**COMP 3270 Assignment 1**

100 points. Due by **11:59pm (midnight) on Monday, September 9th, 2019**

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.

**First, let’s assume the machine we’re running these algorithms on can execute 2 \* 107 steps per second. Next, let’s determine the problem solving time for each algorithm based on the problem of size 200 million (2 \* 108).**

1. **A1 (O(n)): (2 \* 108) / (2 \* 107) = 10 seconds, so this algorithm should solve the problem very quickly!**
2. **A2 (O(nlogn)): (56 \* 108) / (2 \* 107) = approximately 280 seconds, so this algorithm should still solve the problem relatively quickly. As a side note, (56 \* 108) is just log2(n) \* n = 28 \* (2 \* 108), in case there was any confusion as to where these numbers came from.**
3. **A3 (O(n2)): (4 \* 1016) / (2 \* 107) = 2 \* 109 seconds = 2 billion seconds, so this algorithm is inefficient and will not work in a decent amount of time for any large input.**

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.

1. **The mobile application should function with one or two locations as its’ input, with a location being a pair (latitude, longitude). The source and destination locations can be chosen by the user, or the source location can automatically be set with location features on the phone.**
2. **We can define the source location as S(latitude, longitude), the destination location as D(latitude, longitude). We can also define the location of the parking lot closest to the chosen destination as P(latitude, longitude); all inputs and data received from the AUparking application will be objects of this location structure. The Google Map API calls will handle the map display and directions, so no other data representation should be necessary for the application. The application can query the AUparking application if the destination location is registered as an AU parking lot to determine which parking spaces are open; it can then use this information to set the destination location to a more specific spot (the available parking space) and route to this destination appropriately with the Google Map API.**
3. **There are two types of desired outputs, and one that will output if an error occurs. The first desired output is that the destination location is accessible and has been reached. The second desired output is that the destination location has been moved to an accessible location (like a parking lot with at least 5 empty parking spots) and has been reached successfully. The third output is a catch-all for unreachable destinations; it should just be the standard error message about the destination being unreachable. The desired result of this algorithm is to route the user to specific parking spots on AU campus from a current or desired source location.**

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.

**Strategy: We can start by assigning the first number pair difference to both the maximum and minimum variables; note that for an input of one number pair, it is both the maximum and the minimum pair value, so the algorithm should terminate and return this. We can now implement the part that checks for the next number pair difference and assigns it to its’ corresponding variable; if it is smaller than the first number pair difference, it is the new minimum, and if it is larger then it becomes the new maximum. This can be used recursively until the final number pair is reached, and any value that is larger than the previous maximum value or smaller than the previous minimum value will update the respective variable; the resulting maximum and minimum pair value differences will be returned to the user. The total number of number pairs that any algorithm using this strategy will have to consider (as a function of n) should be [n(n – 1)]/2.**

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: **14**

What does the algorithm return when the input array contains all negative integers? **The algorithm will return 0 every time an array filled with negative values is passed to the function, since adding any of the negative values to the sum variable results in a value that is 0 or less every time, so the max variable will never be assigned a new value.**

What does the algorithm return when the input array contains all non-negative integers? **The algorithm will always return the sum of the array as 0 if it contains only non-negative values, since none of the values in the array will be larger than that in comparison.**

What does the algorithm return when the input array contains negative, zero and positive integers? **The algorithm is going to return the sum as the largest sum of any subsequence of numbers in the array; i.e., it may contain the sum of a portion of the array (usually whichever portion contains the most subsequent positive integers).**

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 | **O(1)** |
| 2 | **O(1)** |
| 3 | **O(n)** |
| 4 | **O(1)** |
| 5 | **O(n)** |
| 6 | **O(1)** |
| 7 | **O(1)** |
| 8 | **O(1)** |
| 9 | **O(1)** |
| Complexity of the algorithm | **O(n2)** |

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 | **C1 = 1** | **1** |
| 2 | **C2 = 1** | **1** |
| 3 | **C3 = depends, so we will use 1** | **n + 1** |
| 4 | **C4 = 1** | **n** |
| 5 | **C5 = depends, so we will use 1** | **(i = 1 to n)(i)** |
| 6 | **C6 = 6** | **(i = 1 to n)(i – 1)** |
| 7 | **C7­ = 3** | **(i = 1 to n)(i – 1)** |
| 8 | **C8 = 2** | **(i = 1 to n)(i – 1)** |
| 9 | **C9 = 1** | **1** |

T(n) = **C1 + C2 + C3(n + 1) + C4n + C5[(i = 1 to n)(i)] + C6[(i = 1 to n)(i – 1)] + C7[(i = 1 to n)(i – 1)] + C8[(i = 1 to n)(i – 1)] + C9 = C5[(i = 1 to n)(i)] + (C6 + C7 + C8)[(i = 1 to n)(i – 1)] + (C3 + C4)n + C1 + C2 + C3 + C9 = (i = 1 to n)(i) + 11[(i = 1 to n)(i – 1)] + 2n + 4 = 12[(i = 1 to n)(i)] – 11n + 2n + 4 = 12[(n2 + n)/2] – 9n + 4 = 6n2 + 6n – 9n + 4 = 6n2 – 3n + 4.**

**Answer: T(n) = 6n2 – 3n + 4.**

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 the input file contains 1 2 3 3 3 4 3. Then for the first execution, i = 1, j = 2, and count = 0. On the next execution, i = 3, j = 3, and count = 1. For the next execution, i = 3, j = 4, and count = 0. For the last execution, i = 3, j does not receive a new value (the end of the file is reached) and count remains 1. This means we would get an output of 1, instead of the desired output of 3; therefore, the algorithm is incorrect.**

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 k = 0, then the algorithm returns 1, which is correct since F0 = 1.**

**If k = 1, then the algorithm returns 1, which is correct since F1 = 1.**

So in both cases the algorithm returns the correct answer.

