**COMP 3270 Homework 1**

100 points. Due by **11:59pm (midnight) on Monday, June 7th, 2021**

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) *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(n2) 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.

**What we know: Machine speed = 4 GHz (4 \* 109 clock cycles/sec) or (2\*107 steps/sec)**

**N = 200 mil (2\*108)**

* **O(n) = 2\*108/2\*107 = 10 sec**
* **O(nlogn) = [(2\*108) log2(2\*108)]/ (2\*107) = (5.5 \* 109)/ (2\*107) = 275.75 sec**
* **O(n2) = [(2\*108)2]/ (2\*107) = (4\*1016)/ (2\*107) = 2\*109  sec**

2. (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.

* **Inputs:** 
  + **Selected destination building (in the background the user’s location is also an input)**
* **Data representation:** 
  + **Two locations: Current and Destination each represented in (latitude,longitude)**
  + **Vacant spots near location of interest represented by a key, value dictionary where the key is the parking spot (latitude, longitude) location. The value is the availability of the parking spot. “Vacant” list contains lots that have at least 5 available spots.**
* **Desired Outputs:**
  + **User interface containing the turn by turn directions to a parking lot with at least 5 empty spaces nearest the selected destination building.**

3. (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.

**A correct and efficient strategy to check the maximum difference between any pair of numbers would be to first sort the array. After sorting, take the difference from the ends of the array, so the first element and the last. Those two will be the maximum difference since this array is sorted.   
The total number of number pairs would be . This combinatorics is the number of ways of choosing 2 numbers (or 1 pair) from the numbers of “n”.**

4. (7 points) *Computational problem solving: Understanding an algorithm and its strategy by simulating it on a specific input:*

Understand the following algorithm. Simulate it mentally on the following four inputs, and state the outputs produced (value returned) in each case: (a) A: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]; (b) A: [─1, ─2, ─3, ─4, ─5, ─6, ─7, ─8, ─9, ─10], ; (c) A: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], (d) A: [─1, 2, ─3, 4, ─5, 6, 7, ─8, 9, ─10].

Algorithm-1 (A:array[1..n] of integer)

sum, max: integer

1 sum = 0

2 max = 0

3 for i = 1 to n

4 sum = 0

5 for j = i to n

6 sum = sum + A[j]

7 if sum > max then

8 max = sum

9 return max

Output when input is array (a) above:

* **55**

Output when input is array (b) above:

* **0**

Output when input is array (c) above:

* **0**

Output when input is array (d) above:

* **11**

What does the algorithm return when the input array contains all negative integers?

* **0 because the sum of the array is a negative number, so the sum will never be greater than max. Which is initialized to 0.**

What does the algorithm return when the input array contains all non-negative integers?

* **Positive integer of the sum of the array.**

What does the algorithm return when the input array contains negative, zero and positive integers?

* **It returns the maximum sum of any consecutive set of numbers in the array. Since there are negatives and non-negative numbers in the array that means that the maximum will not always be the sum of the entire array.**

5. (9 points) *Computational problem solving: Calculating approximate complexity:*

Using the approach described in class (L5-Complexity.pptx), calculate the approximate complexity of Algorithm-1 above by filling in the table below.

|  |  |
| --- | --- |
| Step | Big-Oh complexity |
| 1 **max = 0** | **O(1)** |
| 2 **sum = 0** | **O(1)** |
| 3 **for i = 1 to n** | **O(n)** |
| 4 **sum = 0** | **O(1)** |
| 5 **for j = i to n** | **O(n2)** |
| 6 **sum = sum + A[j]** | **O(n2)** |
| 7 **if sum > max then** | **O(n2)** |
| 8 **max = sum** | **O(n2)** |
| 9 **return max** | **O(1)** |
| Complexity of the algorithm | **O(n2)** |

Notes from class: When approx. the complexity of the algorithm, with no recursion, and there is a nested loop involved… you want to start from the inner loop and out.

