CS4102 - ALGORITHMS - FALL 2019

DUE THURSDAY, SEPTEMBER 12, 2019 AT 11PM VIA COLLAB

H1:ER6QT

Collaboration Policy: You are encouraged to collaborate with up to 4 other students, but all work submitted must be your own independently written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite them. Do not submit a solution that you are unable to explain orally to a member of the course staff.

Collaborators: Zoha Husain(zh2yn), Ahnaf Khan(aak6sk)

Sources: Cormen, et al, Introduction to Algorithms, Chenghan(TA), ”Substitution Method” slide

PROBLEM 1 *Asymptotics*

Prove or disprove each of the following statements. For 1-3, let $f(n)$ and $g(n)$ be arbitrary asymptotically positive functions.

1. $f(n) \in \Omega(g(n))$ implies $g(n) \in O(f(n))$. (TRUE)

   $\{ f \mid \exists$ constants $c, n_0 > 0$ s.t. $\forall_n > n_0, f(n) \geq c * f(n) \}$
   $c * g(n) \leq f(n)$
   Let $c = 1$
   Let $n_0 = 1$
   $g(n) \leq (\frac{1}{c})\, f(n)$, so $k = (\frac{1}{c})$ and $k > 0$ since $c > 0$
   $g(n) \leq f(n)$
   Susbtitute n: $g(1) \leq f(1)$
   $\forall_n \geq 1, g(n) \leq f(n) \Rightarrow g(n) * c < f(n)$
   Final: $\forall_n \geq 1, g(n) * c \leq f(n) \Rightarrow g(n) \leq f(n)$

2. $f(n) \notin O(g(n))$ implies $\log(f(n)) \notin O(\log(g(n)))$. (FALSE)

   Assume $f(n) \notin O(g(n))$ and $O(log(g(n)))$
   $f(n)$ not less than or equal to $g(n)$
   $f(n) > g(n)$ implies $log f(n) > log(g(n))$ Let's say $f(n) = n^3$ and $g(n) = n^2$
   Take the log of both sides: $logn^3, logn^2$
   As previously mentioned, this proves $f(n) > g(n)$ as n^3 is greater than n^2
   Simplify and you get that $log(n) = log(n)$, which means $log(n) = \Theta log(n)$
   Proof by Contradiction:
   $3log(n) > 2log(n)$
   $log(n) > (\frac{2}{3})log(n)$
   Therefore, we disapprove because $log(n) \in log(n)$, which implies that $log(f(n)) \in log(g(n))$
   In other words: $log(f(n))$ is an element of $log(g(n))$, therefore we disapprove the statement through proof by contradiction

3. $f(n) \in \Theta(f(n/2))$. (FALSE)

   Definition: $\Omega(f(n)) = c * f(n) \leq g(n)$
   Definition: $O(f(n)) = g(n) \geq c * f(n)$
   Θ is $c * (\frac{n}{2}) \leq f(n) \leq f(\frac{n}{2}) * c$
   Proof by Counter Example:

It is true that $2^n \geq \frac{2^n}{2}$
BUT 2^n cannot be less than or equal to $\frac{2^n}{2} * c$ because no matter the value of c, there is a value of n that makes $2^n > \frac{2^n}{2} * c$
Therefore, $f(n)$ cannot be in $\Theta(f(n/2))$

4. $n^{\sqrt{n}} \in O(2^n)$. *Hint: consider* log *rules!* (TRUE)

   Definition: $f(n) \leq c * g(n)$

   Proof.
   There exists $c, n > 0$ such that $\forall_n > n_0, n^{\sqrt{n}} \leq c * 2^n$
   Let $c = 1$
   $n^{\sqrt{n}} \leq 2^n$
   $log(n^{\sqrt{n}}) \leq log(2^n)$
   $\sqrt{n}\, log(n) \leq n * log(2)$
   $log(n) \leq \sqrt{n}$
   $(log(n))/(\sqrt{n}) \leq 1$
   True because the ratio of the two functions will be a fraction since $log(n)$ grows at a slower rate than $\sqrt{n}$ □

