CS3510-A: Design and Analysis of Algorithms, Spring 2021

Homework-1

January 26, 2021

DUE DATE: Tuesday, January 26, 11:59pm

Note-1: Your homework solutions should be electronically formatted as a single PDF document that you will upload on Gradescope. If you have to include some handwritten parts, please make sure that they are very clearly written and that you include them as high resolution images.
Note-2: Please think twice before you copy a solution from another student or resource (book, web site, etc). It is not worth the risk and embarrassment.
Note-3: You need to **explain/justify** your answers. Do not expect full credit if you just state the correct answer.
Note-4: You will get 2 extra points if you submit electronically typed solutions instead of hand-written.

Problem-1 (35 points)

Take the following list of functions and arrange them in ascending order of growth rate. That is, if the function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.
$g_1(n) = 2^{\sqrt{log(n)}}$
$g_2(n) = 2^n$
$g_3(n) = n(log(n))^3$
$g_4(n) = n^{4/3}$
$g_5(n) = n^{log(n)}$
$g_6(n) = 2^{2^n}$
$g_7(n) = 2^{n^2}$

You can represent the ordering of the functions with inequalities. For example, if your answer is $g_1 < g_2 < g_3 < g_4 < g_5 < g_6 < g_7$, then you will need to explain each of the six inequalities separately.

Problem-1 Response:

$g_1 < g_4 < g_3 < g_5 < g_2 < g_6 < g_7$

$g_1(n) <= c * g_4(n)$
for $n >= n_0$ where $c = 2, n_0 = 1$

$g_4(n) <= c * g_3(n)$
for $n >= n_0$ where $c = 3, n_0 = 8$

$g_3(n) <= c * g_5(n)$
for $n >= n_0$ where $c = 2, n_0 = 0$

$g_5(n) <= c * g_2(n)$
for $n >= n_0$ where $c = 1, n_0 = 1$

$g_2(n) <= c * g_6(n)$
for $n >= n_0$ where $c = 1, n_0 = 0$

$g_6(n) <= c * g_7(n)$
for $n >= n_0$ where $c = 2, n_0 = 1$

Problem-2 (35 points)

You're given an array A consisting of n integers $A[1], A[2], ..., A[n]$. 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, $\sum_{k=i}^{j} A[k]$.

The value of the array entry B[i,j] is left unspecified whenever $i \geq j$, 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) 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) For the same function f, show that the function 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) Although the algorithm you analyzed in parts (a) and (b) is 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 $\lim_{n \to +\infty} \frac{g(n)}{f(n)} = 0$.

Problem-2 Response:
a) Upper Bound
$f(n) = n^3$

b) Lower Bound
The given sudo code will have to iterate over $n^2/2$ cells to build the output matrix, and then iterate over at most n values in the input array to determine the new cells value. The resulting formula: $n^2/2 * n$ simplifies to: n^3, meaning $f(n) = n^3$ for $\Omega(f(n))$.

c) Improved sudo code: (This will has a running time of: $O(n^2)$)

```
for i = 1, 2, ... n
  for j = i + 0, i + 1, i + 2, ..., n
     if i == j:
        Store A[i] into B[i, j]
     else:
        value = B[i, j-1] + A[j]
        store value in B[i, j]
  endfor
endfor
```

Problem-3 (30 points)

Prove that, if c is a positive real number, then the geometric series $g(n) = 1 + c + c^2 + \cdots + c^n$ is:
a) $\Theta(1)$ if $c < 1$
b) $\Theta(n)$ if $c = 1$
c) $\Theta(c^n)$ if $c > 1$.

Problem-3 Response:

a) $\Theta(1)$ if $c < 1$

With $0 < c < 1$, then as n approaches infinity $g(n)$ will approach a limit. This limit can be bounded above and below by some scalar multiple of $f(n) = 1$.

b) $\Theta(n)$ if $c = 1$

With $c = 1$, then $g(n) = n + 1$. This can always be bounded below by $f(n) = n$, and can be bounded above by a scalar of the same bounding function (for example: $f(n) = 2n$) for any point $n >= 1$.

c) $\Theta(c^n)$ if $c > 1$.

With $c > 1$, then g(n) will return some scaled value of c^n (For $c = 2$, then $g(n) = 2 * c^n - 1$. For $c = 3$, then $g(n) = (3/2) * c^n - (1/2)$.).

Therefore it can be bounded above and below by a greater and lower multiple of $f(n) = c^n$.