COMP3112 Analysis of Algorithms
Fall 2024
HW #1 (75 points)
Due Date: 6th Nov, Wednesday 23:59

This is a strictly individual assignment. That means, you are not allowed to share your answers with anyone. You are only allowed to use your lecture notes and the textbook.

When asked for a complexity analysis of some algorithm in the assignment, your complexity analysis should include clear indication of the following items: ***the size of the problem***, the ***basic operation involved*** in the analysis (mostly the *comparison* operation should do the purpose, unless there is no comparison involved in the algorithm, or some other operation (e.g., multiplication, etc.) has more weight on your overall analysis, and is the determinant factor on the result), and the ***formula*** (a recurrence relation or a summation series, etc.) for which the solution gives the resulting complexity. Unless otherwise specified, you are expected to solve the formula you devised, and find out or prove the complexity asked in the question.

Please use the pseudocode notation that the textbook has adopted, and we have been using in our lectures, while writing your algorithms. Use left arrow (←) for your assignments, and single equal sign (=) for your equality comparisons, etc.

---

1. (12,5 points) Given functions $f(n) = nlog_2 n$, $g(n) = nlog_{10} n$, $h(n) = n^2$, prove that;

a. $f(n) \in \theta(g(n))$

b. $f(n) \in O(h(n))$

2. (12,5 points) You are given an array A of integers and an integer k. Design and write the pseudocode for a $\theta(n^2)$ (worst-case) algorithm that finds a sequence of elements placed in consecutive indexes of the array whose sum is k. Return the beginning index of the sequence. Return -1 if there is no such sequence in A such that their sum is equal to k. Do a time complexity analysis on your algorithm.

3. (12,5 points) Assume that we are given an array of integers. Our aim is to find the most frequent element (the element that appears the most number of times, not necessarily consecutively) in this array.

Design and write the pseudocode for a brute-force $\theta(n^2)$ (worst-case) algorithm. Do a complexity analysis to show that your algorithm is $\theta(n^2)$. What about your algorithm's best case and average case behaviors?

4. (12,5 points) Design and write the pseudocode for an algorithm to rearrange elements of a given array of n real numbers so that all its negative elements precede all its positive elements. You should use an extra array of the same size with the input array in your algorithm, (i.e., the space complexity of the algorithm, considering the input itself as an intrinsic part of the problem and hence we assume that it does not contribute to the space complexity), should be linear $(\theta(n))$. Your algorithm should work in linear time complexity as well. Prove that the time complexity of your algorithm is $\theta(n)$. What about best case, and average case time complexities?

5. (12,5 points) Closest pair problem on a single dimension. We have n points sprinkled on the x-axis of our cartesian space (i.e., y-coordinates of all of the points are zero). We want to find the distance between the closest of these two points, i..e, the minimum distance between all point pairs on the x-axis. Devise a ***divide-and-conquer algorithm*** that finds the distance between the two closest points among n distinct points spread on the x-axis. The input to your algorithm is an array of values,

representing the x-coordinate values of the given points (no need for y-coordinates, since all are zero), and your algorithm should return the minimum distance between any point pairs. Do a complexity analysis of your algorithm and indicate its time complexity. Support your analysis by proper arguments about the characteristics of the problem.

Algorithm *closestPairOneDimension (P[0..n])*
//Computes the minimum distance between the
//closest pair of n points placed on the x-axis,
//with the **divide-and-conquer** technique.
//Input: An array representing n points on x-axis.
//Output: The distance between the closest point pair.

6. (12,5 points) Consider the following algorithm.

```
Algorithm whatIs(n)
        if (n%2 != 0) then
                return 0
        if (n = 0) then
                return 0
        return n*n + whatIs(n-2)
```

a. What does this algorithm compute?
b. Which strategy does it follow to solve the problem?
c. What is the time-complexity of this algorithm? What is its basic operation? Comparison, Multiplication, Addition? Does your complexity analysis depend on which basic operation you choose? Please explain and validate your answer.