Homework Assignment #1

Due: Oct 10th, 2017

CMPUT 204
Department of Computing Science
University of Alberta

Note: All logarithms are in base 2 unless specified otherwise. We denote $\mathbb{R}^+ = \{x \in \mathbb{R} : \ x > 0\}$ and $\mathbb{R}_{\geq 1} = \{x \in \mathbb{R} : x \geq 1\}$.
This assignment is partitioned into Exercises and Problems. Exercises are optional and will not be graded. Problems are mandatory and will be graded. You are, however, strongly advised to work out a full solutions for both.
Your max-grade for this assignment is 110 (+2 pts bonus). However, you **must** submit answers to Problems 1 & 2.

Exercise I. In the binary-search problem we take as input a *sorted* array A with n elements, and a key k and we return the largest element in A that which isn't strictly greater than k. That is we return

$$\max\{x \in A : \ x \leq k\} \text{ , or } \perp \text{ if all } x \in A \text{ satisfy } x > k$$

Write a pseudo-code for binary search that (a) uses recursion and (b) runs in $O(\log(n))$ time.

Exercise II. Prove that for any $n \geq 1$ and $0 \leq k \leq n$ we have that $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$.
Use this fact to give a recursive algorithm that computes $\binom{n}{k}$. Prove the correctness of your algorithm.

Exercise III. True or False? Prove.

1. $\sqrt{n} \in O(n)$.
2. $\sqrt{2^n} \in O(2^{\sqrt{n}})$
3. $n! \in O(2^{n \log(n)})$
4. $n! \in \Theta(2^{n \log(n)})$
5. For any function $f : \mathbb{N} \to \mathbb{R}^+$ it holds that $o(f) \cap \omega(f) = \emptyset$.
6. For any function $f : \mathbb{N} \to \mathbb{R}^+$ it holds that $\Omega(f) = \omega(f) \cup \Theta(f)$.

Exercise IV. Pick your favorite programming language.
For each of the following values of n — from 1000 to $50,000$ in increments of 1000 — do the following:

1. Generate an array A large enough to hold n integers.
2. Start the clock.
3. Repeat n times:
   (a) Generate an arbitrary integer.
   (b) Insert the integer into A.
4. Stop the clock and compute the runtime.

Now draw the plot of the runtime as a function of n.

However, you should repeat this experiment twice: once where the insertion of Step 3(b) is done using the standard `Insert`, and once where the insertion is done using `Insert2` (see slides of Unit 01).

Draw both plots on the same figure. Explain the two plots.

Problem 1. (10 pts) True or False? Provide a *short* explanation.

(i) $n^5(\log(n))^{29} + 52\sqrt{n^9} \in O(n^{\sqrt{529}})$ (Note: $\sqrt{529} = 23$)

(ii) $n^{52}(\log(n))^9 + 52\sqrt[9]{n} \in O(52^9 n^{\frac{52}{9}})$

(iii) $52^{n^9} \in O(92^{n^5})$

(iv) $52^{9n} \in \Theta((52 \cdot 9)^n)$.

(v) $1.5^n + 1.29^n \in O(1.529^n)$.

(vi) $\sqrt{1.5^n + 1.29^n} \in O(1.529^{\sqrt{n}})$.

(vii) $n^{\frac{529 \log\log n}{\log n}} \in O(n^{0.529})$.

(viii) There exists a constant $c > 1$ such that $n^c \in \Theta(n(\log(n))^{529})$.

(ix) $1^{529} + 2^{529} + \ldots + n^{529} \in \Theta(n^{530})$.

(x) $\sum_{i=1}^{n} \sum_{j=i+1}^{n} (j-i)^{529} \in O(n^{531})$

Problem 2. (10 pts) Prove the following claims. Make sure you use statements such as "Let x be *any*..." or "We prove the statement by constructing the following example."

(i) There exists two functions $f, g : \mathbb{N} \to \mathbb{R}^+$, such that $f \notin O(g)$ and $g \notin O(f)$.

(ii) (Geometric series) Given a constant c, denote $f_c(n) = c^0 + c^1 + c^2 + ... + c^n$.

Then $f_c(n) \in \begin{cases} \Theta(c^n), & \text{if } c > 1 \\ \Theta(n), & \text{if } c = 1 \\ O(1), & \text{if } 0 < c < 1 \end{cases}$

(iii) For any two functions $f, g : \mathbb{N} \to \mathbb{R}^+$, denoting $\max\{f, g\}(n) = \max\{f(n), g(n)\}$, then we have that $\max\{f, g\}(n) \in \Theta(f(n) + g(n))$.

(iv) For any functions $f, g : \mathbb{N} \to \mathbb{R}^+$, $\sqrt{f(n)} + \sqrt{g(n)} \in \Theta(\sqrt{f(n) + g(n)})$.

(v) There exists two functions $f, g : \mathbb{N} \to \mathbb{R}^+$ such that $f(n) \in \Theta(g(n))$ yet $2^{f(n)} \notin \Theta(2^{g(n)})$;
(2pt Bonus) and there exists two functions $f, g : \mathbb{N} \to \mathbb{R}^+$ such that $2^{f(n)} \in \Theta(2^{g(n)})$ yet $f(n) \notin \Theta(g(n))$.

Problem 3. (20 pts) Order the following list of functions by increasing big-O growth rate. Group together (for example, by underlining) those functions that are big-Θ of one another.