1. 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.
2. When n=k and k>1 **then n cannot be 0 or 1, so steps 1-2 will be ignored** and steps 3-9 will be executed.
3. 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 8 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.
4. If k=3, **the for loop in steps 5-8 will be executed twice. By step 6 on the second execution, temp = last + current = 1 + 2 = F1 + F2. Then step 7 updates last to be equal to current = F2. Step 8 updates current to be equal to temp which is F1 + F2. So the value returned in step 9 is current = F1 + F2 = F3. This is the correct answer. So the k for which the algorithm fails must be greater than 3.**
5. But if k= 4, **the for loop in steps 5-8 will be executed three times. By step 6 on the third execution, 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 algorithm produces the correct output for k < 5.**
6. The above argument can be repeated to show that **for any k > 4, Fk = Fk-1 + Fk-2**.
7. That is, for all k > 1 the algorithm returns the correct k-th Fibonacci number.
8. So there is no k for which the algorithm will return a value not equal to Fk-1+Fk-2. This contradicts (3).
9. 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.

**Reverse(A:array[p…q] of char)**

**i, j: integer**

**1 i = p**

**2 j = q**

**3 if i>=j then return(A)**

**4 swap(A[i],A[j])**

**5 i = i + 1**

**6 j = j – 1**

**7 Reverse(A[i…j])**

(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 close up of text on a white background

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:

**For n = 0, g(0) = 0 == 30 – 20 = 0.**

**For n = 1, g(1) = 1 == 31 – 21 = 1.**

Inductive Hypothesis:

**For n = k, g(k) = 3k – 2k.**

Inductive Step:

**We know that g(n) = 3n – 2n, but we need to show that g(k+1) = 3k+1 – 2k+1. We know that**

1. **g(k) = 3k – 2k**

**by our Inductive Hypothesis, which in turn means that**

1. **g(k-1) = 3k-1 – 2k-1**

**We also get that the value returned from the algorithm for g(k+1) = 5\*g([k+1]-1) – 6\*g([k+1]-2) =**

1. **5\*g(k) – 6\*g(k-1)**

**If we substitute g(k) and g(k-1) in (3) with (1) and (2) respectively, we get**

1. **g(k+1) = 5\*(3k – 2k) – 6\*(3k-1 – 2k-1) = 5\*(3k – 2k) – (2\*3k – 3\*2k) = 3k(5 – 2) – 2k(5 – 3) = 3k+1 – 2k+1**

**Since g(k+1) = 3k+1 – 2k+1 for n = k + 1, we know that the algorithm will hold for the function 3n – 2n for all n >= 0.**

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 filling 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:

1. **Before the start of the first iteration of the loop for i = k = 2, both aforementioned variables will have the value 1 assigned to them.**
2. **Last = 1 == F0 = 1 and current = 1 == F1 = 1, so our LI has to be true, since they are both initialized to this before the for loop executes.**

Maintenance:

1. **Suppose that our LI holds true, and also suppose that on the k-th iteration of the loop, Fk = Fk-1 + Fk-2. This means that (a) the variable current = Fk and (b) the variable last = Fk-1.**
2. **If case (a) holds, then at the end of the k-th iteration, the variable current must be equal to Fk.**
3. **On step 6 of the algorithm for this k-th iteration, the variable temp must be equal to the previous values of current and last, i.e., temp = Fk = current + last = Fk-1 + Fk-2.**
4. **On step 7 of the algorithm, the variable last is updated to the value of the variable current, i.e., last = current => Fk-2 = Fk-1.**
5. **On step 8 of the algorithm, the variable current is updated to the value of the variable temp, i.e., current = temp => Fk-1 = Fk = Fk-1 + Fk-2; this proves that case (a) holds.**
6. **We can also see that case (b) holds, since line 4 of this proof shows that last = Fk-1.**
7. **Since these two cases hold, we can see that on the k-th iteration of the loop, Fk = Fk-1 + Fk-2.**
8. **So our LI holds true, and we know that before the start of the (k+1)-th iteration of the loop, Fk-1 and Fk-2, i.e., the variables current and last will hold the correct values and the LI will be true.**

Termination:

1. **Initialization showed that the LI will be true before the loop begins (i.e., before the 1st iteration of the loop with i = 2).**
2. **Maintenance showed that if the LI was true before an iteration, it would still be true before the next iteration.**
3. **So LI must be true before the second iteration of the loop with i = 3, before the third with i = 4 and so on, i.e., the LI will be true before every iteration of the loop.**
4. **The loop ends after the nth iteration, i.e., before the (n+1)-th iteration. The LI must be true at that time.**
5. **LI: 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.**
6. **This means at the end of the nth iteration, last = Fn-1 and current = Fn-1 + Fn-2 = Fk.**
7. **Line 9 of our algorithm returns the variable current as the answer which is the value of the nth Fibonacci number. Thus, this algorithm computes the first n numbers of the Fibonacci sequence correctly!**