6. (18 points) Calculate the detailed complexity T(n) of Algorithm-1. 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 |  |
| 6 | 6 | n2 |
| 7 | 3 | n2 |
| 8 | 2 | n2 |
| 9 | 2 | 1 |

**T(n) = 1 + 1 + (n+1) + n + (n-i) + 6n2 + 3n2 + 2n2 + 2**

7. (5 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

**Proof by Counterexample:**

**Suppose n = 1 2 3 3 3 4 3 5 6 6 7 8 8 8 8**

**Actual Algorithm Output: 3**

**Expected Algorithm Output: 9**

**This algorithm is incorrect. Variable “i” is being reassigned at the beginning of the while loop to be the Array[j+1] instead of Array[j] if j is a consecutive integer. For example, the 3rd identical integer would not be compared to the second integer. This algorithm only checks for a second identical integer in relation to whatever value i is set to at the beginning of the while loop.**

8. (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 F0=1, F1=1, Fi=Fi-1+Fi-2 for i>1.
3. So the assumption in (1) implies that there is at least one input parameter n=k, k≥0, for which the algorithm will produce an incorrect answer.
4. **If n = 0, then the algorithm returns 1, if n = 1 then 1 is still returned.**
5. So in both cases the algorithm returns the correct answer.
6. 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 Fk=Fk-1+Fk-2.
7. When n=k and k>1, **the first two statements that check if n = 0 or if n = 1 will not get executed,** and steps 3-9 will be executed.
8. If k=2, the for loop in steps 5-8 will be executed exactly once. By step 6, temp = last + current = 1 + 1 = F0 + F1. Then step 7 updates last to be equal to current = F1. Step 7 updates current to be equal to temp which is F0 + F1. So the value returned in step 9 is current = F0 + F1 = F2. This is the correct answer. So the k for which the algorithm fails must be greater than 2.
9. If k=3,

**The first two if statements will not be executed since n = k and k > 1, the for-loop will execute twice. The first iteration will return current = F(0)+F(1) = 2, which is correct. Second iteration will return current = F(1) + F(2), which is correct.**

1. But if k= 4,

**The same output happens as k=3, except the loop is executed 3 times and the third iteration outputs current = F(2) + F(3), which is correct.**

1. The above argument can be repeated to show that

**K must be greater than 4, or 5, or 6 … etc to fail. Therefore, showing the above argument and repeatedly showing the output is correct. proves that this algorithm is in fact correct.**

1. That is, for all k > 1 the algorithm returns the correct k-th Fibonacci number.
2. So there is no k for which the algorithm will return a value not equal to Fk-1+Fk-2. This contradicts (3).
3. Therefore, the algorithm must be correct.

9. (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.

**function reverseString(A[p->q])**

**if p < A.length/2 or q < A.length/2:**

**A = swap(A[p+1], A[q-1])**

**else**

**return A**

**return reverseString(A[i+1->j-1])**

**Solution:**

But if k= 4,the for loop in steps 5-8 will be executed exactly once. By step 6, temp = last + current = 2 + 3 = F2+ F3. Then step 7 updates last to be equal to current = F3. Step 8 updates current to be equal to temp which is F2+ F3. So the value returned in step 9 is current = F2+ F3= F4. This is the correct answer. So the k for which the algorithm fails must be greater than 4.

(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.

Diagram

Description automatically generated

10. (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 3n-2n for all n ≥ 0.

**Base Case Proof:**

**n = 0:**

**30-20 = 0. This is correct since the algorithm returns n if n<=1. n has been returned.**

**n = 1:**

**31-21 = 1. This is correct.**

**Therefore, the base case has been proven to be correct.**

**Inductive Hypothesis:**

**We assume that 3n-2n is the output for all n = 0, 1 …. k.**

**Inductive Step:**

**We will prove the algorithm provides the correct output for the next k+1-th input. We know that the minimum number that k could be is 0 since the algorithm only works for n>=0. So let’s find out what the k+1-th input would output.  
  
