**CSC 3110**

**Algorithm Design and Analysis**

(30 points)

Due: ﻿﻿01/31/2024 11:59 PM﻿﻿﻿

**Note:** **Submit answers in PDF document format. Please read the submission format for appropriate file naming conventions.**

1. Exercises 1.1
   1. Problem 4 (2 points)

A black text on a white background

Description automatically generated  
The algorithm needs to deal with an unsorted section of the array, and then put sorted sections into it.

The laziest solution is an insertion sort

1.)Define an array of length |n|  
2.)Fill array with the sorted numbers (with the array starting at zero)

3.)Starting at i = 0,  
4.) find the minimum from i to n using a temporary variable that compares each position against it, and swaps out the value compared against if it is less than the temporary variables value.  
5.)place the minimum at position i

6.)Increase the value of i by plus one

7.)Repeat until i = (n-1)

* 1. Problem 12 (2 points)  
     A close up of text

     Description automatically generated  
     Rule set:  
     1.) All doors initially closed

2.) Every pass begins at locker 1.  
3.)The door of every ith locker is toggled on each pass. Locker/i is always toggled by pass.  
In boring math terms this means that for any door on any pass, the number of times it is toggled is equal to the number of times that i can divide the value of i. In more boring math this means that unless the number dividing the ith door is a square of the number i, then there will be an even number of numbers that can divide i. This means that the doors that are in the position for squares of i are the only doors that will remain open after the last pass for ANY number of passes. This roughly works out to abs(sqrt(n)).

1. Exercises 1.2
   1. Problem 2 (2 points)  
      A close up of a puzzle

      Description automatically generated  
      Ruleset:  
      1.)Everyone must cross the bridge

2.)The flashlight MUST be present to cross the bridge (and since it can only be walked across, this means that one person will have to constantly double back every time)

3.)The pace of the crossing party is equal to the speed of the slowest party member.

4.)The party members may cross in pairs

5.)There is only 17 minutes to cross.  
6.)The party members as represented by time it takes to cross are 1,2,5,10

My first instinct when solving this puzzle is to continuously shuttle the fastest person across the bridge as the runner for the flashlight, let’s test that out to determine how long that takes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Crossing Number | 1 | 2f | 3 | 4f | 5 |
| Party Members | 1,2 | 1 | 1,5 | 1 | 1,10 |
| Slowest Party Member | 2 | 1 | 5 | 1 | 10 |
| Time taken to cross | 2 | 1 | 5 | 1 | 10 |
| Total time elapsed | 2 | 3 | 8 | 9 | 19 |

Well, that didn’t work, I exceeded the total allotted time by two minutes. Let’s try that again but try bunching up the slowest party members together since this means that they won’t delay the runner. However this also means that we need to send someone to the other side to get the runner first.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Crossing Number | 1 | 2f | 3 | 4f | 5 |
| Party Members | 1,2 | 1 | 5,10 | 2 | 1,2 |
| Slowest Party Member | 2 | 1 | 10 | 2 | 2 |
| Time taken to cross | 2 | 1 | 10 | 2 | 2 |
| Total time elapsed | 2 | 3 | 13 | 15 | 17 |

This crossing order solves the puzzle.

* 1. Problem 7 (2 points)  
       
     QUALITIES AN ALGORITHM SHOULD POSSESS  
     1.) Efficiency: An algorithm should strive to be efficient in speed and also the amount of space that the algorithm uses.  
     2.) Simplicity: Programs that are easier to understand and program usually contain fewer bugs – they are also easier for future coders to work with.  
     3.)Generality – The more applicable an algorithm is to more circumstances, the more useful it is.  
       
     Steps to convert an algorithm to code  
     1.) Determine the set of rules your algorithm is bound by.  
     2.) Convert your algorithm to a step by step process

3.) Convert each step in the process to line of code that matches the steps iterated.

4.) Test your code

5.) Refine your code.

1. Exercises 1.3
   1. Problem 5 (2 points)  
      A diagram of a hexagon

      Description automatically generated  
      The shape of the graph is an outer shape with an inner shape, if one were to ONLY complete the outer shape or inner shape, they would be forced to double back, therefore the path MUST enter the inner shape, complete a circuit within, then exit back out to the outer shape.  
        
