Problem Set 2

data structures and algorithms 2 - fall 2023

cfr5spw

due september 28, 2023 at 11:59 pm

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 share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. 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. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: kab6uw, ejr2ds, hhv6bx

Sources: list your sources

problem 1 *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. For the following two recurrences, it may be helpful to draw out the tree. However, you should prove the asymptotic bound using induction.

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

     Solution:

     proof by induction To prove there is an upper bound to $T(n)$. using c, as some positive constant, We need to prove $T(n) \geq cn$ base case: we will use a value suitable c such that $T(n) \leq cn$ holds true
     ie: n_i=1
     induction:
     we can assume that the relation holds for all values up to n
     T(k) $\leq$ ck for all k where, k$\leq n$

     $Now for$T(n)
     T(n)=T(n$_{2)+T(\frac{n}{4})+T(\frac{n}{8})+n}$

     To ensure that $T(n) \leq cn$, n should not exceed c.
     Using geometric series formula, we know that $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... = 1$. ie for the worst case the work at each level is proportional to n. using the log depth of this tree, the total work done is $O(nlogn)$
     ie: $T(n) = O(nlogn)$

   - $T(n) = 2T(\frac{n}{3}) + T(\frac{n}{6}) + n$

     Solution:for some positive constant c

     T(n) $\leq$ cn
     BASE CASE
     n $\leq$ 1
     For this base case set a constant c so that $T(n) \leq cn$

     INDUCTION T(n)$\leq$ cn holds true for all values up to n

$T(k) \leq ck$
for all $k < n$
For $T(n)$:
$T(n) = 2T\frac{n}{3} + T\frac{n}{6} + n$
Using our inductive hypothesis -¿
$T(n) \leq 2c\frac{n}{3} + c\frac{n}{6} + n$ -¿ $T(n) \leq \frac{2cn}{3} + \frac{cn}{6} + n$ -¿ $T(n) \leq n(\frac{5c}{6} + 1)$
For $T(n) \leq cn$ to hold, the term in the parenthesis should not exceed c.

since we are breaking the problem down each step by $1/3$ is $O(log_3 n)$. The work done at each level is $O(n)$. Ie: the overall work done which is the product of work per level and the number of levels is $O(nlogn)$.

Ie: $T(n) = O(nlogn)$

2. For the following recurrence relations, indicate: (i) which case of the Master Theorem applies (if any); (ii) justification for why that case applies (if one does) i.e., what is a, $f(n)$, ε, etc; (iii) the asymptotic growth of the recurrence (if any case applies).

   - $T(n) = 7T(\frac{n}{5}) + n \log n$

     Solution:a = 7

     b = 5
     f(n) = $(nlogn)$
     to figure out $n^(log_b)a$
     we look at three cases of the Master Theorem:
     1 If $f(n) = O(n^(log_b)a - \in) for some \in > 0, then T(n) = \Theta(n^(log_b)a)$
     2 If $f(n) = \Theta(n^(log_b a)a) then T(n) = \Theta(n^(log_b)a log n)$
     2 If $f(n) = \Omega(n^(log_b)a + \in) for some \in > 0, and a f(n/b) \leq f(n) for some k < 1 and large enough n, then T(n) = \Theta(f(n))$

     COMPARE f(n) = nlogn with $n^(log5)7$ as logn gros more slowly than any positive power of n.

     so we have the in this case the first case of the master theorem applies because: a = 7
     b = 5
     $f(n) = nlogn$
     $\in$ exists such that $nlogn = O(n^(log_5)7 - \in)$
     (iii) asymptotic growth of the recurrence is $T(n) = \Theta(n^(log_5)7)$

   - $T(n) = 3T(\frac{n}{3}) + n \log n$ **Solution:**a = 3

     b = 3
     $f(n) = nlogn$
     calculate $n^log(b)a$ to compare against $f(n)$ to determine which case of the Master Theorem applies

     $nlog(b)a = nlog(3)3 = n^1 = n$

     Comparing $f(n) with n^log(b)a$
     $f(n) = nlogn$ and $nlog(b)a = n$

Master Theorem cases:
1 If $f(n) = O(n^3) where c < log_(b)a$
2: If $f(n) = \Theta(n^l og_(b)alog^k n) for k \geq 0$
3: If $f(n) = \Omega(n^c) where c > log_(b)a and a f(\frac{n}{b}) \leq kf(n) for some k < 1 and sufficiently large n$

Here f(n) is $nlogn$ which is larger than n from $n^l og_(b)a$ but not polynomially larger. It fits the mold of Case 2 where k = 1

Applying the Master Theorem:
Case 2 applies as: a = 3, b = 3, $f(n) = nlogn$
$n^l og(b)a = n$
$f(n) = \Theta(nlog^k n) for k = 1$

Asymptotic Growth:
using case 2 Master Theorem: the solution is $T(n) = \Theta(n^l og_(b)alog^k + 1n)$
$T(n) = \Theta(nlog^2 n)$

ie: $T(n) = \Theta(nlog^2 n)$

PROBLEM 2 *Climate History*

Scientists call *paleoclimatologists* study the history of climate on earth before instruments were invented to measure temperatures, precipitation, etc. They try to reconstruct climate history using data found from analyzing rocks, sediments, tree rings, fossils, ice sheets, etc.

