CS5050 Advanced Algorithms

Fall Semester, 2019

Assignment 1: Algorithm Analysis

Due Date: 8:30 a.m., Monday, Sept. 16, 2019 (**at the beginning of CS5050 class**)

Note: To turn in your homework, you may either submit it electronically on Canvas, or write your solutions on papers and bring it to me in class before/on the due day. Note that the Canvas online submission will be **automatically closed** at 8:30 a.m. on the due day, so please make your submission on time.

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

1. **(10 points)** 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 that 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, as of November 2018, the fastest supercomputer in the world is "Summit" (developed by IBM for use at Oak Ridge National Laboratory) and can perform about 2.0×10^{17} instructions per second.

   Suppose we run the algorithm A on Summit. Answer the following questions.

   (a) For the input size $n = 100$ (which is a relative small input size), how much time does Summit need to finish the algorithm? Give the time in terms of **centuries** (you only need to give an approximate answer).

   (b) For the input size $n = 1000$, how much time does Summit need to finish the algorithm? Give the time in terms of **centuries** (you only need to give an approximate answer).

   Note: You may assume that a year always has exactly 365 days.

2. **(20 points)** Order the following list of functions in asymptotically increasing order (i.e., from small to large).

$$\begin{array}{cccccc} \log n^2 & n! & 2^{1000} & 2^n & \log(\log n)^2 & 2^{\log n} \\ \log^3 n & n \log n & \log_4 n & n^{2.5} & \sqrt{n} & n^2 \log^5 n \end{array}$$

3. **(30 points)** 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.

   (a) $f(n) = 8 \log n$ and $g(n) = \log n^3 - 30$.

   (b) $f(n) = 8n^2 + n \log^8 n$ and $g(n) = n^3 + \log^2 n$.

   (c) $f(n) = 5^n$ and $g(n) = n^7 \cdot 2^n$.

(d) $f(n) = n \log^2 n$ and $g(n) = \frac{n^2}{\log^3 n}$.

(e) $f(n) = \sqrt{n} \log n$ and $g(n) = \log^8 n + 25$.

(f) $f(n) = n \log n + 6n$ and $g(n) = n \log_5 n - 8n$

4. **(20 points)** This is a "warm-up" exercise on algorithm **design** and **analysis**.

We study a *knapsack problem*, defined as follows: Given as input a knapsack of size K and n items whose sizes are $k_1, k_2, \ldots, k_n$, where K and $k_1, k_2, \ldots, k_n$ are all **positive real** numbers, the problem is to 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* equal to 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 polynomial runtime) 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 *approximate solution* for the knapsack problem. To simplify the problem, we assume that a factor of 2 approximate 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.

For example, if the sizes of the n items are $\{9, 24, 14, 5, 8, 17\}$ and $K = 20$, then $\{9, 5\}$ is a factor of 2 approximate solution. Note that such a solution may not be unique. For example, $\{9, 8\}$ is also a solution. Also note that the item sizes may not be integers (but are positive real numbers), although they are integers in this example.

Design a **polynomial time** algorithm for computing a factor of 2 approximate solution, and analyze the running time of your algorithm (using the big-O notation). Note that although there may be multiple solutions, your algorithm only needs to find one solution.

If your algorithm runs in $O(n)$ time and is correct, then you will get **5 bonus points**.

Note: I would like to emphasize the following, which applies to the algorithm design questions in all assignments in this semester.

1. **Algorithm Description** You are required to clearly describe the main idea of your algorithm.
2. **Pseudocode** The pseudocode is usually very helpful for explaining the algorithm. So you are strongly encouraged to provide pseudocode for your algorithm.
3. **Correctness** You also need to explain why your algorithm is correct. For instance, for this knapsack problem you need to explain why your algorithm can produce a factor of 2 approximate solution.
4. **Time Analysis** You are required to analyze the running time of your algorithm.

Total Points: 80 (not including the 5 bonus points)