CS 413: Analysis of Algorithms
Homework 1

Instructor: Zahra Derakhshandeh
Due at the *beginning of class*
(11:00 am) on Monday, September 10, 2018, for 20189_CS_413_04_1
(2:00 pm) on Monday, September 10, 2018, for 20189_CS_413_03_1

This is an *individual* homework assignment. All solutions has to be **typed**. No credit will be received for hand written solutions (except for figures or long equations which might be handwritten). Along with a hard copy (submitted in class) You also need to submit a copy of your assignment in the digital repository **before the beginning of your class on Monday, September 10, 2018**.

As it appears in the course syllabus, "for the homework assignments students are *encouraged to discuss* the problems with others, but one is expected to turn in the results of one's own effort (not the results of a friend's efforts)". Even when not explicitly asked you are supposed to concisely justify your answers.

1. Suppose you have algorithms with the five running times listed below.(Assume these are the exact running times.) How much slower do each of these algorithms get when you (a) double the input size, or (b) increase the input size by one?
   (i) n^2
   (ii) n^3
   (iii) $100n^2$
   (iv) $nlog(n)$
   (v) 2^n

2. 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)).

$$f_1(n) = n^{2.5}$$
$$f_2(n) = \sqrt{2n}$$
$$f_3(n) = n + 10$$
$$f_4(n) = 10^n$$
$$f_5(n) = 100^n$$
$$f_6(n) = n^2 \log n$$

3. Assume you have functions f and g such that f(n) is O(g(n)). For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

(a) $\log_2 f(n)$ is $O(\log_2 g(n))$.

(b) $2^{f(n)}$ is $O(2^{g(n)})$.

(c) $f(n)^2$ is $O(g(n)^2)$.

4. Express the time complexity of the quick sort as a recurrence relation. Explain why you obtain this relation. Solve the recurrence relation and provide the worst-case running time of quick sort accordingly.

5. Consider the following basic problem. You're given 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 $A[i]+A[i+1]+\ldots+A[j]$. (The value of 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 do
    for j = i + 1, i + 2, ··· , n do
        Add up array entries A[i] through A[j]
        Store the result in B[i, j]
    end
end
```

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