**COMP 3270**

**Introduction to Algorithms**

**Homework 2**

**Due: September 20th, Friday by 11:59 PM (midnight)**

**Please submit as a PDF document using Canvas**

**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 uploaded on the due date and time will incur late penalty (i.e., 10 point deduction for each late hour) 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 using a document editor (e.g., Word) and submit online as a PDF 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. **10pts)** Compare the following pairs of functions in terms of order of magnitude. In each case, say whether f(n) = O(g(n)), f(n) = Θ(g(n)), and/or f(n) = Ω(g(n))

f(n) g(n)

a. 100n + logn n + (logn)2

**Answer:** f(n) = Ω(g(n))

b. logn log(n2)

**Answer:** f(n) = O(g(n))

c. n2/logn n(logn)2

**Answer:** f(n) = Ω(g(n))

d. n(1/2) (logn)5

**Answer:** f(n) = Ω(g(n))

e. n2n 3n

**Answer:** f(n) = Θ(g(n))

1. **10pts)**

Algorithm Mystery(A: Array [i..j] of integer)

i & j are array starting and ending indexes

begin

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

end

1. **(1 pts)** What does the recursive algorithm above compute?

**Answer:** This algorithm recursively computes the smallest number in an array.

1. **(3 pts)** 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.

**Answer:**

1. There is only one element in the array (i = j).
2. If the base case happens, only the 1st step of the algorithm will be executed, returning the only number in the array.
3. The cost of the if statement is 3, the i = j comparison costs 2, and the return A[i] costs 3, so it has a total cost of 8 for the base case.
4. T(1) = 8
5. When the input size is n, it will check the if condition and fail that at a cost of 5. It will execute the floor function but we do not know the cost of that. It will then hit the first recursive call, and after control is returned it will hit the second recursive call.
6. 2T(n/2) recursive calls will be made, since the array is being split into two halves (the first half is handled in the first recursive call, the second half is handled in the second).
7. The complexity of the last step is 5, since it’s comparing two variables (with <) and can return one of two variables that it must access.
8. T(n) = 2T(n/2) + 8

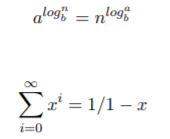
1. **(6 pts)** Use the Recursion Tree Method to determine the precise mathematical expression T(n) for this algorithm. First, simplify the recurrences from part (b) by substituting the constant “c” for all constant terms. Drawing the recursion tree may help but you do not have to show the tree in your 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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** | **cn** | **cn** |
| One level below root | **1** | **2** | **n/2** | **c(n/2)** | **cn** |
| Two levels below root | **2** | **4** | **n/22** | **c(n/22)** | **cn** |
| The level just above the base case level | **logn - 1** | **n - 1** | **n/2logn – 1** | **c(n/2logn – 1)** | **c(n/2n)** |
| Base case level | **logn** | **n** | **n/2logn** | **c(n/2logn)** | **c(n/2n + 1)** |

1. **(1 pts)** Based on T(n) that you derived, state the order of complexity of this algorithm.

**Answer:** O(nlogn)

1. **10 pts)** *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** | **n/8** | **cn/8** | **cn** |
| 2 levels below | 2 | **49** | **n/64** | **cn/64** | **cn** |
| The level just above the base case level | **log8n – 1** | **n – 1** | **n/8log8n – 1** | **cn/8log8n – 1** | **cn/8n** |
| Base case level | **log8n** | **n** | **n/8log8n** | **cn/8log8n** | **cn/8n – 1** |

T(n) = cnlog8n + cn

1. **5 pts)** 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 have to show T(n) >= clogn.

Base Case proof: 5 = T(1) >= clog1 = 0 (for any value of c).

Inductive Hypotheses: Assume 3T(n/3) >= 3(clog(n/3)).

Inductive Step: We have to show T(n) >= clogn.

T(n) >= 3c(logn – log3) + 5 = 3clogn – 3clog3 + 5 <= clogn

when 0 <= 3clog3 – 3clogn – 5.

Value of c: c <= 5/(logn + 3log3 – 3logn)

1. **6 pts)** Find a counterexample to the following claim:

f(n) = O(s(n)) and g(n)=O(r(n)) imply f(n) – g(n) = O(s(n) – r(n))

**Answer:** s(n) = n + 1, f(n) = O(s(n)) = O(n + 1) = O(n) + O(1) = O(n) = n

r(n) = n, g(n) = O(r(n)) = O(n) = n

f(n) – g(n) = O(s(n) – r(n)) <==> n – n = O(n + 1 – n) <==> 0 = O(1) = 1,

which is a counter example, since 0 cannot equal 1.

