School of Computer Science
University of Guelph

CIS*3490 The Analysis and Design of Algorithms

Winter 2021
Instructor: Fangju Wang

Assignment 1 Guide

Note: for questions from the textbook, there are hints in the book. They may be helpful. Please check.

1. Use $\lim_{n\to\infty} f(n)/g(n) = c$ and the theorem "If $t_1(n) \in \Theta(g_1(n))$ and $t_2(n) \in \Theta(g_2(n))$ then $t_1(n) + t_2(n) \in \Theta(\max\{g_1(n), g_2(n)\})$" to find the Θ efficiency classes and to prove. The "simplest $g(n)$ possible" in the question means *basic efficiency class*. Please note that you are required to prove the efficiency classes you detemrined for the functions.

2. You may first find the Θ class for each formula and then use the order of the basic efficiency classes to compare them. If necessary, you can use the method of limits and L'Hopital's rule.

3. Use the formulas and rules on page 476 in the textbook.

4. Simplify the expressions to be summed and use the formulas and rules on page 476 in the textbook.

5. Denote the number of divisions as $D(n)$, the number of multiplications as $M(n)$, and the number of additions/subtractions as $A(n) + S(n)$. The number of operations can be directly found from the formulas. For example, the number of additions to calculate $\sum_{i=1}^{n} x_i$ is $n - 1$. You may use a small n, for example $n = 5$, to find the numbers of divisions, multiplications, and additions and subtractions.

6. See the following for subquestions.

   1. You may use some sample data to run the algorithm (with pen and paper) to see what it does.

   2. You may first identify the input, and then its size.

   3. It should be the operation executed in every iteration in the inner loop.

   4. Set up a sum for the algorithm and simplify it to obtain a function of n.

   5. Find the Θ for the function of n.

7. Use the method of backward substitution to solve the recurrences.

8. See the following for subquestions.

    1. You may run the algorithm with a small n, if the algorithm is not clear to you.
    2. The input size is what the computational cost is proportional to.
    3. Find the most expensive operation that is executed in every invocation of the algorithm.
    4. Use the method of backward substitution to solve the recurrence, to obtain a function of n.
    5. Find the Θ for the function of n.

9. See the following for subquestions.

    1. In **a.** set up a recurrence for computing Q. You can substitute a sequence of small n values, e.g. 1, 2, 3, 4, ... into your recurrence to find what the algorithm computes.
    2. In **b.** set up a recurrence for the number of times multiplications are executed.
    3. In **c.** set up a recurrence for the number of times additions/subtractions are executed.

10. See the following for subquestions.

    1. If not clear, apply the algorithm to a small **sorted** array of integers and a K.
    2. Write a small array with indexes l and r to find out.
    3. The recurrence is in the form of $x(n) = ax(n/b) + f(n)$. Use the second method of backward substitution to solve the recurrence.
    4. We studied a rule that may extend the analysis results for $n = 2^k$ to any n values, for some functions and basic efficiency classes.