Computer science - basic
Dejan Lužnic, 17.10.2018

Exercise A:
Problem:

WRITE A FUNCTION THAT SORTS 11 SMALL NUMBERS (<100) AS FAST AS POSSIBLE. ESTIMATE HOW LONG IT WOULD TAKE TO EXECUTE THAT FUNCTION 10 BILLION (10^10) TIMES ON A NORMAL MACHINE?

Since we are dealing with small array (11 numbers) I was looking for a solution to find out the best sorting algorithms for small arrays. I found that quickest sorting algorithms for this case are sorting networks. Insertion sort is one implementation of sorting network.

So I decided zo use for Insertion sort to solve exercise A which has speed of:

Average performance: O(n2).
Worst-case performance: O(n2)
Best-case performance: O(n log n)

Practical implementation:

1. I implemented insertion sort in Python console app.
2. I generated array with random numbers.
3. I used insertion sort on generated array and measured sorting time.

Results:

Sorting time: 1.03e-05 fraction of seconds.

** Sincesorting is so quick on so small array, we need to measure in fractional seconds, as seconds it just simply 0.*
** I run sorting of data 5 times, and sorting time n is average off all runs, since generated data can be different and so also sorting time.*

** In principle Shellsort (quite generalized version of inesrtion sort) could be a better choice. But it depends to much on input data and whether data is partially sorted. Average performance mainly depends on gap sequence, but in best case scenario can be better.*

10 billion times sorting time:

I assumed sorting time in theory:

Average performance is: (10^10) * $O(n^2)$.
Worst-case performance: (10^10) * O(n2)
Best-case performance: (10^10) * O(n log n)

In practice:

On my computer I calculated this will take 22516.5 seconds (6.25 hours).

Exercise **B**:
Problem:

WRITE A FUNCTION THAT SORTS 10000 POWERS (A^B) WHERE A AND B ARE RANDOM NUMBERS BETWEEN 100 AND 10000? ESTIMATE HOW LONG IT WOULD TAKE ON YOUR MACHINE?

For this solution I decided to implement quick sort algorithm since we are dealing with larger list. One reason that quick sort is efficient is that it first divides a large array into two smaller sub-arrays. It's a good solution also because average case is the same as worst case. So we do not have example when worst case take much more time than average.

Quicksort performance:

Average performance: O(n log n)
Worst-case performance: O(n log n)
Best-case performance: $O(n^2)$

Practical implementation:

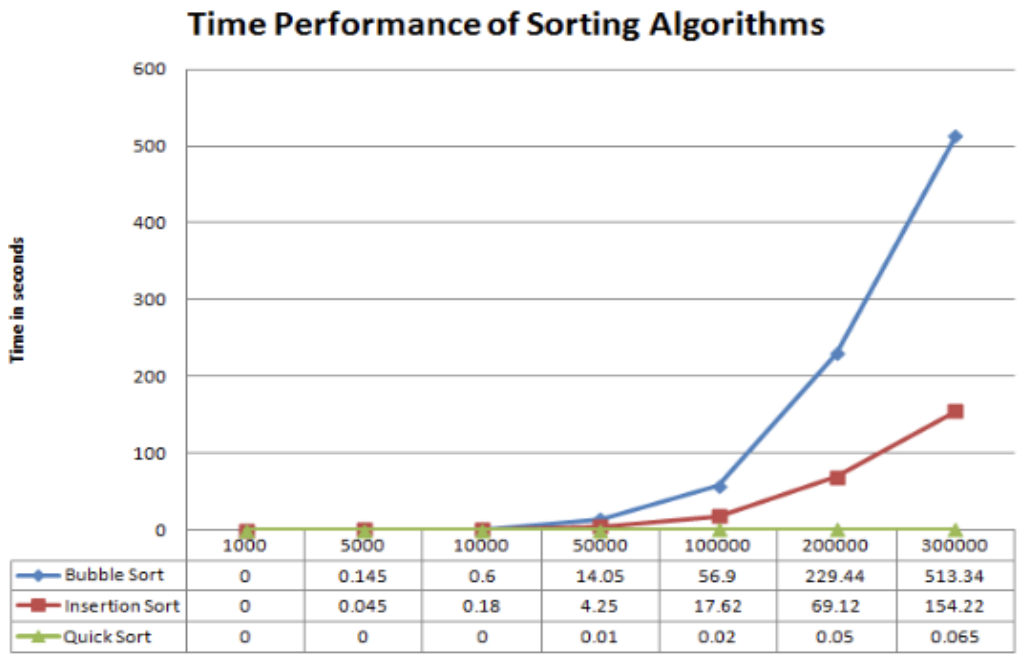
1. I implemented quicksort in Python console app.
2. I generated array with random numbers.
3. I used insertion sort on generated array and measured sorting time.

Results:

On my machine in practice it takes: average: 0.0378546.

**For comparison if I use this data with large numbers, and bigger array to sort, and use insertions sort from exercise A, execution time is: 0.65821328995 seconds which is quite big difference, which shows us how better quick sort if for larger lists.*

Because this first part of exercise is theoretical I also googled and checked for some comparison which are already made. So image bellow shows it all.



	1000	5000	10000	50000	100000	200000	300000
Bubble Sort	0	0.145	0.6	14.05	56.9	229.44	513.34
Insertion Sort	0	0.045	0.18	4.25	17.62	69.12	154.22
Quick Sort	0	0	0	0.01	0.02	0.05	0.065