**COMP 3270**

**Assignment 2**

**100 points**

**Due Friday, June 16th by 11:59PM**

Instructions:

1. This is an individual assignment. There are 8 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

Loop Invariant: at the start of each iteration of the loop, the variable **i** is less than the length **n** of the array **A**, and the element **x** is not equal to any of the first **i** elements of the array.

V k (1, i) x

1. Initialization: Before the loop begins, **i** is initialized to 0. Since the array index starts from 0.

i = 1 🡪 (1, i) == 🡪 V k A[k] V, which is true, so any statement regarding the empty set is true.

1. Maintenance: assume the loop invariant is true the start of the iteration, **i** < **n**, and the element **x** is not equal to any of the first **i** elements of the array.

If **A[i] = x** , the algorithm returns **i**, and the loop invariant will hold because **i** < **n**, and **x** is not equal to any of the first **i** elements of the array.

If **x** **A[i]**, the invariant loop will still be true at the start of the next iteration (**i = i + 1**), and the loop invariant holds for the next iteration.

1. Termination: The loop will terminate when **i** >= **n**, and **x** won’t be to any of the first **i** elements of the array.

After the loop terminates, the algorithm will return -1, the element **x** won’t be found in the array **A**. and the loop invariant will return -1.

**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

Group 1: 1/n

Group 2: 2^100

Group 3: log log n

Group 4:

Group 5: log^2 n

Group 6: n^0.01

Group 7: [ ], • 3n^0.5

Group 8: 2^log n, 5n

Group 9: n log4 n, 6n logn

Group 10: [2n log^2 n]

Group 11: 4 n^3/2

Group 12: 4^logn

Group 13: n^2 logn

Group 14: n^3

Group 15: 2^n

Group 16: 4^n

Group 17: 2^2^n

**3. (11 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. **You are required to use an inductive strategy.**

The first step for finding both the minimum and maximum of n numbers using fewer than 3n/2

comparisons are to divide the numbers into pairs and compare the numbers in each pair, taking n/2 comparisons. For each pair, the smaller number will be a candidate minimum and the larger number will be a candidate maximum. Now you have two groups, candidate minimums and candidate maximums.

Compare each candidate minimum with the other candidate minimums to find the overall minimum, taking n/2-1 comparisons. Do the same with the candidate maximums to find the overall maximum. Now you have both the overall minimum and overall maximum in fewer than 3n/2 comparisons.

**4. (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 minimum value in the given 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.

Comparing i=j has a cost of c1 (constant).

Calculating the midpoint k = i + floor((j-i)/2) has a cost of c2 (constant).

The cost of temp1- Mystery(A[i..k]) can be expressed as T(floor(i-i+1)/2).

The cost of temp2 = Mystery(A[(k+1)..j] can be expressed as T(ceiling(-i+1)/2).

Comparing the two values temp1<temp2 has a cost of c3 (constant).

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

= 2T(n/2) + c. when n > 1

T(1) = 1. When n = 1

T(n) = c1 when n =1

T(n) = 2T(floor(n/2) + 2c2 + T(ceiling(n/2) + c3 if n>1

(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 | n/2 | c | 2c |
| Two levels below root | 2 | 4 | n/4 | c | 4c |
| The level just above the base case level | 4 | 2(log(n)-1) = n/2 | n/2(log(n)-1) = 2 | c | (n/2)\*c |
| Base case level | 5 | n/2 | 1 | c | (n/2)\*c |

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

T(n) = O(1)

**5. (20 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 | c | c |
| 1 level below | 1 | 7 | n/8 | cn/8 | 7cn/8 |
| 2 levels below | 2 | 72 | n/82 | cn/82 | 72cn/82 |
| The level just above the base case level | 3 | 7(log8(n) -1) | 8 | C\*8log8(n-1) | 7(log8 (n) -1) \* c\* 8log8(n-1) |
| Base case level | 4 | 7(log8(n)) | 1 | c | 7(log8(n))c |

T(n) = c + 7cn/8 + 72cn/82 + 7(log8 (n) -1) \* c\* 8log8(n-1)  + 7(log8(n))c =(nlog8(7)) = 0.88)

**6. (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: T(n) ≤ cn

Base Case proof: when n = 1. T(1) = 3T(1/3) + 5 = 3T(1) + 5 = 3(5) + 5 = 15 + 5 = 20

T(1) = 5 ≤ c\*5, 1 ≤ c

Inductive Hypotheses: Assume that T(k) ≤ ck holds for all k such that 1 ≤ k < n, where c is a constant chosen such that c ≥ 20.

Inductive Step: T(n) = 3T(n/3) + 5, replace T(n/3) with cn/3:

T(n) ≤ 3(cn/3) + 5 = cn + 5

Since c ≥ 20 and n ≥ 2, cn + 5 ≤ cn + 5n = cn(1 + 5/n).

Now, we show cn(1 + 5/n) ≤ cn for all n ≥ 2.

cn(1 + 5/n) ≤ cn is true if 1 + 5/n ≤ 1 for all n ≥ 2.

If we choose c ≥ 20, then 1 + 5/n ≤ 1 holds true for all n ≥ 2.

then T(n) ≤ cn holds for all n ≥ 2.

Value of c:

T(n) = O(n) for T(n) = 3T(n/3) + 5 is correct with c = 20.

**7. (15 points)** Use the Master Method to solve the following three recurrence relations and state the complexity orders of the corresponding recursive algorithms.

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

a = 2, b = 100/99, f(n) = 100n

nlog100/992 = n1.0202, f(n) = O(nlog100/992-E)

T(n) = O(n)

1. T(n)=16T(n/2)+n3logn

A= 16, b = 2, f(n) = n3logn

Nlog216 = n4, f(n) = n3 \* log(n)

T(n) = O(n3 \* log(n)).

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

a = 16, b =4, f(n) = n2

nlog416 = n2 = f(n)

T(n) = O(n2logn)

**8. (20 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.

To solve the recurrence relation T(n) = 2T(n-1) + 1, with the initial condition T(0) = 1, we will use both backward substitution and forward substitution methods.

Backward:

T(n) = 2T(n - 1) + 1;T(0)= 1

T(n-1) = 2T(n - 2) + 1

So T(n) = 2(2T(n - 2) + 1) + 1 = 4T(n - 2) + 3

T(n - 2) = 2T(n - 3) + 1

T(n) = 2(2(2T(n - 3) + 1) + 1) + 1 = 8T(n - 3) + 7

T(n - 3) = 2T(n - 4) + 1

So T(n) = 2(2(2(2T9n - 4) + 1) + 1) + 1) + 1 = 16T(n - 4) +15

T(n) = 2(n + 1) - 1

Check:

LHS: T(n) = 2T(n - 1)+1 = 2(21 - 1) + 1 = 2(n+1) - 1

RHS: 2T(n - 1) + 1 = 2(n+1)-1

Complexity: T(n) = O(2n)

Forward:

T(n) = 2T(n - 1) + 1

T(0)=1

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

T(2) = 2T(1) + 1 = 6 + 1 = 7

T(3) = 2T(2) + 1 = 14 + 1 = 15

T(4) = 2T(3) + 1 = 30 + 1 = 31

T(5) = 2T(4) + 1 = 62 + 1 = 63

T(n) =2(n-1) T (1) + 2(n-2) + 2(n-3) + 2(n-3) + … + 22 + 21 + 1

T(n) = 3(2(n-1) -1

Check

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

RHS = (2(n-1) -1 = 2\*2n – 1 = 2n+1

Complexity : T(n) = O(2n)