$529n$	5^{2^9}	$\log((\log n)^{529})$	$(\log(n))^{529}$	$2^{\log(n+529)}$
$\log(2^{2^{\cdot^{\cdot^{\cdot^2}}}\}n \text{ times}})$	$2^{\frac{\log(n)}{529}}$	$2^{(\log(n)+529)}$	$2^n + 2^{n-1} + 2^{n-2} + \ldots + 2^0$	4^n
$529n^{0.529} + 2^{\sqrt{\log(n)}}$	$2^{\log^*(n)}$	$n^2 \log_{529}(n) + 2^{529}n^2$	$n \log(n^{529})$	$n \log(529n)$
$529n^{0.529} + 925n^{9.25}$	$n^3 - n^{2.99}(\log(n))^{529}$	$\lfloor 529n(\log(n))^2 \rfloor$	$4^{\log n}$	$\log_{529}(n)$

Problem 4. (15 pts) We define a series using the following rule: $a_0 = 0$, $a_1 = 1$ and $a_n = 2a_{n-1} + a_{n-2}$ for any $n \geq 2$.

(i) (3 pts) Denote $\phi = 1 + \sqrt{2}$. Let $f(x) = x^2 - 2x - 1$. Show that $f(\phi) = 0$, and that $f'(x) > 0$ for any $x > 1$. Infer that for any c, d such that $1 < c < \phi < d$ we have that $2c + 1 > c^2$ and $2d + 1 < d^2$.

(ii) (7 pts) Prove that for any two constants c, d such that $1 < c < \phi < d$ we have that $a_n \in O(d^n)$ and that $a_n \in \Omega(c^n)$.

(iii) (5 pts) Prove that for any c, d such that $1 < c < \phi < d$ we have that $a_n \in o(d^n)$ and that $a_n \in \omega(c^n)$. (Hint: use (ii))

Problem 5. (15 pts)
(i) (7 pts) Give a *recursive* pseudo-code for an algorithm solving the following problem. Input: a linked-list L with integers; Output: (a pointer to) the sub-linked-list whose *head* holds the largest element in L or `nil` if L is empty.
Example: suppose L has 6 elements: $\boxed{1} \to \boxed{9} \to \boxed{6} \to \boxed{44} \to \boxed{3} \to \boxed{15}$, then the output should be the sub-list whose head is 44: $\boxed{44} \to \boxed{3} \to \boxed{15}$.
State a formal claim, and provide a formal proof, as to the correctness of your pseudo-code.
(ii) (8 pts) Give a pseudo-code that executes `SelectionSort` on a list L of integers (repeatedly finds the largest element in L and puts it at the end of the unsorted elements). Claim (and prove) formally the correctness of your algorithm.
Here is an example of the code for `SelectionSort` on an *array*:

```
procedure SelectionSort(A, 1, n)
for (j from n downto 1) do
    ind ←FindMax(A, 1, j)        ** finds the index of the largest element in A[1, ...j]
    exchange A[ind] ↔ A[j]
```

You should write a similar code for a *linked-list*.
Hint: It is probably easiest to create a new list $L2$, find and *remove* the largest element in L and insert this element into the right place in $L2$. You may refer to the code for `InsertHead()` and `DeleteHead` from the slides without re-writing them.

Problem 6. (20 pts) Consider the following program:

```
procedure Foo(A, f, l)
**Precondition: A[f ... l] is an array of integers, f, l are two naturals ≥ 1 with f ≤ l.
if (f = l) then
    return (A[f])^2              ** i.e. A[f] × A[f]
else
    m ← ⌊(f+l)/2⌋
    return Foo(A, f, m)+ Foo(A, m + 1, l)
end if
```

(i) (2 pts) In a sentence: what does `Foo` return? State a formal claim.

(ii) (4 pts) Prove that indeed `Foo` returns what you claim it returns. (You may assume all elements in A are unique.)

(iii) (4 pts) Using induction, argue that `Foo` invokes the ^{2}-operator at most n times.

(iv) (3 pts) Write a non-recursive version of `Foo`(A, f, l) (in pseudo-code).

(v) (4 pts) Claim and prove formally the correctness of your algorithm.

(vi) (3 pts) Find the worst case running time of this new program and express it in Θ notation.

Problem 7. (20 pts) Professor Harry Potter has devised an ingenious concoction: it looks like water, tastes like water and resembles water in any way, shape or form, and yet consuming even a single drop of this potion will cause the drinker's skin to turn bright blue for a day. However there's a delay in the effect of the potion — it takes about 45 minutes for the effect to kick in.

Alas, Prof. Potter's students, showing him the same respect undergraduates show to their professors (sigh), have scraped off the bottle's label and hid it among $n-1$ water bottles. So now Prof. Potter's cabinet holds n identically looking bottles, where one of them is the turn-skin-to-blue potion and the rest contain very ordinary water. To make matters worse, Prof. Potter has no more than 1 hour to find the potion before potion-class begins. Luckily, he can still call on a *few* of his trusted grad-students (and on you!) to assist him in finding the potion.

Help Prof. Potter! Design an algorithm that finds which one of n bottles contains the special potion in a single hour, using no more than $\lceil \log(n) \rceil$ tasters. Prove your claims.

Hint: Start by thinking recursively. The cases of $n = 1$ or $n = 2$ are easy. Assuming you have a "tasting scheme" for n bottles with k drinkers, devise a potion-finding tasting scheme for $2n$ bottles using $k + 1$ tasters. Then "unravel" the recursion to see what bottles the j-th taster drinks from. It is OK if you think of n as a power of 2 for the purpose of finding the algorithm, but your algorithm should work for any n. (Also, it is probably a tad more convenient to number the bottles from 0 to $n - 1$ rather than from 1 to n.)