COMP 3270 Homework 1

100 points. Due by **11:59pm (midnight) on Thursday, September 8th, 2022**

Instructions:

1. This is an individual assignment. You should do your own work. Any evidence of copying will result in a zero grade and additional penalties/actions.
2. Submissions not handed on the due date and time **will not** be accepted unless prior permission has been granted or there is a valid and verifiable excuse.
3. Think carefully; formulate your answers, and then write them out concisely using English, logic, mathematics and pseudocode (no programming language syntax).
4. Type your final answers in this Word document and submit online through Canvas.
5. Don't turn in handwritten answers with scribbling, cross-outs, erasures, etc. If an answer is unreadable, it will earn zero points. **Neatly and cleanly handwritten submissions are also acceptable**.

1. (3 points) Bill has an algorithm, find2D, to find an element x in an n×n array A. The algorithm find2D iterates over the rows of A and calls the algorithm arrayFind (see below) on each one, until x is found or it has searched all rows of A. What is the worst-case running time of find2D in terms of n? Is this a linear-time algorithm? Why or why not?

Algorithm arrayFind(x, A):
  Input: An element x and an n-element array, A.
  Output: The index i such that $x = A[i]$ or -1 if no element of A is equal to x.
$i \leftarrow 0$
while $i < n$ **do**
  if $x = A[i]$ **then**
    return i
  else
    $i \leftarrow i + 1$
return -1

- The worst-case running time of find2D is $O(n^2)$ in n terms, because where the worst case where x is the final element in the array, the algorithm find2D calls arrayFind n total times. The algorithm HAS to go through all n elements until x is found, which arrayFind calls a total of n comparisons. The total operations are (n x n) or $O(n^2)$, due to the total number of calls in arrayFind is n number of times, proving that arrayFind is a not a linear-time algorithm, but a quadratic algorithm.

2. (4 points) Computational problem solving: Developing strategies: An array A contains n−1 unique integers in the range [0,n−1]; that is, there is one number from this range that is not in A. Describe a strategy (not an algorithm) for finding that number. You are allowed to use only a constant number of additional spaces beside the array A itself.

- First, find the sum from 1 – (n-1) for [(n(n-1))/2], then find the sum of values in array A. To find the specified number not in Array A, find the difference from the previously solved sums.

3. (3 points) *Computational problem solving: Developing strategies:* Given a string, S, of *n* digits in the range from 0 to 9, describe an efficient strategy for converting S into the integer it represents.

- Initialize a variable (variable = x), set it equal to 0. Given a string (S), and digit (N), variable x will be newly updated as 10x + N

4. (3 points) *Computational problem solving: Estimating problem solving time:* Suppose there are three algorithms to solve a problem- a O(n) algorithm (A1), a O(nlogn) algorithm (A2) and a $O(n^2)$ algorithm (A3) where log is to the base 2. Using the techniques and assumptions in slide set L2-Buffet(SelectionProblem).ppt, determine how long in seconds it will take for each algorithm to solve a problem of size 200 million. You must show your work to get credit, i.e., a correct answer without showing how it was arrived at will receive zero credit.

- machine speed i= 4GHz → (4 * 10 cycles/sec) or ($2 * 10^7$ steps/sec)

N = 200 million → ($2 * 10^8$)
O(n) = $2 * 10^8 / 2 * 10^7 = 10$ seconds
O(nlogn) = $((2 * 10^8) \log2(2 * 10^8)) / (2 * 10^7) => (5.5 * 10^9) / (2 * 10^7) = 275.75$ seconds
O(n^2) = $((2 * 10^8)^2) / (2 * 10^7) => (4 * 10^{16}) / (2 * 10^7) = 2 * 10^9$ seconds

5. (6 points) *Computational problem solving: Problem specification*
Suppose you are asked to develop a mobile application to provide **turn by turn** directions on a smartphone to an AU parking lot in which there are at least five empty parking spots nearest to a campus building that a user selects. Assume that you can use the Google Map API for two functions (only) – display campus map on the phone so user can select a campus building, and produce turn-by-turn directions from a source location to a destination location – where any location in the map is specified as a pair (latitude, longitude). Also assume that there is an application called AUparking that you can query to determine the # of vacant spots in any parking lot specified as a pair (latitude, longitude). Specify the problem to a level of detail that would allow you to develop solution strategies and corresponding algorithms: State the problem specification in terms of (1) inputs, (2) data representation and (3) desired outputs; no need to discuss solution strategies.

