**COMP 3270**

**Assignment 2**

**100 points**

**Due Tuesday, September 20th by 11:59PM**

Instructions:

1. This is an individual assignment. There are 10 problems.
2. Late submissions **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.
5. Don’t turn in handwritten answers with scribbling, cross-outs, erasures, etc. If an answer is unreadable, it will earn zero points.
6. **(6 points)** Prove that the following algorithm is correct by using the “Proof by Loop Invariants” method.

**Hint**: Loop Invariant **Si=x is not equal to any of the first i elements of the array**

Text, letter

Description automatically generated

**Initiation: Before the first execution of the loop, the variable ‘i’ is set to a value of 0. The index ‘i’ will always take on the value of number of times that the loop has already been executed.**

**Maintenance: Before the kth iteration of the loop, we can find that the value of ‘i’ will be that of k-1, and after the kth iteration i=k. This means that before the k+1 iteration i=k, following the pattern.**

**Termination: The loop will continue to run and iterate until it finds the array element where the ith index will hold a value that is equivalent to the value of x, at which point the loop will return i, or the loop will terminate and return a value of -1 if the index i takes on a value that is greater than n, the size of the array.**

**2. (5 points)** Order the following list of functions by the big-Oh notation. Group together (for example by underlining) those functions that are big-Theta to each other.

Text

Description automatically generated with medium confidence

1/n, 2100, log log n, (logn)1/2, log2n, n0.01, **n1/2, 3n1/2**, 2logn, 5n, 4n3/2, nlog4n, 6nlogn, 2nlog2n, 4logn, n2logn, n3, 2n, 4n, 22^n

**3. (5 points)** Describe a method for finding both the minimum and maximum of n numbers using fewer than 3n/2 comparisons. ***Hint:*** First construct a group of candidate minimums and a group of candidate maximums.

Say we take the n numbers and group them all into pairs and the two numbers in each group, placing the smaller numbers into a group of candidate minimums and the larger numbers into a group of candidate maximums. After going through all the pairs, we know that the maximum of the n numbers will also be the maximum from the candidate maximums, and that the minimum will also be the minimum from the candidate minimums.

**4. (6 points)** Consider the following “proof” that the Fibonacci function F(n), defined as F(1) = 1, F(2) = 2, F(n) = F(n-1) + F(n-2), is O(n):

* Base case (n<=2): F(1) = 1 which is O(1), and F(2) = 2, which is O(2).
* Inductive hypothesis (n>2): Assume the claim is true for n’ < n.
* Inductive step: F(n) = F(n-1) + F(n-2). By induction, F(n-1) is O(n-1) and F(n-2) is O(n-2). Then, F(n) is O((n-1)+(n-2)). Therefore, F(n) is O(n), since O((n-1)+(n-2)) is O(n).

What is wrong with this proof?

The above proof fails to show how the Fibonacci function will run with O(n) time complexity as it is assuming that O(n) is just another function to model a Fibonacci function F(n), which showing O((n-1)+(n-2)) = O(n), is an incorrect way to prove the time complexity of a recursive algorithm.

**5. (12 points)**

Algorithm Mystery(A: Array [i..j] of integer) i & j are array starting and ending indexes

if i=j then return A[i]

else

k=i+floor((j-i)/2)

temp1= Mystery(A[i..k])

temp2= Mystery(A[(k+1)..j]

if temp1<temp2 then return temp1 else return temp2

(a) (1 points) What does the recursive algorithm above compute?

The recursive algorithm computes the minimum value from the array.

(b) (4 points) Develop and state the two recurrence relations exactly (i.e., determine all constants) of this algorithm by following the steps outlined in L7-Chapter4.ppt. Determine the values of constant costs of steps using directions provided in L5-Complexity.ppt. Show details of your work if you want to get partial credit.

T(n) = T(n/2) + T(n/2) if i<j | O(1) if i=j

T(n) = 2T(n/2) if i<j | O(1) if i=j

T(1) = 1

T(n) = 2T(n/2) + 11

(c) (6 points) Use the Recursion Tree Method to determine the precise mathematical expression T(n) for this algorithm. First, simplify the recurrences from part (b) by substituting the constant “c” for all constant terms. Drawing the recursion tree may help but you do not have to show the tree in your answer; instead, fill the table below. Use the examples worked out in class for guidance. Show details of your work if you want to get partial credit.