      A drawing of a hexagon

      Description automatically generated

Look at this squiggly solution, it sure is.

* 1. Problem 8 (parts a & b) (2 points)  
     A diagram of a square

     Description automatically generated  
     a.) We can use the graph coloring problem to color the map so that no two neighboring regions are colored the same by creating a graph where there are vertices for each of the regions, with edges between the vertices where the regions share a border.  
     A drawing of a house and a number

     Description automatically generated with medium confidence  
       
     b.) Since no two colors of the same can border each other, we can think of the solution like a pinwheel, with a central color in the middle, and alternating colors on each side. This should minimize the number of colors needed.   
       
     A yellow and blue squares with black scribbles

     Description automatically generated

Looks like it took four colors.

* 1. Problem 9 (2 points)  
     A black text on a white background

     Description automatically generated  
     Fully connected means the node has an edge connecting it to every other node. Therefore we need to check to see if each node has a direct connection to every other node.

1. Start at node I = 0
2. Check to see if i connects to nodes i+1 to n, return false if there is no connection, return true after checking to connection n and no false has been returned.
3. Increase i by one
4. Repeat 3 and 4 until I = n  
     
   This algorithm should work, it also takes into account connections that have already been checked previously in the algorithm due to the increasing value of i.
5. Exercises 1.4:
   1. Problem 6 (2 points)  
      A black text on a white background

      Description automatically generated  
      The inequality presented says that the height of a binary tree lies between the log2(n) of its vertices, and the value of its total number of vertices minus one.  
        
      The height of a tree is the largest possible number of edges from the root to the leaf. Since it is not possible for there to be more edges than the total number of edges minus one, the right side of the inequality is true.  
        
      Assuming a binary tree is perfectly full, this means there are two edges for any single edge at any point in the tree 2^n, this means that the shortest a tree can be is represented by taking the logarithm of this. This means the left side of the inequality is true.
   2. Problem 7 (parts a-c) (2 points)   
      A white background with black text

      Description automatically generated  
      ADT priority queue means ‘abstract data type’ priority queue, so….just a priority queue. Which requires the following operations:  
      1.) Finding its largest element

2.)Deleting its largest element  
3.)Adding a new element.  
  
a.) Finding the largest element in an unsorted array can be accomplished by scanning through the array to find the largest element (use a temporary variable assigned value array[0], and compare it against all other elements, replacing it if the element is larger), once the largest element is found it can either be returned as a value, or deleted. Because the array is unsorted the new values to the queue can be added anywhere, usually this is done with a .append type function which either directly adds items to the array, or creates a new array out of all the previous values of the old array, and adds the new value in there too.  
b.)Assuming the array is already sorted in ascending order, this means that the largest element of the array can be found at array[n], where it can also be deleted. Adding a new element requires scanning the array up to array [n-1] starting at position array[i+1], and comparing the values of array[i] and array[i+1], if the inserted value is between the values, it should be placed at that position. (Or create a new array of length n+1, copy the original array into it, and then perform swaps of the new element until it is no longer greater than the item being compared against)  
c.)The largest element in a binary search tree can be found on the right most edge of the binary search tree. Deleting it involves deleting the edge indicating that the element should point there. Adding a new element to the binary search tree involves starting at the root of the binary search tree, and comparing the new item against the value of the item stored in the node, if the new element is greater it is compared against the value of the node to the right, if it is less is compared against the value of the node to the left. This process repeats until the bottom of the tree is reached, and the new value is added in via an edge.

1. Exercises 2.1:
   1. Problem 4 (parts a & b) (2 points)  
      A white text on a white background

      Description automatically generated  
      a.) Remember, gloves come in right and left handed pairs The best case scenario is that a matching pair of gloves is pulled on the first try, so that only two gloves need to be pulled.   
        