---

i. Inputs – Current location, Destination location
ii. Data Representation -
    a. Current location = (Cur[latitude, longitude]) == Input Value
    b. Destination location = (Des[latitude, location]) == Input Value

    Parking lot closest to destination that has at 5 empty spots = (Emp[latitude, longitude]) == Output Value

    Turn by turn directions to lot from Cur (current location) can be represented by an (n x n) matrix where n is the number of nodes from current location to the destination

iii. Desired Outputs – Parking lot location that is closest to the destination that has 5 empty spots at least, and set of turn by turn directions to that parking lot from current location

6. (5 points) *Computational problem solving: Developing strategies*
Explain a correct and efficient **strategy** to check what the maximum difference is between any pair of numbers in an array containing n numbers. Your description should be such that the strategy is clear, but at the same time, the description should be at the level of a strategy, not an algorithm. Then state the total number of number pairs any algorithm using the strategy "compute the difference between every number pair in the array and select that pair with the largest difference" will have to consider as a function of n.

- Have 2 variables, min and max which hold the smallest and largest numbers in the array, respectively. Set both min and max to the first number in the array. Then check each element in the array one-by-one, comparing them to min and max. If the number is lower than min, set min to the new number. If the number is larger than max, set max to the new number. Once you reach the end of the array, the maximum difference = max – min.

- Using the described algorithm, the 1st element will compare with the other n-1 elements after it. Then the 2nd will check the other n-2 following it, repeating all the way to the end. For example, in an array of n = 4: element 1 makes 3 comparisons, element 2 makes 2, element 3 makes 1, and element 4 makes none. 3 + 2 + 1 = 6, or 4 * (4-1) / 2 = 6. The generalized case will make n*(n-1)/2 comparisons.

7. (9 points) *Computational problem solving: Understanding an algorithm and its strategy*

Algorithm Mystery(A[1..n])
Input: An n-element array. Indexed from 1 to n

```
m ← 0     // the maximum found so far
for j ← 1 to n do
    for k ← j to n do
        s ← 0     // the next partial sum we are computing
        for i ← j to k do
            s ← s + A[i]
        if s > m then
            m ← s
return m
```

(a) Explain what the following algorithm outputs and simulate its operation on a valid input instance (e.g., an array of n elements - you can choose n to be 10)

(b) What is the approximate time complexity (running time) of the above algorithm (you can use Big-Oh notation)

(c) How does the following algorithm improve the time complexity of the algorithm (what is its strategy)? What is its time complexity?

```
S_0 ← 0     // the initial prefix sum
for i ← 1 to n do
    S_i ← S_{i-1} + A[i]
m ← 0     // the maximum found so far
for j ← 1 to n do
    for k ← j to n do
        s = S_k − S_{j-1}
        if s > m then
            m ← s
return m
```

(a) Algorithm outputs the partial sum of possible subarrays using the sum of values. For each of the sums, there are comparisons for a temporary max sum, and replaces the max if there arises a sum greater than the temporary max sum

(b) $O(n^3)$

(c) It considers the sums of the first t integers in Array A. When the prefix sums are solved, we can solve all subarrays in constant time complexity, iterating n times total. Time Complexity = $O(n^2)$

8. (9 points) *Computational problem solving: Calculating approximate complexity:*
Using the approach described in class (L5-Complexity.pptx), calculate the approximate complexity of Mystery algorithm above by filling in the table below.

Step	Big-Oh complexity
1	O(1)
2	O(1)
3	O(n)
4	O(n)
5	$O(n^2)$
6	$O(n^2)$
7	$O(n^2)$
8	$O(n^2)$
9	O(1)
Complexity of the algorithm	$O(n^2)$

---

9. (9 points) Calculate the detailed complexity T(n) of Mystery. Fill in the table below, then determine the expression for T(n) and simplify it to produce a polynomial in n.