You will need the following result:



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Level | Level number | Total # of recursive executions at this level | Input size to each recursive execution | Work done by each recursive execution, excluding the recursive calls | Total work done by the algorithm at this level |
| Root | 0 | 1 | N | c | c |
| One level below root | 1 | 2 |  | c | 2c |
| Two levels below root | 2 | 22 |  | c | 4c |
| The level just above the base case level | lg(n-1) | 2lg(n-1)=n-1 | 2 | c | (n-1)c |
| Base case level | lg(n) | 2lg(n)=n | 1 | c | nc |

(d) (1 points) Based on T(n) that you derived, state the order of complexity of this algorithm:

T(n) = → → 2lg(n)+1 → since lg(n) is base 2, we get 2n-1

T(n) = 2n - 1

T(n) = O(n)

**6. (10 points)** T(n)=7T(n/8)+cn; T(1)=c. Determine the polynomial T(n) for the recursive algorithm characterized by these two recurrence relations, using the Recursion Tree Method. Drawing the recursion tree may help but you do not have to show the tree in your answer; instead, fill the table below. You will need to use the following results, where and b are constants and x<1:



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| level | Level number | Total # of recursive executions at this level | Input size to each recursive execution | Work done by each recursive execution, excluding the recursive calls | Total work at this level |
| Root | 0 | 1 | N | cn | cn |
| 1 level below | 1 | 7 |  | cn |  |
| 2 levels below | 2 | 72 |  | cn |  |
| The level just above the base case level | log7(n-1) |  |  | cn |  |
| Base case level | log7(n) | =n | 1 | c | cn |

T(n) = → → 8cn

T(n) = O(n)

**7. (11 points)** Use the substitution method to prove the guess that is indeed correct when T(n) is defined by the following recurrence relations: T(n)=3T(n/3)+5; T(1)=5. At the end of your proof state the value of constant c that is needed to make the proof work.

Statement of what you have to prove: We are proving that T(n) <= c(n) for all numbers of n > 1 where c is a constant value. O(n) represents the upper bound, or worst case, for the equation T(n).

Base Case proof: T(1) = O(1) → We know that T(1) is equivalent to 5, which represents a constant runtime, and we also know that O(1) represents a constant runtime, so it is a valid statement to say that T(1) = O(1).

Inductive Hypotheses: T(k) = O(k) → T(k) <= ck, where k>1

Inductive Step: T(k+1) = O(k+1) → T(k+1) <= 3c(k/3) + 5 → T(k+1) <= ck + 5 →

T(k+1) <= (k+1)c – c + 5 → If we make c = 5, then we can say that T(k+1) <= c(k+1), which is true based on the inductive hypothesis that we have made.

Value of c: c = 5

**8. (16 points)** Guess a plausible solution for the complexity of the recursive algorithm characterized by the recurrence relations T(n)=T(n/2)+T(n/4)+T(n/8)+T(n/8)+n; T(1)=c using the Substitution Method. (1) Draw the recursion tree to three levels (levels 0, 1 and 2) showing (a) all recursive executions at each level, (b) the input size to each recursive execution, (c) work done by each recursive execution other than recursive calls, and (d) the total work done at each level. (2) Pictorially show the shape of the overall tree. (3) Estimate the depth of the tree at its shallowest part. (4) Estimate the depth of the tree at its deepest part. (5) Based on these estimates, come up with a reasonable guess as to the Big-Oh complexity order of this recursive algorithm. Your answer must explicitly show every numbered part described above in order to get credit.

Diagram, text, letter

Description automatically generated

**9. (10 points)** Use the Substitution Method to prove that your guess for the previous problem is indeed correct.

Statement of what you have to prove: We are trying to prove that T(n) = O(nlog(n)), or in other words, that T(n) <= c(nlog(n)) for all n > 1.

Base Case proof: T(1) = O(log(1)) → we know that T(1) = c, representing a constant runtime, and when we simplify O(1log(1)) we will get O(0), which can also be expressed as O(1), which represents an upper bound of constant runtime, therefore T(1) = O(1log(1))

Inductive Hypotheses: T() <= O(nlg(n)) → T(n) <= cnlg(n), where n > 1

Inductive Step: T(n) = O(nlgn) → we can simplify the given equation to say T(n) = 2T(n/2) + n → T(n) <= cnlg(n/2) + n → by logarithmic rules T(n) <= cnlgn – cnlg2 + n → T(n) <= cnlgn – cn +n →

Which is to say that our hypothesis will hold true for a given constant c.

**10. (9 points)** Use the Master Method to solve the following three recurrence relations and state the complexity orders of the corresponding recursive algorithms.

Rules Used:

* If f(n) = O(nc) where c < Logba then T(n) = Θ(nLogba)
* If f(n) = Θ(nc) where c = Logba then T(n) = Θ(ncLog n)
* If f(n) = Ω(nc) where c > Logba then T(n) = Θ(f(n))

1. T(n)=2T(99n/100)+100n

Text, letter

Description automatically generated

1. T(n)=16T(n/2)+n3lgn

A picture containing text, watch

Description automatically generated

1. TT(n)=16T(n/4)+n2

Text

Description automatically generated

**11. (10 points)** Use Backward Substitution (10 points) and then Forward Substitution (10 points) to solve the recurrence relations T(n)=2T(n-1)+1;T(0)=1. In each case, do the following: (1) Show at least three expansions so that the emerging pattern is evident. (2) Then write out T(n) fully and simplify using equation (A.5) on Text p.1147. (3) Verify your solution by substituting it in the LHS and RHS of the recurrence relation and demonstrating that LHS=RHS. (4) Finally, state the complexity order of T(n). You must show your work for parts (1)-(3) to receive credit.

Text, letter

Description automatically generated

Text, letter

Description automatically generated