5. For any constant $c > 0$, $n^{c \log c} \in O(c^{n \log n})$. (TRUE)

   Proof.
   There exists $c, n > 0$ such that $\forall_n > n_0, n^{clogc} < c^{nlogn}$
   Definition: $n^{c \log c} \in O(c^{n \log n})$
   $log_2 n^{(c*logc)} \leq log_2 n^{(n*logn)}$
   $(c * logc) * logn \leq (n * logn) * logc$
   $logn \leq nlogn$
   True by Definition and Log Rules □

6. Let ε, k be constants. Given that $\forall \varepsilon > 0$, $\log(n) \in o(n^{\varepsilon})$, then $\forall \varepsilon, k > 0$, $\log^k(n) \in o(n^{\varepsilon})$. (TRUE)

   $log^k(n) < c * n^{\epsilon}$
   $\sqrt[k]{(log(n))^k} < \sqrt[k]{c * n^{\epsilon}} \rightarrow \sqrt[k]{c} \rightarrow \sqrt[k]{n^{\epsilon}} = c * n^{\frac{\epsilon}{k}}$
   $log(n) < c * n^{\frac{\epsilon}{k}}$
   n to the power of any positive constant will always be greater than $log(n)$ after some constant c_0

PROBLEM 2 *Iterated Functions*

When solving recurrence relations, we typically need to determine the depth of the recursion. For instance, consider the BetterAttendance algorithm discussed in Lecture 2. The depth of our recursion in this case was $\lceil \log_2 n \rceil$ for a class of n students, i.e. the number of times you can repeatedly divide n by 2 before reaching a value ≤ 1. For this problem, we will explore this idea of iterative re-application of a function until a target value is reached.

Definition 1 *Define the iterated function $f_c^*(n) = \min\{i \geq 0 | f^{(i)}(n) \leq c\}$ where f is a monotonically increasing function over the reals, c is a given constant, and $f^{(i)}(n) = f(f^{(i-1)}(n))$, e.g. $f^{(3)}(n) = f(f(f(n)))$.*

Intuitively, $f_c^*(n)$ gives the count of how many times f must be repeatedly applied to n before reaching a value $\leq c$. Note that since the value is a count, it will always be integral.

We could describe the depth of the recursion for BetterAttendance using this iterated functions notation with $f(n) = \frac{n}{2}$ and $c = 1$, giving $f_1^*(n) = \lceil \log_2 n \rceil$.

Fill in the table below. For each of the given functions $f(n)$ and constants c, give as tight a bound as possible on $f_c^*(n)$. The first row is done for you. In the following, $k > 1$ and $0 < \varepsilon < 1$ are fixed positive constants. *Hint: try a few values for k and ε before generalizing.*

$f(n)$	c	$f_c^*(n)$
$n/2$	1	$\lceil \log_2 n \rceil$
n/k	1	$\lceil \log_k n \rceil$
$n-1$	0	n
$\sqrt{n}$	1	∞
$\sqrt{n}$	2	$\lceil \log_2(\log_2 n) \rceil$
n^{ε}	2	$\lceil \log_{\frac{1}{\epsilon}}(\log_2 n) \rceil$

PROBLEM 3 *Solving Recurrences*

Prove a (as tight as possible) O (big-Oh) asymptotic bound on the following recurrences. You may use any base cases you'd like.

1. $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n$

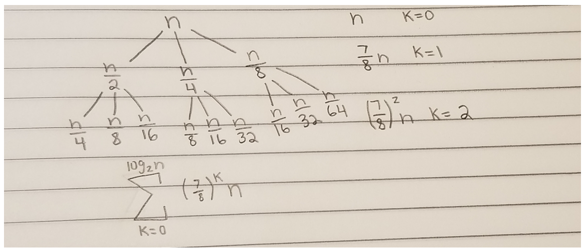
n
K=0
$\frac{7}{8}n$
K=1
$\frac{n}{2}$ $\frac{n}{4}$ $\frac{n}{8}$
$\frac{n}{4}$ $\frac{n}{8}$ $\frac{n}{16}$ $\frac{n}{8}$ $\frac{n}{16}$ $\frac{n}{32}$ $\frac{n}{16}$ $\frac{n}{32}$ $\frac{n}{64}$
$(\frac{7}{8})^2 n$
K=2
$\sum_{K=0}^{\log_2 n} (\frac{7}{8})^K n$