      The worst case scenario is that you pull the left handed variants of all colors of gloves before drawing the right handed variant of the glove, meaning that worst case is n+1, or 5+4+2+1 = 12 tries  
        
      b.)The best case scenario is that two of the same pair are missing, and the worst case scenario two of different pairs are missing. There are a total of 10 socks, the total number of different outcomes to choose 2 socks out of the 10 is 45. The number of best case scenarios is 5/45 (the 5 times where it’s a pair that is missing), the number of worst case scenarios is 40/45 (all other scenarios). The average scenario is roughly 3 + 5/45 socks are missing.
   2. Problem 5 (parts a & b) (2 points)  
      A white background with black text

      Description automatically generated  
      a.) Positive decimal integer means whole number above the value of zero. Two digits places are needed for every value of 1 for a base 10 number system, this requires taking the logarithm of that number log2(n) and adding +1 (there is always one digit required for the decimal place in the event that the number produces a 0 from the log function). The resulting function is therefore always log2(n) +1.  
        
      b.) Remember, n>0, which means that there will always be at least one decimal place occupied by the resulting binary number. See previous answer in a why log2(n) digit places(bits) are needed. The resulting answer is therefore log2(n+1). To be honest I am deeply uncertain in this answer, it does not feel right – I fail to see the purpose of this in algorithm design.
2. Exercises 2.2:
   1. Problem 10 (2 points)  
      A close up of a text

      Description automatically generated  
      To determine range we have to find the:  
      1.) Maximum  
      2.) Minimum  
      a.)The array is unsorted, which means every element must be considered because it is unknown where each element resides. which means one pass has to be made through the array while comparing temporary variables for maximum and minimum, therefore the efficiency of moving through the unsorted array. (I don’t know how to insert symbols) Is Theta(R).

b.)The array is sorted, but contains repetition, this however represents no issue, because the array is sorted we known that the largest and smallest elements reside at the ends of the arrays, therefore the time necessary to find the maximum and minimum is Theta(1) A[n-1]-A[0].

c.)The linked list is unsorted, which means it is necessary to go through every item in the list. This requires Theta(R) efficiency (linear) because a pass must be made through the entire list.

d.)A height balanced tree has the largest element in the rightmost node, and the smallest element in the left most node, because the tree is balanced the difference between its rightmost and leftmost nodes is at maximum 1. This means that the height of the tree is at minimum log(n+1) -1, which must be traversed twice, so it would be a Theta(log) type efficiency.

* 1. Problem 12 (2 points)  
     A text on a white background

     Description automatically generated  
     Starting at n=0  
     1.) Walk to the right n steps and check for the door.

2.) Walk n steps to the left to return to the middle.

3.)Walk to the left n steps and check for the door.

4.)Increase the value of n by one

5.)Repeat steps 1-4 until the door is located.

1. Exercises 2.3:
   1. Problem 4 (parts a-e) (2 points)   
      A screenshot of a computer

      Description automatically generated  
      a.) The above algorithm adds the summation of the initial sum of zero plus the value of the iterator/factorial of the iterator. So the summation of i/i! from i to n. This looks a lot like the formula used to calculate e. (I’ve seen it before).  
      b.)The basic operation is the sum being added to the the iterator divided by the factorial of the iterator.  
      c.)The basic operation is executed n-1 times (this assumes that in the way it is written that ‘to’ only means ‘up to’ as it means in most programming languages where the right end of the range is NOT included)  
      d.) The efficiency class of this algorithm is factorial, because a factorial is used in its calculation, meaning it will grow more and more inefficient as the value of n expands.  
      e.) The summation of this formula as n approaches infinite is e, so at a certain point it   
      i/i! can be simplified to 1/(i-1)!, which removes one basic component of multiplication. As I approaches infinite, the value of -1 becomes less and less relevant, this means the formula simplifies down to 1/n!, which is the formula for e. Simplifying this function down to ~e would make the efficiency class O(1), which is as good as it gets, so let’s ignore the approximation bit and just skip to making it e.
   2. Problem 7 (2 points)   
      A screenshot of a math problem

      Description automatically generated  
      A close up of black text

      Description automatically generated  
      The current algorithm as implemented above has three iterative loops, whereupon each row is multiplied by the corresponding column, where a third iterative loop actually handles the calculation for the length of the column n. The current total is added to the total for a specific multiplication. I guess you could reduce it by moving from of the initial calculations for A[I,0] and B[0,j] which will save one pass or so through the loop, which is entirely negligible for n as it approaches large sizes, the fact there are three loops means that saving a pass in one of the loops is meaningless, and the efficiency of the algorithm will continue to be cubic.