BS): n^2

countingsort(CS): n

shellsort(ShS): n^(3/2) (fith function y=2^i+1 as gap finder according to wikipedia)

heapsort(HS): nlogn

t(BS)>t(ShS)>t(HS)>>>t(CS)

Memory activity:

Every method except CS is sorting in place. This can make us thinking that sorting not in

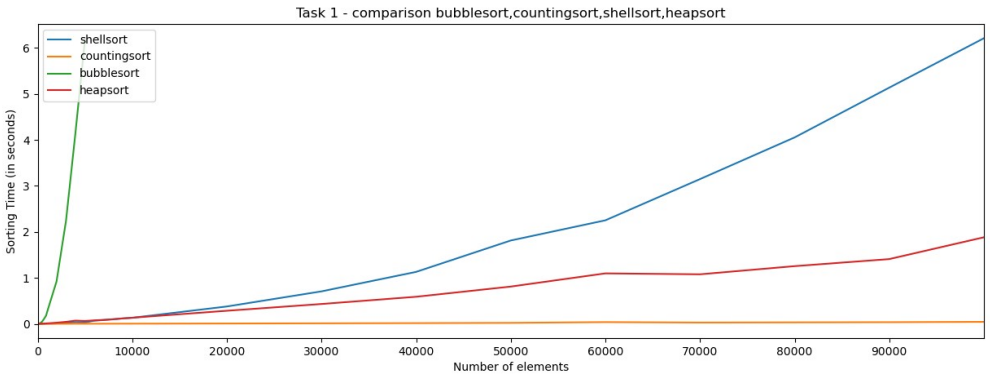
place is faster but this is not true since there are different sorting methods e.g. merge sort

which are not in place and have complexity nlogn (comparable with heap sort).

What makes bubble sort so inefficient? Unnecessary comparisons. For 1e4 elements it is almost 1e8 comparisons.

Countingsort allocates a lot of memory though.

I have learnt from this exercise that I will not use bubble sort anymore in my life.



EXERCISE II

As we know the computational complexity of heapsort in the worst case scenario is nlogn, for quicksort

is n^2 and for mergesort as in heapsort nlogn.

Heapsort is in this approach the slowest one and quicksort's execution time is the lowest.Quicksort sorts an array faster if elements are in in(de)creasing order than when they are distributed

normally.

Quicksort is not a good choice if data set is in A-shape or in V-shape because then an alghorytm exceeds

recursion depth = 1000.

My personal opinion is that we should use quicksort in pre-sorted data because it is very efficient.

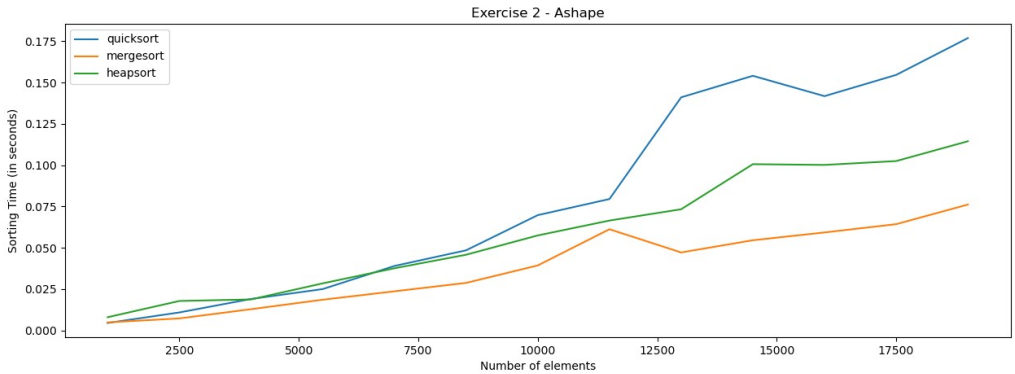
Median effect on Quicksort sorting time is an improvement to the quicksort alghoritm where instead of taking

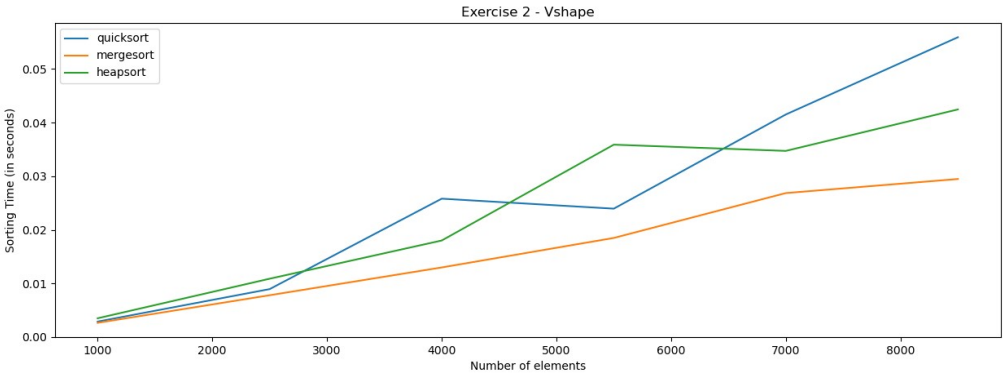
the middle value as pivot we are taking three elements from the array(often first last and middle one) and we are

