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.

1. (6 points) Prove that the following algorithm is correct by using the "Proof by Loop Invariants" method.

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

Algorithm arrayFind(x, A):
  Input: An element x and an n-element array, A.
  Output: The index i such that $x = A[i]$ or -1 if no element of A is equal to x.

```
i ← 0
while i < n do
    if x = A[i] then
        return i
    else
        i ← i + 1
return −1
```

Solution: The loop is true at the start of the first iteration of the loop. In iteration I, x is compared to element A[i] and if equal to eachother, returns the index. If x and A[i] aren't equal, i increments. The loop invariant is till true for the new value of I, which would begin the next iteration. If the loop terminates without returning any index from Array A, the algorithm returns -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.

$6n \log n$	2^{100}	$\log \log n$	$\log^2 n$	$2^{\log n}$
2^{2^n}	$\lceil \sqrt{n} \rceil$	$n^{0.01}$	$1/n$	$4n^{3/2}$
$3n^{0.5}$	$5n$	$\lfloor 2n \log^2 n \rfloor$	2^n	$n \log_4 n$
4^n	n^3	$n^2 \log n$	$4^{\log n}$	$\sqrt{\log n}$

Hint: When in doubt about two functions $f(n)$ and $g(n)$, consider $\log f(n)$ and $\log g(n)$ or $2^{f(n)}$ and $2^{g(n)}$.

Solution:

$\frac{1}{n}$, 2^{100}, $\log \log n$, $\sqrt{\log n}$, $\log^2 n$, $n^{0.01}$, $\lceil \sqrt{n} \rceil$, $3n^{0.5}$, $2^{\log n}$, $5n$, $n \log_4 n$, $6n \log n$, $\lfloor 2n \log^2 n \rfloor$, $4n^{\frac{3}{2}}$, $4^{\log n}$, $n^2 \log n$, 2^n, 4^n, 2^{2^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.

---

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?

---

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?

Solution: The smallest number inside of 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.

Solution:

- $T(n) = \Theta(1)$ when $i = j$
- $T(n) = 2T(\frac{n}{2}) + \Theta(1)$ when $i \neq j$

(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: $\sum_{i=0}^{k} x^i = \frac{x^{(k+1)}-1}{x-1}$

Level	Level number	Total # of recursive	Input size to each recursive execution	Work done by each recursive execution,	Total work done by the algorithm at this level

		executions at this level		excluding the recursive calls	
Root	0	2^0	$n/2^0$	16	$16(2^0)$
One level below root	1	2^1	$n/2^1$	16	$16(2^1)$
Two levels below root	2	2^2	$n/2^2$	16	$16(2^2)$
The level just above the base case level	$\log_2(n-1)$	$2^{\log_2(n-1)}$	$n/2^{\log_2}(n-1)$	16	$16(2^{\log_2(n-1)})$
Base case level	$\log_2 n$	$2^{\log_2 n}$	$n/2^{\log_2 n}$	7	$6(2^{\log_2 n)}$

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

$$a^{\log_b n} = n^{\log_b a}$$

$$\sum_{i=0}^{i=\propto} x^i = 1/(1-x) \ when\ x < 1$$

level	Level number	Total # of recursive	Input size to each recursive execution	Work done by each recursive execution,	Total work at this level

		executions at this level		excluding the recursive calls	
Root	0	7^0	$n/8^0$	$cn/8^0$	$cn \times (7/8)^0$
1 level below	1	7^1	$n/8^1$	$cn/8^1$	$cn \times (7/8)^1$
2 levels below	2	7^2	$n/8^2$	$cn/8^2$	$cn \times (7/8)^2$
The level just above the base case level	2	$7^{\log_8(n-1)}$	$n/8^{\log_8(n-1)}$	$cn/8^{\log_8(n-1)}$	$cn \times (7/8)^{\log_8(n-1)}$
Base case level	$\log_8(n-1)$	$_7\log_8(n-1)$	$n/8^{\log_8 n}$	c	$c \times (7/8)^{\log_8 n}$

T(n) =

7. (11 points) Use the substitution method to prove the guess that $T(n) = O(n)$ 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:

$T(n) \leq Cn - 5$ where $n \geq n_0$

Base Case proof:

$T(1) = 5 \leq C \times 1 - 5$

$5 \leq C - 5$

True when $C \geq 10$ & $n_0 = 1$

Inductive Hypotheses:

$T(\frac{n}{3}) \leq \frac{cn}{3} - 5$ when $n \geq n_0$

Inductive Step:

