

# CS5050 ADVANCED ALGORITHMS

Fall Semester, 2015

## Assignment 1: Algorithm Analysis

**Due Date: 8:30 a.m.**, Friday, Sept. 18, 2015 (at the beginning of CS5050 class)

**Note:** If not specified, the base of log is 2. This applies to all assignments of this course.

1. This exercise is to convince you that exponential time algorithms should be avoided.

Suppose we have an algorithm  $A$  whose running time is  $O(2^n)$ . For simplicity, we assume the algorithm  $A$  needs  $2^n$  instructions to finish for any input size of  $n$  (e.g., if  $n = 5$ ,  $A$  will finish after  $2^5 = 32$  instructions).

According to Wikipedia, the fastest supercomputer in the world in 2015 is Tianhe-2 (which is located in Guangzhou, China) can perform about  $3.4 \times 10^{16}$  instructions per second.

Suppose we run the algorithm  $A$  on Tianhe-2. Answer the following questions. **(10 points)**

- (a) For the input size  $n = 100$  (which is a relative small input), how much time does Tianhe-2 need to finish the algorithm? Give the time in terms of **centuries**.
- (b) For the input size  $n = 1000$ , how much time does Tianhe-2 need to finish the algorithm? Give the time in terms of **centuries**.

2. This exercise is to give you some feeling on the growth of functions when the input size grows.

Suppose you have algorithms with the five running times given below. (Assume these are the exact running times.) How much slower does each of these algorithms get when you double the input size (i.e.,  $n$  becomes  $2n$ )? (For example, consider an algorithm with running time  $3n$ ; when the input size doubles, the algorithm becomes  $\frac{3 \cdot (2n)}{3n} = 2$  times slower than before.)

- (a)  $n^2$       (b)  $n^3$       (c)  $100n^2$       (d)  $n \log n$       (e)  $2^n$  **(20 points)**

3. For each of the following pairs of functions, indicate whether it is one of the three cases:  $f(n) = O(g(n))$ ,  $f(n) = \Omega(g(n))$ , or  $f(n) = \Theta(g(n))$ . For each pair, you only need to give your answer and the proof is not required. **(30 points)**

- (a)  $f(n) = 100n + \log n$  and  $g(n) = 6n + \log^2 n$ .
- (b)  $f(n) = 20 \log n + 4$  and  $g(n) = \log n^2 - 100$ .
- (c)  $f(n) = \frac{n^2}{\log n}$  and  $g(n) = n \log^2 n$ .
- (d)  $f(n) = \sqrt{n}$  and  $g(n) = \log^5 n$ .
- (e)  $f(n) = n2^n$  and  $g(n) = 3^n$ .
- (f)  $f(n) = 4n \log n$  and  $g(n) = n \log_3 n$

4. This is a “warm-up” exercise on algorithm **design** and **analysis**.

The *knapsack problem* is defined as follows: Given as input a knapsack of size  $K$  and  $n$  items whose sizes are  $k_1, k_2, \dots, k_n$ , where  $K$  and  $k_1, k_2, \dots, k_n$  are all positive real numbers, find a full “packing” of the knapsack (i.e., choose a subset of the  $n$  items such that the total sum of the sizes of the items in the chosen subset is *exactly*  $K$ ).

It is well known that the knapsack problem is NP-complete, which implies that it is very likely that efficient algorithms (i.e., those with a polynomial running time) for this problem do not exist. Thus, people tend to look for good **approximation algorithms** for solving this problem. In this exercise, we relax the constraint of the knapsack problem as follows. We still seek a packing of the knapsack, but we need not look for a “full” packing of the knapsack; instead, we look for a packing of the knapsack (i.e., a subset of the  $n$  input items) such that the total sum of the sizes of the items in the chosen subset is *at least*  $K/2$  (but no more than  $K$ ). This is called a *factor of 2 approximation solution* for the knapsack problem. To simplify the problem, we assume that a factor of 2 approximation solution for the knapsack problem always exists, i.e., there always exists a subset of items whose total size is at least  $K/2$  and at most  $K$ .

Design a polynomial time algorithm for computing a factor of 2 approximation solution, and analyze the running time of your algorithm (in the big- $O$  notation). **(20 points)**

**Note:** You are required to clearly describe the main idea of your algorithm. Although the pseudo-code is not required, you may also give the pseudo-code if you feel it is helpful for you to explain your algorithm. (The reason I want to see the algorithm description instead of only the code or pseudo-code is that it would be difficult to understand another person’s code without any explanation.) You also need to briefly explain why your algorithm works, i.e., why your algorithm can produce a factor of 2 approximation solution. Finally, please analyze the running time of your algorithm.

**Total Points:** 80