CSCI 1311: Problem Set 3

Due: 2 Mar. 2020

Instructions:

- Your submission must be **typed** and submitted to gradescope as a single pdf.
- You must include a cover page, that contains your name, the assignment information, the date, and your GW email address. No answers to questions should appear on the cover page.
- Try and organize your submission such that answers to questions (or parts of questions) do not span multiple pages. This will make it much easier to grade. **Ideally, each page will start with a new question (or part of question). See the sample for PS0 for a nice easy formatting.**
- On gradescope, be sure to mark which page your answer to each question (or sub question) is located. Doing so inaccurately could lead to issues with grading.

Question Weighting

Question:	1	2	3	Total
Points:	90	10	15	115

---

1. For each recurrence, solve the following recurrence relation and provide a proof (using induction) that your solution describes the recurrence relation. That is, generate a formula for the recurrence in terms of n alone, for the n'th term of the sequence. Then prove the statement, "if a_n is described by the recurrence relation, then a_n equals the solution ..."

   Place the answer to each sub-part (a, b, c, ...) on a single page for grading

   (a) **[10 points]** $a_n = 5 + a_{n-1}$ where $a_0 = 9$

   (b) **[10 points]** $a_n = n + a_{n-1}$ where $a_0 = 10$

   (c) **[20 points]** $a_n = 2a_{n/2}$ where $a_1 = 1$. (*Hint: you'll need strong induction to prove this result*)

   For this question, you can assume that n is always a power of 2, that is $n = 2^x$ for some $x \geq 0$. However, if you want to better generalize the recurrence, you can define division by 2 as $\lceil n/2 \rceil$ which is well defined for all n and will always reach 1 after successive divisions.

   (d) **[25 points]** $a_n = 3a_{n-1} + 4a_{n-2}$ where $a_0 = 5$ and $a_1 = 15$

   (e) **[25 points]** $a_n = 8a_{n-1} - 16a_{n-2}$ where $a_0 = 2$ and $a_1 = 4$

2. **[10 points]** There is a close connection between recurrence relations and program analysis. For example, if we have the following recurssion function

```
def foo(int n):
   if n == 0: return 0          # 1 comparison + 1 return
   else: return 1 + foo(n-1)    # 1 addition + 1 function call + 1 return
                                # + num. operations in foo(n-1)
```

We can describe the "steps" or "work" of each function based on each operation, like a comparison, addition, or subtraction. However, since the total steps of operation depends on a recursive call, we can represent the calculation of the number of steps as a recurrence relation of the step function $T(n)$.

$$T(n) = \begin{cases} 3 + T(n-1) & \text{if } n > 0 \\ 2 & \text{if } n = 0 \end{cases}$$

How many steps will this function take in terms of n, that is solve the recurrence relation for $T(n)$.

3. Let's consider a slightly more complicated algorithm that you all may be aware of, *binary search*. This is the routine for looking up a word in a (physical) dictionary, or a number in a phone book (when phone books were a thing). In plain English and pseudo-code, the binary search routine could be described as:

1. Open to the middle of dictionary, look at the first word
2. If the word I'm looking for is less than the first word,
   - Search the **first half** of the dictionary in step (1), treating the half of the dictionary as if it was the whole dictionary.
3. If the word I'm looking for is greater than the first word,
   - Search the **second half** of the dictionary in step (1), treating the half of the dictionary as if it was the whole dictionary.
4. Continue until there is only one page left in the dictionary, then scan that page until you find the word.

```
//W is a word in the dictionary
//D is a subset of the dictionary pages
findWord(W,D){

  //base case
  if(D is only one page){
    scan the page for the word
    return if found
  }

  //recursive case
  w := first word on the middle page of D
  if( W comes before w){
    d := first half of D pages, inclusive
    return findWord(W, d)
  }else{
    d := second half of D pages, exclusive
    return findWord(W, d)
  }
}
```

(a) **[5 points]** Describe a recurrence relation $B(n)$ which counts the number of steps in the binary search routine of `findword()` in terms of $n = |D|$, the number of pages currently being searched. When counting steps, each of the following counts as 1 step:

- any comparison (e.g., `W comes before w`, or `D is only one page`)
- any assignment (e.g., the `:=` operator in pseudo-code)
- finding a word on a page (either the first word or later in the page)
- finding the middle page of a dictionary
- splitting a dictionary in two first-half or second-half of pages
- function calls (e.g, calling `findWord()` recursively)
- returning a result (e.g., all the `return` calls)

If it helps, you can always assume n is a power of 2 to simplify your analysis.

(b) **[10 points]** Solve your recurrence relation described in the previous part, that is describe the number of steps in terms of n only.