Jalen Powell

6/17/2022

**COMP 3270**

**Assignment 2**

**100 points**

**Due Friday, June 17th 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

* **Initialization:** Before the iteration of the loop, the algorithm is true as it stands with i having been assigned the value of 0. The *x* element is to be found in A[i], so if the only index of A[i] is A[0] then *x* = 0.
* **Maintenance:** Throughout the following iterations, the array A[i] will be scanned continuously as it goes from i < n. If the element x hasn’t been found, i will increment one spot to the right so it can run through the next iterations. If it is true before the first iteration of the loop, it remains true before the next iteration.
* **Termination:** In order for this algorithm to terminate, x has to be found in the array A[i]. The index [i] is incremented to n until the element x is shown at the last iteration. If x is not in the array, the algorithm will terminate once it [i] is = n.

**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. 1/n
2. 2100
3. log(log(n))
4. sqrt(log(n))
5. log2(n)
6. n0.01
7. [sqrt(n)], 3 n0.5
8. 2log(n), 5 n
9. n log4n, 6 n log n
10. [2nlog­2n]
11. 4 n3/2
12. 4log n
13. n2 log n
14. n3
15. 2n
16. 4n
17. 22n

**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.

- Array A of n elements with two variables, Max and Min. If n is odd, then Max and Min will be initialized as A[0], if else, Max and Min will be initialized as A[0] and A[1].

With each index up next in the each of the two cases, the iterations will increment by two. If A[i] > A[i+1], then A[i] is a candidate for Max and A[i+1] is a candidate for Min. The checks will be respective with 3 comparisons no matter what.

Each step will have 3 comparisons and it iterates through the array by increment of two, this makes the number of comparisons less than 3(n/2).

The exact number of comparison are found through:

If n is odd: 3(n-1)/2

If n is even: (3n/2)-2

**4. (18 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 algorithm above is finding out the largest element present in the array and return it after it terminates.

(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.

|  |  |
| --- | --- |
| Stepwise Code | Complexity |
| i & j are array starting and ending indexes |  |
| begin |  |
| If i=j then A[i] | O(1) |
| else |  |
| k=i+floor((j-i)/2) | O(1) |
| temp1=Mystery(A[i…k]) | O(n/2) |
| temp2=Mystery(A[k+1)…j]) | O(n/2) |
| If temp1<temp2 then return temp1 else return temp2 | O(1) |
| end |  |

Total time = T(n) and Input N

T(N) = 2T(N/2)+O(1): O(1) because constant .

T(N) = 2T(N/2)+1

Nlog22=N

T(N)= O(N)

T(n) = T(n/2)+O(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 | 2 | n/2, n/2 | C | C |
| One level below root | 1 | 4 | n/4 | C | 2C |
| Two levels below root | 2 | 8 | n/8 | C | 4C |
| The level just above the base case level | log(n)-1 | n | 1 | C | 11C/2 |
| Base case level | log(n) | 0 | 0 | C | nC |

C

C C

C C C C

| | | |

| | | |

C C C C

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

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

=C+2C+4C…NC

T(n) = Cn

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

* The summation C+2C+4C…NC (log(n) terms), you’ll get T(n)=c(n-1). T(n)∈ θ(n). So the order of complexity of the algorithm is θ(n).

**5. (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 | 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 | Log(n)-1 | 7log7n-1 | 7/8(log7n-1) | cn/8log7n-1 |  |
| Base case level | Log(n) | 7log7n | 1 | c | cn |

T(n) = cn + 7cn/8 + 72cn/82 + …+cn

=cn(1+7/8+72/82+…)log7n

=cn(1/1-7/8)

=8cn

**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:

Base Case proof:

* T(1) = O(n)

As T(1) = 5

Inductive Hypotheses:

* T(k) = O(k) for whatever c and N are set to for all K n N

Inductive Step:

* T(k+1) = O(k+1)

T(k+1) < 3c k/3 + 5

T(k+1) < ck +5

< c(k+1) + 5 -c

If c = 5, we have T(k+1) < c(k+1)

Value of c:

* c = 5

**7. (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.

(3) The depth of the tree in its shallowest part is shown with the shortest path occurring when taking a heavy branch each time. The height *k* is given by n(1/2)k1, meaning nk or k log3n.

(4) The depth of the tree in its deepest part is shown with the longest path occurring when taking a light branch each time. The height *k* is given by n(1/8)k 1, meaning n 8k or klog3/2n.

(5) Big O complexity: T(n) = T(n/2) + T(n/4) + T(n/8) + n -> T(n) = (n).

**8. (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:

Base Case proof:

* T(n) <= kn

For n = 1

T(1) <= k

C <= k

Since k is an arbitrary constant it can be greater than or equal to c, proving the base case

Inductive Hypotheses:

* The statement T(n) <= kn is true for some positive constant k

Inductive Step:

* T(n) = T(n/2) + T(n/4) + T(n/8) + T(n/8) +n

T(n) = kn/2 + kn/4 + kn/8 + kn/8 + n

T(n) = (k+1)n

Kn = (k+1)n

The relation holds for all k >= 1

Value of c:

* C = 5

**9. (9 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

* Master Theorem, a = 198, b = 100, c = 1, f(n) = 100n

Logab = log198100 = > 1

C = 1, since c < logab, T(n) = (nlogab) = (n)

T(n) = (n)

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

* Master Theorem, a = 16, b = 2, c = 3, k = 1

logab = log162 = 4

c < logab, T(n) = (nlogab) = (n4)

T(n) = (n4)

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

* Masters Theorem, a = 16, b = 4, c = 2

logab = log164 = 2

c = logab, T(n) = (nclog(k+1n)) = (n2logn)

T(n) = (n2logn)

**10. (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.

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

T(1) = 2T(0) + 1 = 2x 1+1 = 3 = (2n+1-1)

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

T(3) = 2T(2) + 1 = 2x 7+1 = 15 = (2n+1-1)

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

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

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

= 8T(n-3) + (1+2+4)

= 2nx1+ 2n-1 = 2n+2n-1

= 2n+1-1

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

* Base Case: n = 0, T0= 20+1-1 = 1
* Tk= 2k+2-1 for k0
* Tk+1= 2Tk=1= 2(2k+1-1) + 1 = 2k+2-1

1. The order is exponential. For an input of size n, it will take 2n operations to compute the output.