Prove that $T(n) \leq Cn - 5$ when $n > n_0$

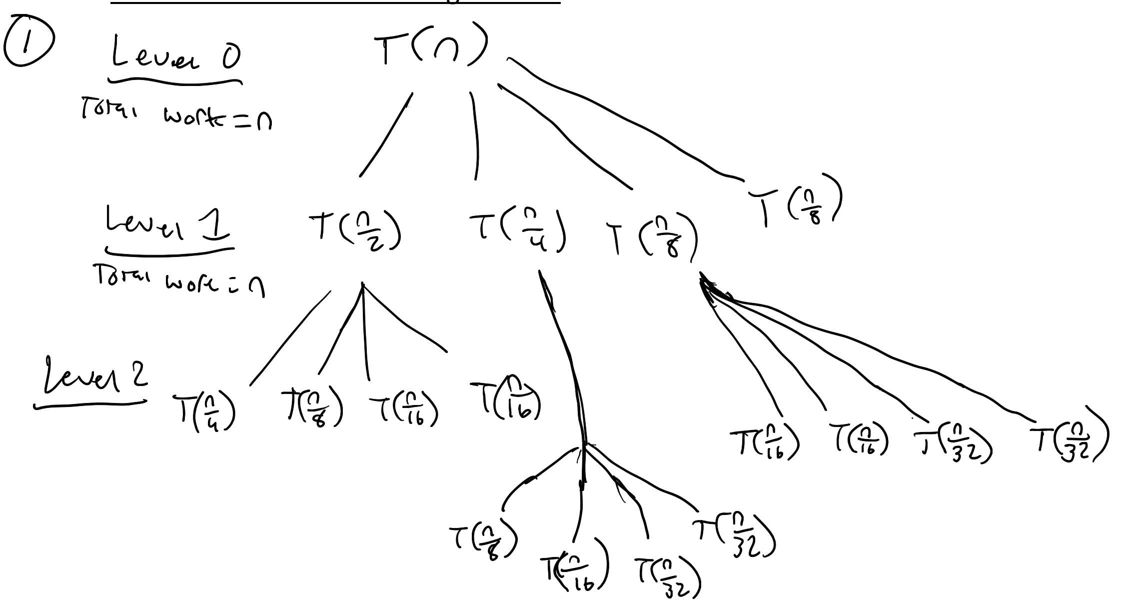
$T(n) = 3t(\frac{n}{5}) + 5 \leq 3[\frac{cn}{3} - 5] + 5$

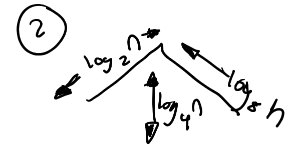
$T(n) = Cn - 15 + 15 \rightarrow T(n) \leq (Cn - 5) - 5 \rightarrow T(n) \leq (Cn - 5)$

Value of c:

$C \geq 10$

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.





(3) $\log_8 n$

(4) $\log_2 n$

(5) $T(n) = O(n \log n)$

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:

$T(n) \leq cn \log n$

Base Case proof: $T(1) = c$

For $n=2$ $T(2) = T(1) + T(0.5) + 2T(0.75) + 2 = ct$

• $T(2) = c+2 \leq 2c \times \log 2$

$T(2) = c+2 \leq 2c$ ► $T(2) = 2 \leq c$

Inductive Hypotheses:

$T(\frac{n}{2}) \leq \frac{cn}{2} \log \frac{n}{2}$; $T(\frac{n}{4}) \leq \frac{cn}{4} \log \frac{n}{4}$; $T(\frac{n}{8}) \leq \frac{cn}{8} \log \frac{n}{8}$

Inductive Step: Prove $T(n) \leq cn\log n$ When $n > n_0$

$$T(n) \leq T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + 2T\left(\frac{n}{8}\right) + n \leq \frac{cn}{2}\log\left(\frac{n}{2}\right) + \frac{cn}{4}\log\left(\frac{n}{4}\right) + \frac{cn}{4}\log\left(\frac{n}{8}\right)$$

$$L + n \leq \frac{cn}{2}\log n + \frac{cn}{4}\log n + \frac{cn}{4}\log n - \left[\frac{cn}{2}\log 2 + \frac{cn}{4}\log 4 + \frac{cn}{4}\log 8\right] + n$$

$$\leq cn\log n - \left[\frac{cn}{2} + \frac{cn}{2} + \frac{3cn}{4}\right] \leq cn\log n - \left(\frac{7cn}{4} - n\right)$$

$$T(n) \leq cn\log n \text{ for } \left[\frac{7cn}{4} - n\right] \geq 0$$

$$\frac{7c}{4} \geq 1 \quad c \geq \frac{4}{7}$$

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

(a) T(n)=2T(99n/100)+100n

$a = 2, \quad b = \frac{100}{99} \qquad f(n) = 100n$

$\log_b a = \log_{\frac{100}{99}}(2) = 68.97$

$T(n) = \Theta(n^{68.97})$

(b) T(n)=16T(n/2)+n³lgn

$a = 16 \qquad b = 2 \qquad f(n) = n^3 \log n$

$\log_b a = \log_2 16 = 4$

$T(n) = \Theta(n^4)$

(c) TT(n)=16T(n/4)+n²

$a = 16 \qquad b = 4 \qquad f(n) = n^2$

$\log_b a = \log_4 16 = 2$

$T(n) = \Theta(n^2)$

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). <u>You must show your work for parts (1)-(3) to receive credit</u>.

Backward Substitution

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

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

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

$T(n-1) = [2T(n-3)+1]+1$

$= 2^2 \cdot T(n-3) + 2 + 1$

$T(n) = 2[2^2 \cdot T(n-3) + 2^1 + 1] + 1$

$= 2^3 \cdot T(n-3) + 2^2 + 2^1 + 1$

$T(n) = \sum_{i=0}^{n} 2i = \frac{2^{n+1}-1}{2-1}$

$T(n) = 2^{n+1} - 1$

Foward Substitution

$T(0) = 1$; $T(1) = 2T(0) + 1 = 2 + 1$

$T(2) = 2T(1) + 1 = 2^2 + 2 + 1$

$T(n) = \sum_{i=0}^{n} = \frac{2^{n+1}-1}{2-1}$

$T(n) = 2^{n+1} - 1$