Figure 1: Tree Method

Let $x_0 = 0$
Let $x_1 = n$
Hypothesis: $\forall n \leq x_0$, $T(n) \leq g(n)$
Geometric Sequence: $(\frac{a}{1-r})$ $(\frac{a}{1-\frac{7}{8}})$
$\frac{a}{\frac{1}{8}} = 8a = T(n) = 8(n)$
Base Cases: $T(n) = n \leq 8n = g(n)$ $T(n+1) \leq 8(n+1)$
$T(\frac{n+1}{2}) + (\frac{n+1}{4}) + (\frac{n+1}{8}) + (n+1) \leq 8(n+1)$
$8(\frac{n+1}{2}) + 8(\frac{n+1}{4}) + 8(\frac{n+1}{8}) + (n+1) \leq 8(n+1)$
$4(n+1) + 2(n+1) + 1(n+1) + (n+1) \leq 8(n+1)$
$7(n+1) + (n+1) \leq 8(n+1)$
$8(n+1) \leq 8(n+1)$
Thus, $T(n) = \Theta(n)$

2. $T(n) = 2T(\sqrt{n}) + 2$

Case 1: if $f(n) = O(n^{log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{log_b a})$
Master Theorem: $T(n) = aT(\frac{n}{b}) + f(n)$

Proof.
Let $n = 2^m$
Let $m = log_2 n$ (substitute)
$T(2^m) = 2T(2^{\frac{m}{2}}) + 2$
Let $S(m) = 2S(\frac{m}{2}) + 2$
Master Theorem can be applied to the line above as it is follows the format $T(n) = aT(\frac{n}{b}) + f(n)$, except in terms of m, with $a = 2, b = 2, f(m) = 2$
Plug in: $\Theta(m^{log_b a}) \rightarrow m^{log_2 2}$
Therefore, $S(m) \in \Theta(m) \Rightarrow T(n) \in \Theta(logn)$
Final: $\Theta(logn)$ □

PROBLEM 4 *Mission Impossible*

As the newly-appointed Secretary of the Impossible Missions Force (IMF), you have been tasked with identifying the double agents that have infiltrated your ranks. There are currently n agents in your organization, and luckily, you know that the majority of them (i.e., strictly more than $n/2$ agents) are loyal to the IMF. All of your agents know who is loyal and who is a double agent. Your plan for identifying the double agents is to pair them up and ask each agent to identify whether the other is a double agent or not. Agents loyal to your organization will always answer honestly while double agents can answer arbitrarily. The list of potential responses are listed below:

Agent 001	Agent 002	Implication
"002 is a double agent"	"001 is a double agent"	At least one is a double agent
"002 is a double agent"	"001 is loyal"	At least one is a double agent
"002 is loyal"	"001 is a double agent"	At least one is a double agent
"002 is loyal"	"001 is loyal"	Both are loyal or both are double agents

1. A group of n agents is "acceptable" for a mission if a majority of them ($> n/2$) are loyal. Suppose we have an "acceptable" group of n agents. Describe an algorithm that has the following properties:
   - Uses at most $\lfloor n/2 \rfloor$ pairwise tests between agents.
   - Outputs a smaller "acceptable" group of agents of size at most $\lceil n/2 \rceil$.

   Prove that your algorithm satisfies both requirements.

   In the algorithm I came up with, the two needs are met, but there is a difference between an even and odd number of agents. If there's a chance of one of the agents being disloyal, such as if the pair of agents are DD or LD, then the pair should be removed. If the pair of agents is LL, then one of the agents from the pair should be randomly selected; this means there could be either of the extreme sides, both could be disloyal or both could be loyal. In essense, though, as long as the majority of the agents are loyal, this algorithm will satisfy both requirements. You run through this algorithm until a base case is reached (DL L or LL D). Therefore, if you reach an odd number of agents when going through each pair, then you leave the unpaired agent who remains, whereas if you reach an even number of agents, then you take the unpaired agent remaining.

   For an odd number of agents, such as 3, there are two bases cases. The first base case would be if you had a disloyal(D) and loyal(L) agent together. For a DL pair of agents, the responses could be LD,DD - this would mean there has to be atleast one disloyal agent within the pair. Therefore, as mentioned before, we would remove the pair and get the remainder, which in

this case would be loyal.
The second base case would be if there are two loyal agents paired up with one disloyal agent, LL D. These two loyal agents would ofcourse say they are loyal, but both could be loyal or disloyal. Therefore, you would take one of the agents from this pair, and get rid of the remainders since both of the agents would have to be loyal in this case for the majority to be loyal.