1. **16 pts)** 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.

A close up of text on a white background

Description automatically generated

1. **10 pts)** Use the Substitution Method to prove that your guess for the previous problem is indeed correct.

Statement of what you have to prove: T(n) >= O(nlogn).

Base Case proof: c = T(1) >= nlogn = log1 = 0, (for any value of c).

Inductive Hypotheses: Assume that T(n/2) >= (n/2)log(n/2)

T(n/4) >= (n/4)log(n/4)

and T(n/8) >= (n/8)log(n/8).

Inductive Step: T(n) >= (n/2)(logn – log2) + (n/4)(logn – log4) + (n/8)(logn – log8) + (n/8)(logn – log8) + nlogn = (nlogn)/2 – (nlog2)/2 + (nlogn)/4 – (nlog4)/4 + (nlogn)/8 – (nlog8)/8 + (nlogn)/8 – (nlog8)/8 + nlogn = 2nlogn – (nlog2)/2 – (nlog4)/4 – (nlog8)/4 when 0 <= (nlog8)/4 + (nlog4)/4 + (nlog2)/2 – 2nlogn.

1. **9 pts)** Use the Master Method to solve the following three recurrence relations and state the complexity orders of the corresponding recursive algorithms.
2. T(n) = 2T(99n/100) + 100n

**Answer:** Since a = 2, b = 100/90, f(n) = 100n, and nlogab = nlog10/92, there will be no value for є that could make cases 1.) or 3.) of the Master Theorem work. This means that case 2 must hold, since f(n) = Θ(nlogba) <==> 100n = Θ(nlog10/92), and thus the solution to the recurrence is T(n) = Θ(nlog10/92lgn).

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

**Answer:** Since a = 16, b = 2, f(n) = n3lgn, and nlog216 = n4, f(n) = Θ(n4 – є) <==> n3lgn = Θ(n4 – є), where є ≈ 1.5, and case 3.) of the Master Theorem works. Therefore, our solution to the recurrence is T(n) = Θ(f(n)) = Θ(n3lgn).

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

**Answer:** Since a = 16, b = 4, f(n) = n2, and nlog416 = n2, f(n) = Θ(n2) <==> n2 = Θ(n2), so case 2.) of the Master Theorem works. Therefore, our solution to the recurrence is T(n) = Θ(n2lgn).

1. **10 pts)** 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.

**Answer:**

Backward Substitution:

1. T(n – 1) = 2T(n – 2) + 1

So T(n) = 2(2T(n – 2) + 1) + 1

T(n – 2) = 2T(n – 3) + 1

So T(n) = 2(2(2T(n – 3) + 1) + 1) + 1

T(n – 3) = 2T(n – 4) + 1

So T(n) = 2(2(2(2T(n – 4) + 1) + 1) + 1) + 1

1. Eventually, T(n) = 2nT(n – n) + 2n – 1 = 2n + 2n – 1 = 2n + 1 – 1
2. LHS = T(n) = 2n + 1 – 1

RHS = 2T(n – 1) + 1 = 2(2(n – 1) + 1 – 1) + 1 = 2(2n) – 2 + 1 = 2n + 1 – 1

1. O(n2)

Forward Substitution:

1. T(1) = 2T(0) + 1 = 3

T(2) = 2T(1) + 1 = 7

T(3) = 2T(2) + 1 = 15

T(4) = 2T(3) + 1 = 31

T(5) = 2T(4) + 1 = 63

1. T(n) = 2n + 1
2. LHS = T(n) = 2n + 1

RHS = 2T(n – 1) + 1 = 2n + 1

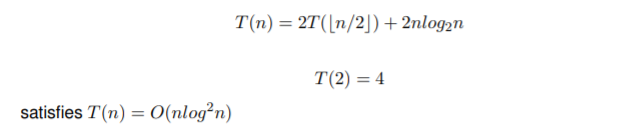
1. O(n2)
2. **5 pts)** Solve the following recurrence relation. Give an exact solution:

T(n) = T(n – 1) + n/2; T(1) = 1

A close up of text on a whiteboard

Description automatically generated

1. **5 pts)** Prove that T(n), which is defined by the recurrence relation



A close up of text on a whiteboard

Description automatically generated

1. **4 pts)** Problem 3.1-3 (p.53) – Explain why the statement “The running time of algorithm is at least O(n2),” is meaningless.

**Answer:** Any algorithm with a quadratic running time is ineffective; that is, when algorithm A receives large values of n, it will take a quadratic (long) amount of time to process, so much so that it is useless to us.