Step	Cost of each execution	Total # of times executed
1	1	1
2	1	1
3	1	n+1
4	1	n
5	1	$\sum_{i=1}^{n} (n - i + 2)$
6	6	$\sum_{i=1}^{n} (n - i+1)$
7	3	$\sum_{i=1}^{n} (n - i+1)$
8	2	$\sum_{i=1}^{n} (n - i+1)$
9	2	1

$T(n) = 1 + 1 + n + 1 + n + \sum_{i=1}^{n} (n - i + 2) + 6\sum_{i=1}^{n} (n - i+1) + 3\sum_{i=1}^{n} (n - i+1) + 2\sum_{i=1}^{n} (n - i+1) + 1$

You need to carry out the math by evaluating the summation expressions to derive a closed form polynomial expression as a function of n.

---

10. (3 points) *Computational problem solving: Proving correctness/incorrectness:*
Is the algorithm below correct or incorrect? Prove it! It is supposed to count the number of all identical integers that appear consecutively in a file of integers. E.g., if f contains 1 2 3 3 3 4 3 5 6 6 7 8 8 8 8 then the correct answer is 9

```
Count(f: input file)
count, i, j : integer        //local variables
count=0
while end-of-file(f)=false
        i=read-next-integer(f)
        if end-of-file(f)=false then
                j=read-next-integer(f)
                if i=j then count=count+1
return count
```

- The algorithm is incorrect. If the given example is input, the output will be 3 which is not equal to 9. The given algorithm only checks each partition of 2, without checking the adjacent numbers between partitions.

11. (10 points) *Computational problem solving: Proving correctness:* Complete the proof by contradiction this algorithm to compute the Fibonacci numbers is correct.

```
function fib(n)
1. if n=0 then return(1)
2. if n=1 then return(1)
3. last=1
4. current=1
5. for i=2 to n do
6.      temp=last+current
7.      last=current
8.      current=temp
9. return(current)
```

1. Assume the algorithm is incorrect.
2. Fibonacci numbers are defined as $F_0=1$, $F_1=1$, $F_i=F_{i-1}+F_{i-2}$ for $i>1$.
3. So the assumption in (1) implies that there is at least one input parameter $n=k$, $k\geq 0$, for which the algorithm will produce an incorrect answer.
4. For n=0 and n=1, the algorithm returns 1, which matches the values for F_0 and F_1, so in both cases the algorithm returns the correct answer.
5. This implies that there has to be at least one integer $k>1$, so that when $n=k$ the algorithm does not return the correct answer $F_k=F_{k-1}+F_{k-2}$.
6. When $n=k$ and $k>1$ neither of the if statements are true leading the algorithm to continue, and steps 3-9 will be executed.

7. If k=2, the for loop in steps 5-8 will be executed exactly once. By step 6, temp = last + current = 1 + 1 = $F_0 + F_1$. Then step 7 updates last to be equal to current = F_1. Step 7 updates current to be equal to temp which is $F_0 + F_1$. So the value returned in step 9 is current = $F_0 + F_1 = F_2$. This is the correct answer. So the k for which the algorithm fails must be greater than 2.
8. If k=3, the for loop in steps 5-8 will be executed twice. In the first loop on step 6, temp = 1+1 = $F_0 + F_1$. Step 7 then updates last = current = F_1. Step 8 then updates current = temp = $F_0 + F_1 = F_2$. From here, the loop repeats once more starting again at step 6, where temp = last + current = $F_1 + F_2 = F_3$. Step 7 of this loop updates last = current = F_2. Step 8 then sets current = temp = F_3. The loop does not repeat again, returning current. Now current equals $F_3 = F_2 + F_1$, which is the correct input. Therefore, the k which the algorithm fails must be above 3.
9. But if k= 4, the for loop in steps 5-8 will execute thrice. For the first two loops, we end with last = F_2 and current = F_3 as seen from the previous case where k=3. The third loop begins with temp = last + current = $F_3 + F_2 = F_4$ on step 6. Step 7 sets last = current = F_3. Step 8 sets current = temp = F_4. The loop breaks and current = F_4 is returned, which is the correct answer. This means the k where the algorithm fails must be greater than 4.

10. The above argument can be repeated to show that before each following loop, current will be set to F_{i-1} and last will be set to F_{i-2}. These values are then added in the last loop, set to temp, then set to current, and finally is returned. This follows the pattern such that current = $F_i = F_{i-1}+F_{i-2}$.
11. That is, for all k > 1 the algorithm returns the correct k-th Fibonacci number.
12. So there is no k for which the algorithm will return a value not equal to $F_{k-1}+F_{k-2}$. This contradicts (3).
13. Therefore, the algorithm must be correct.