The first time the loop is executed, the algorithm will return n, since k+1 is at least 1.**

**The second loop will return the (5\*g(n-1) – 6\*g(n-2)) algorithm.**

**One thing we want to think about is how can we simplify g(n-1) or g(n-2)?**

**Since we know that n=k+1, we also know that n-1=k, so therefore we can plug-in k for n-1:**

**(5\*g(n-1) – 6\*g(n-2)) = (5\*g(k) – 6\*g(n-2))**

**Now to deal with g(n-2), we know that n = k + 1, and since we want n-2 on the left side, we can manipulate constants to get n-2.**

**n = k + 1**

**-2 -2 We subtract two from both sides**

**n - 2 = k - 1 Now we have the value for n-2**

**Plugging in k-1 for g(n-2):**

**(5\*g(k) – 6\*g(n-2)) =**

**(5\*g(k) – 6\*g(k-1)) = We know that g(n) = 3n-2n, so we can apply that logic and plug-in 3k-2k**

**(5\*(3k-2k) - 6\*(3k-1-2k-1)) = Multiplying everything together**

**(5\*3k)-(5\*2k) - (6\*3k-1)- (6\*2k-1) =**

**3k+1-2k+1  Therefore, the inductive step holds and this algorithm is correct.**

**Solution:**

The above argument can be repeated to show that,for k = n, the for loop in steps 5-8 will be executed exactly once. By step 6, temp = last + current = Fn-2+ Fn-1. Then step 8 updates last to be equal to current = Fn-1. Step 7 updates current to be equal to temp which is Fn-2+ Fn. So the value returned in step 9 is current = Fn-2+ Fn-1= Fn. This is the correct answer.

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 Fk-1+Fk-2. This contradicts (3).

13.Therefore, the algorithm must be correct.

11. (13 points) *Computational problem solving: Proving correctness:* The algorithm of Q.8 can also be proven correct using the Loop Invariant method. The proof will first show that it will correctly compute F0 & F1 by virtue of lines 1 and 2, and then show that it will correctly compute Fn, 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 **fk-2** and the variable current will contain **fk-1**

Initialization:

We will prove that the LI holds before the loop starts.

The loop will start at i = 2, last = 1, and current = 1, therefore the loop invariant holds.

Maintenance:

We will prove that the LI holds before the i-th execution of the loop, then it will also hold before the next (i+1)th execution.

Suppose that i=k and we want to look at the loop that executes right at current = Fk.

To find current = Fk, we must find the addition of last+current = Fk which would be Fk-2 + Fk-1 = Fk

At the end of the loop, current will be Fk and the loop variable where i=k+1 will hold.

Termination:

With these two proofs, this shows that the loop invariant holds after the last execution of the loop. Therefore the algorithm is correct.

Solution:

Maintenance:

1. Suppose this is true: 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 \_\_Fk-2\_\_\_\_\_and the variable current will contain \_\_\_\_ Fk-1\_\_\_\_\_.
2. When the loop executes, in step 6, temp is updated as temp =last +current = Fk-2+ Fk-1.In step 7, last is updated to current, i.e., last = current = Fk-1. Step 8 updates current as, current = temp = Fk-2+ Fk-1The value returned in step 9, the algorithm returns the value of current = Fk-2+ Fk-1= Fk,which is the correct answer.
3. Before the beginning of the next execution last = Fk-1and current = Fk.
4. Thus the LI holds true at the end of the loop execution with i = k, or before the next execution with i = k+1.

Termination:

1. The algorithm terminates after the execution of the outer loop with i=n.
2. In the previous step we showed that LI holds true after each execution of the loop. Therefore it should be true at this point, i.e., before the next execution of the outermost for loop with i= n+1.
3. This means that the value of last= Fn-1and the value of current = Fn-1 +Fn-2**=**Fn.
4. Line 9 returns current as the answer which is the value of the nthFibonacci number. So the algorithm is correct.