Course: DD2325 - Exercise Set 2

Exercise 1: *Running time I*

You have the option of implementing one of four algorithms to solve a particular problem. The running times of the four different algorithms are $2^n, \frac{1}{2}n^3, 5n^2$ and $100n$.

Suppose you can afford 1000 seconds to solve the problem. How large a problem (size of n) can you solve with each of these algorithms ?

Suppose you buy a new computer that runs ten times faster. Now effectively you can spend 10^4 seconds on the problem where you spent 10^3 seconds before. What is the maximum size problem you can now solve using each of the four programs?

Note! Running time and other issues:

- If the code you are writing is only going to be used a few times, then the cost of writing and debugging $\gg$ running time. Therefore, in this case you generally should choose the algorithm that is easy to implment and not worry overly about the running time.
- If you expect to run your code on only small values of n then the constant factor may by the critical factor. So for instance conside the algorithms with running time of $100n$ and n^2 respectively. For which values of n will the first algorithm be faster?

Exercise 2: *Running time II*

What is the upper bound on the running time of Bubble sort?

Technical Note! The running time $T(n)$ of an algorithm is $O(f(n))$, read as "big oh of $f(n)$", if there exist postive constants c and n_0 such that for all $n \geq n_0$ then $T(n) \leq cf(n)$.

Exercise 3: *Sorting (demonsatration example)*

Here is a list of integers: $80, 30, 50, 70, 60, 90, 20, 30, 40$. Sort them using (a) Bubblesort and (b) selection sort.

Exercise 4: *Sorting - basic algorithms*

Here is a list of eight integers: $1, 7, 3, 2, 0, 5, 0, 8$. Sort them using (a) Bubblesort and (b) selection sort.

Exercise 5: *Running time of basic sorting algorithms*

Potentially swapping the elements in an array is a relatively expensive operation especially if the array is large. If this is the case should I implement either (a) Bubblesort or (b) selection sort? Why?

Exercise 6: *Bin sorting*

Assume you have a list of n postive integers which you know are bounded above by the postive number m. Describe an algorithm which can be used to sort this list in $O(n + m)$.

Exercise 7: *Sorting (demonsatration example)*

Use quicksort to sort the list: 3, 1, 4, 1, 5, 9, 2, 6, 5, 3

Exercise 8: *Sorting - quicksort*

Here are sixteen integers: 22, 36, 6, 79, 26, 45, 75, 13, 31, 62, 27, 76, 33, 16, 62, 47. Sort them using quicksort.

Exercise 9: *Sorting - radix sort*

Sort the list in the previous question using radix sort.

Exercise 10: *Sorting and running time*

Quicktime is fast and its average running time is less than all other currently known $O(n \log(n))$ sorting algorithms (by a constant factor, of course!). What simple improvement will speed up quicksort in some cases. (Hint: an $O(n \log(n)$ is not always faster than an $O(n^2)$ algorithm.)