For an even number of agents, an example is LL LD. For the LD pair, you can get responses of LD or DD. With even one disloyal agent in the pair, we can get rid of the whole pair. For the LL pair, you get the repsonse of LL. You end up with a loyal agent in this case after conducting just one round of tests. This meets the first requirement. This also meets the second requirement as we went from four agents to one agent, which is less than the (n/2) of 4 agents.

This algorithm can be applied to any larger set of agents as long as their is a larger majority. With a recursive function applied to the algorithm, you will always end up at these base cases mentioned above.

2. Using your approach from Part 1, devise an algorithm that identifies which agents are loyal and which are double agents using $\Theta(n)$ pairwise tests. Prove the correctness of your algorithm and show that only $\Theta(n)$ tests are used.

   In my algorithm, you divide n number of agents in half pairs at each level. At each level, each agent is asked who is loyal and who is disloyal, which takes n/2 time. Thus, the algorithm I devised is:

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

   This recurrence is continued until the base case is reached, where there is only 1 loyal agent remaining. There is always a loyal agent at the end because we start with a loyal majority. The one loyal agent will go back up at each level to verify who is loyal or disloyal since loyal(L) agents will always tell the truth. Therefore, this agent will be used to pairwise test each agent, getting Θn as the run time.

3. Prove that a conspiracy of $\lfloor n/2 \rfloor + 1$ double agents (who may be working together to evade identification) can foil *all* attempts to find a loyal agent. Namely, you should show that there is *no* algorithm (which could be different from your particular algorithm in Part 2) that will allow identifying even a single loyal agent if there is no majority of loyal agents.

   Base case 1 : DD L The first two agents could claim they are loyal. Based on the algorithm, you would select one and end up with a disloyal agent. You would discard the unpaired agent. If the response from the first two agents was LD, you wouldnt take the pair and would take the remaining one, the loyal agent. Since this is random chance, there is no guarantee of ending up with a loyal agent.

   Base case 2: DL D Using the algorithm, you would get rid of the first two agents because there may be one that is disloyal. You would be left with the last unpaired agent, which is disloyal. There is no way to end up with a loyal agent if there is a disloyal majority. This could also be applied to a larger case.

   If there is a group of agents: LL DD D, one agent would be chosen from each of the first two pairs, which would lead to a group of LD. Since this is an even number of agents, the last agent would also be added to the group to end up with LD L.

PROBLEM 5 (EXTRA CREDIT) *Karatsuba Example*

Illustrate the Karatsuba algorithm on 20194102 × 47295610. Use 2-digit multiplication as your base case.

Karatsuba combine formula: $10^n(ac) + 10^{\frac{n}{2}}((a+b)(c+d) - ac - bd) + bd$
Recursively compute $ac, bd, (a+b)(c+d)$

$a = 2019, b = 4102, c = 4729, d = 5610$
$10^8(2019 * 4729) + 10^4((2019 + 4102) * (4729 + 5610) - (2019 * 4729) - (4102 * 5610)) + (4102 * 5610)$

Break into subproblems: $ac, bd, (a+b)(c+d)$

$2019x4729$ (ac)
$10^4(20 * 47) + 10^2((20 + 19) * (47 + 29) - (20 * 47) - (19 * 29)) + (19 * 29) = 9,547,851$

$4102x5610$ (bd)
$10^4(41 * 56) + 10^2((41 + 02) * (56 + 10) - (41 * 56) - (02 * 10)) + (02 * 10) = 23,012,220$

$6121x10339$ zero pad $\rightarrow$ $006121x010339$ $(a+b)(c+d)$
$10^6(006 * 010) + 10^3((006 + 121) * (010 + 339) - (006 * 010) - (121 * 339)) + (121 * 339) = 63,285,019$

Final Calculation: $10^8(9,547,851) + 10^4((63,285,019) - (9,547,85) - (23,012,220)) + (23,012,220)$
Final Answer: $955,092,372,492,220$