Graded Assignment 1

Due date: March 21, 2018 at 20:00

This is an individual assignment. You may discuss it with others, but your formulations, your code, and all the required material must be written on your own. In any case, you must acknowledge the sources used and clearly mention any help received from colleagues.

Exercise 1. *(30% grade)*

Answer the following questions on the big-Oh notation.

Question 1: Order the following functions according to their asymptotic growth.

$$n^2 + 100n + 10 \qquad 4\log n^2 \qquad (\log n)^n \qquad 2^n \qquad 10^n \qquad \sqrt{n} \qquad 10^{10^{10}}$$

$$n! \qquad \log_{12} n \qquad (1 + 2 + 4 + \ldots + 2^n) \qquad \sum_{k=1}^{n} \frac{1}{k} \qquad 2^{2^n} \qquad n \log n \qquad n^n$$

$$\log\log n \qquad 2^{\log n} \qquad \log_3^3 n \qquad n^4 + 10n^5 \qquad \log\log \frac{n}{2} \qquad 9999! \qquad n\sqrt{n}$$

Functions that have the same order of growth can be listed in any order. You may find it useful to look up Stirling's approximation of the factorial function.

Question 2: Given a degree-3 polynomial $p(n) = a_0 + a_1 n + a_2 n^2 + a_3 n^3$, with $a_3 > 0$ and $n \in \mathbb{N}$, prove that $p(n) = \Theta(n^3)$. Note that you should use the formal definition of the Θ notation. (*Hint:* To prove that $p(n) = \mathcal{O}(n^3)$, find a constant c such that $p(n) \leq cn^3$ starting from some value n_0. Similarly you can prove $\Omega(n^3)$ bound.)

Question 3: Given any $\epsilon > 0$, prove that $\ln(n) = \mathcal{O}(n^\epsilon)$. In other words, no matter how small we choose ϵ, the function $\ln(n)$ still grows slower than n^ϵ. (*Hint:* L'Hôpital's rule)

Exercise 2. *(20% grade)*

Consider the following Python function which takes as input an array of numbers A and outputs a single number:

```
def calc(A):
  n =  len(A)
  res = 0

  for i in  range(0,n):
    for j in  range(i+1,n):
      res += A[j] - A[i]

  return res
```

Question 1: What does the function compute? What is the time complexity of the algorithm, as a function of the size of the list n? Justify your answer.

Question 2: Rewrite the given function such that it has the same functionality as the original one, but its complexity is $\mathcal{O}(n)$. By same functionality we mean that they produce the same result for all possible lists. You can test your code using the online judge system (http://domjudge.inf.usi.ch:8000/).

Exercise 3. *(30% grade)*

Consider the insertion-sort algorithm.

Question 1: What is the time complexity in the worst case? Give an example of a list of size n where this is achieved. Consider the following array for some given n and $k < n$:

$$[k+1, k+2, k+3, \ldots, n, 1, 2, 3, \ldots, k]$$

How many swaps will the algorithm perform?

Question 2: Complete the following table with the state of the array a after the completion of each iteration of the algorithm, for all different values of i from 2 to 6.

Step	a_1	a_2	a_3	a_4	a_5	a_6
Initial	3	8	1	2	6	4
$i = 2$						
$i = 3$						
$i = 4$						
$i = 5$						
$i = 6$						

Question 3: For some list of numbers a of size n, let $I(a)$ denote the number of pairs (i, j), $1 \leq i < j \leq n$ such that $a_i > a_j$. For example, $I([3, 2, 5, 2, 1]) = 7$. Prove that the number of swaps the insertion-sort algorithm performs on some list a is equal to $I(a)$. (*Hint:* consider what happens to $I(a)$ when insertion-sort performs a swap.)

Exercise 4. *(20% grade)*

In this exercise we will be packing n-dimensional boxes. Given two lists of integers of size n: $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ which represent the sizes of two n-dimensional boxes, we say that the first box can be packed into the second one if there exists some permutation $p : \{1, \ldots, n\} \to \{1, \ldots, n\}$, such that we have:

$$a_1 < b_{p(1)}, a_2 < b_{p(2)}, \ldots, a_n < b_{p(n)}$$

That is, we can rearrange the sequence b (which in turn corresponds to rotating the box in n dimensional space), such that all the corresponding dimensions of a are less than the corresponding dimensions of b. For example we can pack $[1, 3, 2]$ into $[2, 3, 4]$ by transforming $[2, 3, 4] \to [2, 4, 3]$, but there is no way to pack $[2, 3, 5]$ into $[3, 3, 6]$. Implement a python function `can_pack(a,b)` that given the lists a and b of size n, returns *true* if the first box can be packed into the second one, and *false* otherwise. You may also implement other helper functions if needed. The running time of your solution must be $\mathcal{O}(n^2)$. You can test your solution using the online judge system.

Exercise 5. (extra) *(5% grade)*

Consider the following sorting algorithm:

$A \leftarrow$ array of n *distinct* numbers
while A is not sorted **do**
  $A \leftarrow$ random permutation of A
end while

What is the the expected running time of the above algorithm? Justify your answer.