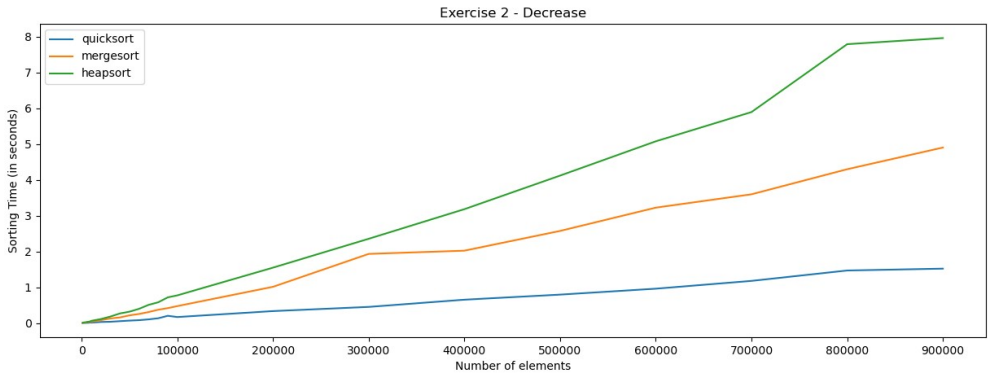
getting median value as a pivot. This will give us a better partitioning but it will extend sorting time by computing

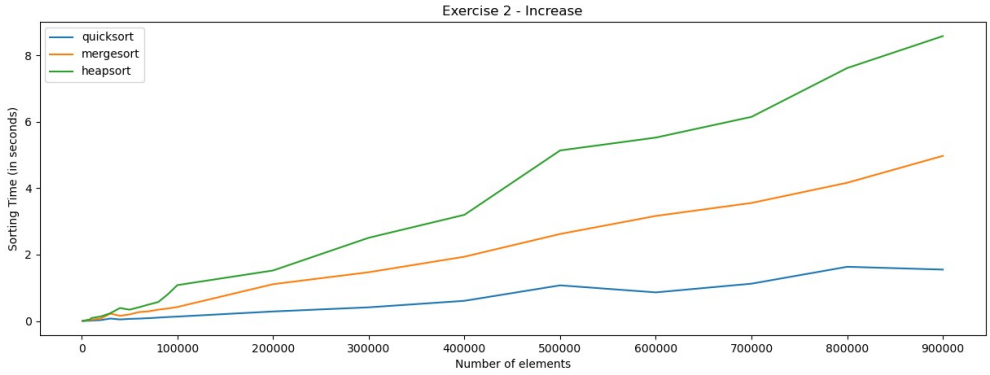
median value.

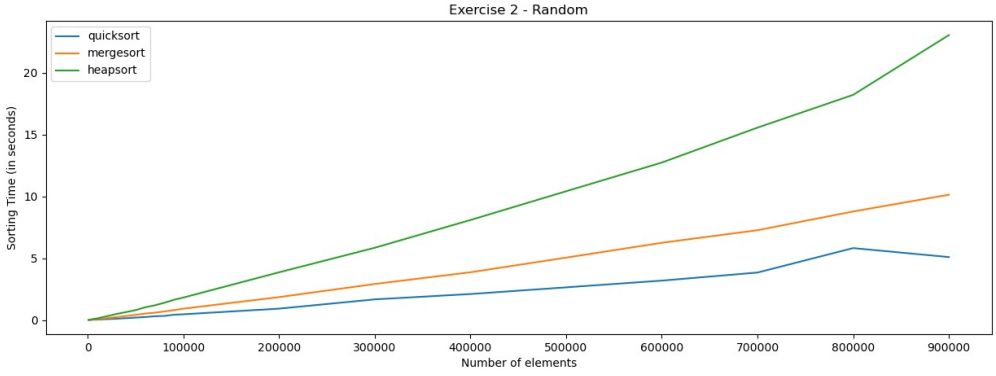
Here are the graphs showing the results:

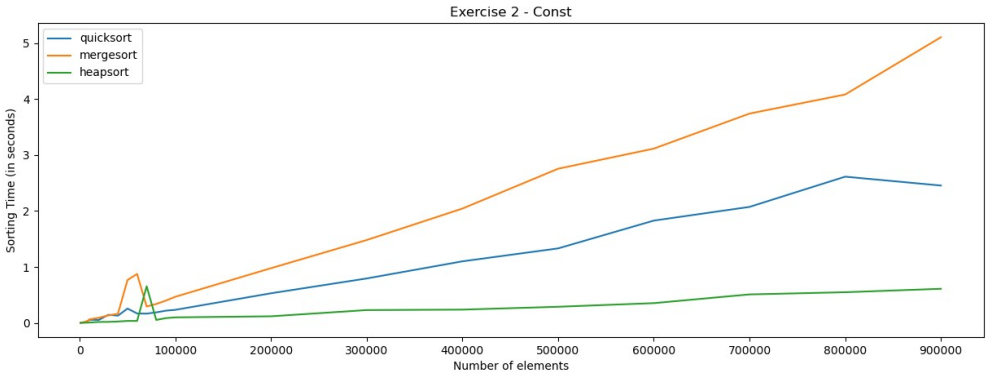



Exercise 2 - Vshape
quicksort
mergesort
heapsort
Sorting Time (in seconds)
0.00
0.01
0.02
0.03
0.04
0.05
1000
2000
3000
4000
5000
6000
7000
8000
Number of elements


Exercise 2 - Decrease
quicksort
mergesort
heapsort
Sorting Time (in seconds)
0
1
2
3
4
5
6
7
8
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
Number of elements


Exercise 2 - Increase
quicksort
mergesort
heapsort
Sorting Time (in seconds)
0
2
4
6
8
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
Number of elements


Exercise 2 - Random
quicksort
mergesort
heapsort
Sorting Time (in seconds)
0
5
10
15
20
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
Number of elements


Exercise 2 - Const
quicksort
mergesort
heapsort
Sorting Time (in seconds)
Number of elements
0
1
2
3
4
5
0
100000
200000
300000
400000
500000
600000
700000
800000
900000