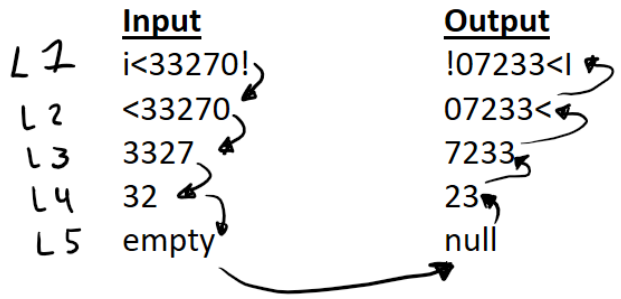
12. (a) (6 points) *Computational problem solving: Algorithm design:* Describe a recursive algorithm to reverse a string that uses the strategy of swapping the first and last characters and recursively reversing the rest of the string. Assume the string is passed to the algorithm as an array A of characters, A[p…q], where the array has starting index p and ending index q, and the length of the string is n=q–p+1. The algorithm should have only one base case, when it gets an empty string. Assume you have a swap(A[i],A[j]) function available that will swap the characters in cells i and j. Write the algorithm using pseudocode without any programming language specific syntax. Your algorithm should be correct as per the technical definition of correctness.

(b) (8 points) Draw your algorithm's recursion tree on input string "i<33270!"- remember to show inputs and outputs of each recursive execution including the execution of any base cases.

(a)

```
reverseString(A,p,q)
        if A is empty
                return
        else
                swap(A[p],A[q])
                reverseString(A[p+1], A[q-1])
        return A
```

(b)



(1st output should be !07233<i), accidentally forgot lowercase i)

13. (10 points) *Computational problem solving: Proving correctness:*

Function g (n: nonnegative integer)

        if n ≤ 1 then return(n)

        else return(5*g(n-1) – 6*g(n-2))

Prove by induction that algorithm g is correct, if it is intended to compute the function 3^n-2^n for all n ≥ 0.

Base Case Proof:

For n = 0, 0 is returned. $3^0-2^0=1-1=0$. For n = 1, 1 is returned. $3^1-2^1=3-2=1$.

Both base cases satisfy the function.

Inductive Hypothesis:

$g(n-1) = 3^{n-1} – 2^{n-1}$, and $g(n-2) = 3^{n-2} – 2^{n-2}$

Inductive Step:

Show $g(n) = 3^n – 2^n$.

$g(n) = 5*g(n-1) – 6*g(n-2) = 5*(3^{n-1} – 2^{n-1}) – 6*(3^{n-2} – 2^{n-2})$

$= 5*3^{n-1} – 5*2^{n-1} – 2*3*3^{n-2} + 3*2*2^{n-2} = 5*3^{n-1} – 5*2^{n-1} – 2*3^{n-1} + 3*2^{n-1}$

$= (5-2)*3^{n-1} + (3-5)*2^{n-1} = 3*3^{n-1} – 2*2^{n-1} = \underline{3^n – 2^n}$

14. (12 points) *Computational problem solving: Proving correctness:* The algorithm of Q.11 can also be proven correct using the Loop Invariant method. The proof will first show that it will correctly compute F_0 & F_1 by virtue of lines 1 and 2, and then show that it will correctly compute F_n, n>1, using the LI technique on the for loop. For this latter part of the correctness proof, complete the Loop Invariant below by filing in the blanks. Then complete the three parts of the rest of the proof.

Loop Invariant:

Before any execution of the for loop of line 5 in which the loop variable i=k, 2≤k≤n, the variable last will contain F_{i-2} and the variable current will contain F_{i-1}

Initialization:

If n = 0 or 1, the algorithm returns the correct values immediately. Before the loops start, last and current are set to 1, which are the correct values of F_0 and F_1.

Maintenance:

In step 6, temp is set to last + current, which is also the value of the next number in the sequence as defined by the Fibonacci sequence ($F_i=F_{i-1}+F_{i-2}$). Last is then set to current, and current set to temp in the next 2 steps. This shifts both variables one spot over in the sequence (last to F_{i-1} and current to F_i) to prepare it for the i+1 loop. This holds the loop invariant true in which the variables last and current are the two numbers before the new loop

Termination:

After the final loop is run, current = F_i as shown in the maintenance step. Current is then returned, which is the correct value as defined by the Fibonacci sequence.