CS 5084 - Algorithms - Design and Analysis

Assignment 2 - Basics of Algorithm Analysis

For full credit, please adhere to the following:

- Unsupported answers receive no credit.
- All answers can be typed or handwritten, and should be readable.
- Submit the assignment in one pdf file

1. (30 points) Suppose you have the algorithms with the five running times listed below. (Assume these are the exact running times.)

   (i) n^2

   (ii) n^3

   (iii) $100n^2$

   (iv) $n\log n$

   (v) 2^n

   How much slower do each of these algorithms get when you do the following:

   (a) (15 points) Double the input size.

   (b) (15 points) Increase the input size by 1.

2. (20 points) Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

   (i) $f_1(n) = n^{2.5}$

   (ii) $f_2(n) = \sqrt{2n}$

   (iii) $f_3(n) = n + 10$

   (iv) $f_4(n) = 10^n$

   (v) $f_5(n) = 100^n$

   (vi) $f_6(n) = n^2\log n$

3. (30 points) Consider the following basic problem. You're givne an array A consisting of n integers $A[1], A[2], \ldots, A[n]$. You'd like to output a two-dimensional n-by-n array B in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$ - that is, the sum $A[i] + A[i + 1] + \cdots + A[j]$. (The value of array entry $B[i, j$ is left unspecified whenever $i < j$, so it doesn't matter what is output for these values.)

Here's a simple algorithm to solve this problem:

```
For i = 1, 2, ..., n
  For j = i + 1, i + 2, ..., n
    Add up array entries A[i] through A[j]
    Store the result in B[i, j]
  Endfor
Endfor
```

(a) (10 points) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

(b) (10 points) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

(c) (10 points) Although the algorithm you analyzedin parts (a) and (b) is s the most natural way to solve the problem - after all, it just iterates through the relevant entries of the array B, filling in a value for each - it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$$

4. (20 points) You're doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows: You have a ladder with n rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the *highest safe rung*.

   It might be natural to try binary search: drop a jar from the middle rung, see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. But this has the drawback that you could break a lot of jars in finding the answer.

   If you primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first run, then the second rung, and so forth, climbing one higher each time until the jar breaks. In this way, you only need a single jar - at the momen it brea,s, you have the correct answer - but you may have to drop it n times (rather than log$n$ as in the binary search solution).

   So here is the trade-off: it seems you can perform fewer drops if you're willing to break more jars. To understand better how this trade-off works at a quantitative level, let's consider how to run this experiment given a fixed "budget" of $k \geqslant 1$ jars. In other words,

you have to determine the correct answer – the highest rung – and can use at most k jars in doing so.

(a) (10 points) Suppose you are given a budget of $k = 2$ jars. Describe a strategy for finding the highest safe rung that requires you to drop a jar at most $f(n)$ times, for some function $f(n)$ that grows slower than linearly. (In other words, it should be the case that $\lim_{n\to\infty} f(n)/n = 0$.)

(b) (10 points) Now suppose you have a budget of $k > 2$ jars, for some given k. Describe a strategy for finding the highest safe rung using at most k jars. If $f_k(n)$ denotes the number of times you need to drop a jar according to your strategy, then the functions $f_1, f_2, f_3, \ldots$ should have the property that each grows asymptotically slower than the pervious one: $\lim_{n\to\infty} f_k(n)/f_{k-1}(n) = 0$ for each k.