CMPSC 465	Data Structures & Algorithms	
Spring 2021	Paul Medvedev and Chunhao Wang	HW 1

Due January 28, 10:00 pm

Instructions: You may work in groups of up to three people to solve the homework. You must write your own solutions and explicitly acknowledge up everyone whom you have worked with or who has given you any significant ideas about the HW solutions. You may also use books or online resources to help solve homework problems. All consulted references must be acknowledged.

You are encouraged to solve the problem sets on your own using only the textbook and lecture notes as a reference. This will give you the best chance of doing well on the exams. Relying too much on the help of group members or on online resources will hinder your performance on the exams.

Late HWs will be accepted until 11:59pm with a 20% penalty. HWs not submitted by 11:59pm will receive 0. There will be no exceptions to this policy, as we post the solutions soon after the deadline. However, you will be able to drop the three lowest HW grades.

For the full policy on HW assignments, please consult the syllabus.

1. **(0 pts.)** **Acknowledgements.** The assignment will receive a 0 if this question is not answered.

   (a) If you worked in a group, list the members of the group. Otherwise, write "I did not work in a group."

   (b) If you received significant ideas about the HW solutions from anyone not in your group, list their names here. Otherwise, write "I did not consult without anyone my group members".

   (c) List any resources besides the course material that you consulted in order to solve the material. If you did not consult anything, write "I did not consult any non-class materials."

2. **(20 pts.)** **Tribonaci numbers**
   Your manager proposed to generate IDs for customers based on what he calls Tribonaci numbers. They define Tribonaci numbers using the recurrence $R(i) = R(i-1) + R(i-2) + R(i-3)$, with the first three Tribonaci numbers defined as $R(0) = 1$, $R(1) = 2$, $R(2) = 3$. They would like to assign the ith customer an ID of $R(i)$. You tell them that they are crazy and that the IDs are going to be huge; in fact, you claim that the ith customer will have an ID number of at least $3^{i/2}$. Your manager, who has never taken lightly to criticism, threatens to fire you on the spot unless you can come up with a proof of your claim.

   (a) To save your job, use induction to prove that in fact $R(i) \geq 3^{i/2}$ for all $i \geq 2$.

   (b) Your manager is perplexed. They didn't take CMPSC360 or CMPSC465 so they don't know what induction is and don't understand your proof. They propose a simple recursive algorithm to compute $R(i)$. The algorithm simply makes three recursive calls and then adds the results. You tell him that his algorithm will take exponential time, but your manager doesn't want to hear anything more about induction. They demand an intuitive picture instead of a proof. Luckily, you just covered something like this in class. Draw a recursion tree that shows the cascade of recursive invocations triggered by a single call to compute $R(i)$. Two levels deep will suffice.

3. **(10 pts.)** **Big Oh definitions** Let f and g be functions from positive integers to positive reals. In class, we saw the following definition of big Oh: $O_1(g(n)) = \{f(n) : \exists \text{ constants } c_1 > 0, n_0 > 0 \text{ s. t. } f(n) \leq$

$c_1 g(n), \forall n \geq n_0\}$. The textbook gives an alternate definition: $O_2(g(n)) = \{f(n) : \exists$ a constant $c_2 > 0$ s. t. $f(n) \leq c_2 g(n)$ for all $n > 0\}$. Prove that the two definitions are equivalent. In other words, prove that $O_1(g(n)) = O_2(g(n))$ for all g.

4. **(20 pts.)** **Analyze Running Time.** For each pseudo-code below, give the asymptotic running time in Θ notation. (You may assume that standard arithmetic operations take $\Theta(1)$ time.) You may assume that n is a power of two if it simplifies your analysis. Briefly justify your answer.

1.
```
for i := 1 to n do
    j := i;
    while j < n do
        j := j + 5;
    end
end
```

2.
```
for i := 1 to n do
    for j := 4i to n do
        s := s + 2;
    end
end
```

3.
```
s := 0;
i := n;
while i ≥ 1 do
    i := i div 2;
    for j := 1 to i do
        s := s + 1
    end
end
```

4.
```
for i := 1 to n do
    j := i²;
    while j ≤ n do
        j := j + 1
    end
end
```

5. **(10 pts.)** **Tighter Analysis.** We are going to analyze the running time of an n-bit counter that counts from 0 to $2^n - 1$. For example, when $n = 5$, the counter has the following values:

Step	Value	# Bit-Flips
0	00000	–
1	00001	1
2	00010	2
3	00011	1
4	00100	3
⋮	⋮	
30	11110	2
31	11111	1

Note that we only need to flip the last bit to go from 0 to 1, but we need to flip the last two bits to go from 1 to 2. It is easy to see that to go from 0 to $2^n - 1$, it takes at most $O(n2^n)$ bit-flips. Show that, in fact, it takes $\Theta(2^n)$ bit flips.

(*Hint:* How many times is the last bit flipped? How many times is the second last bit flipped?)