A group is trying to better understand periods of low precipitation (e.g. dryness) for a time period that spans many thousands of years. Using various data sources, they have assigned a value d_i for the relative dryness for each time-unit (let's say it's measured for every century) for this very long time period. For each time-unit i, one of five values has been assigned as d_i:

- -3 for very high precipitation
- -1 for high precipitation
- 0 for normal (or unknown)
- 1 for low precipitation
- 3 for very low precipitation

The scientists want to find the score for the driest period of consecutive years in their data. Because the effects of a drought are cumulative, the score for a period starting at time i and ending at time j will be the value when all scores from i and j are added. Consider this example with $n = 16$:

d_i	0	3	3	0	-3	-3	1	0	3	-1	-1	0	1	1	3	0
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The period [1:2] looks high with a score of 6. But the highest score is 7 for this period [6:14]. Note there are some negative values in the middle of this period, but the high values around those make it worth having a period that includes those negative values. (The period [6:15] also has score 7. There can be more than one period with the largest score, but you just need to return the highest score, so the existence of multiple periods with the same best score doesn't matter.)

There are many ways to solve this problem, including a $\Theta(n^2)$ brute-force approach that calculates scores for all combinations of starting and ending values for a period and keeps track of the largest score. You need to do better than that. (Though if you were ever to code your solution, coding the brute-force approach is a good way to test your algorithm. Coding is **not** required for this problem.)

What you need to do: Describe an algorithm (clearly in words or in pseudocode) that uses a divide and conquer algorithm that solves this problem in $\Theta(n \log n)$ time. Your description should make it clear what the base case is, what work is done in dividing and in combining, and what recursive subproblems are solved. Along with your algorithm's description, give a brief but convincing argument that the time complexity is $\Theta(n \log n)$.

The inputs to your algorithm will be n, the number of samples, and a list d that stores n values, where each is either -3, -1, 0, 1 or 3. Your algorithm will output the largest dryness score for a period in the data (as described above).

Solution:

Create a Hash map(h). The key of the hash map will be the time period(i). Value will be the relative dryness of the time unit d_i.

divide the hash map in half. find the time period of the maximum cumulatative score from time frame of i to time frame of j for each halfl recursively for each. set int variable sum to 0.
the set another which will be equal to $Integer.MIN_VALUE$.
Then go through a for loop that will look at each index and add the value at the index to the sum. If our variable sum is less than $leftSum$ then $leftSum$ = sum. key values of the maximum sum will be stored inside the for loop. This process is then repeated for the other half

Then find the max score of the midpoint of the hash map. Since we must take into account that the overall maximum score exists between the two halves of the hash map. We can find the sum between the two halves in O(n) by starting at the mid point and finding the max sum that will end at a left of mid position, then find the max sum starting from mid + 1 and ending with a point to the right of mid + 1.

Finally combine the sums from each half and compare the sum of the left halve, sum of the right half, and sum of the middle to see where the largest sum exists.

TIME COMP
$T(n) = 2T(n/2) + (n)$
ie: $O(nlogn)$

PROBLEM 3 *Fast Transformations*

Computer graphics software typically represents points in n dimensions as $(n+1)$-dimensional vectors. To make transformations on the points (e.g., rotating a modelled figure, zooming in, or making the figure appear as though seen through a fish-eye lens), we use a $(n+1) \times (n+1)$ matrix T which defines the transformation, and then we multiply each vector by this matrix to transform that point. That is, for vector v, the transformed vector is $v' = T \times v$.

Let's say we are developing software for very high dimension graphics (n dimensions), and we have a transformation T that we would like to apply to a particular point n times. Develop an algorithm which can multiply this $(n+1)$-dimensional point by T (the $(n+1) \times (n+1)$ transformation matrix) n times in $o(n^3)$ (little-oh of n^3) time. Prove this run time.

Solution:

(written out in python)

//if we have a functin function matrixPower(T,n): // we know our base cases are as follows: if n==0: //return IdentityMatrix of size $(n+1) \times (n+1)$ (error here but i think it still makes sense) if n==1: return T

// Calculate halfpower and resilt if n is odd result should = strassM(res,T) halfPower = matrixPower(T, n/2) result = strassenMult(halfPower, halfPower)

if n is odd: result = strassenMult(result, T)

return result

function strassenMult(A, B): if size of A and B are both 1×1: (edited for latex) return $[[A[1][1] * B[1][1]]]$

// split following into 1/4 $A11, A12, A21, A22$ = split(A) $B11, B12, B21, B22$ = split(B)

// Calculate 7 Products with Recursion

P1 = strassenMult($A11$, subtract($B12, B22$) P2 = strassenMult(add($A11, A12$), $B22$) P3 = strassenMult(add($A21, A22$), $B11$) P4 = strassenMult($A22$, subtract($B21, B11$) P5 = strassenMult(add($A11, A22$), add($B11, B22$)
P6 = strassenMult(subtract($A12, A22$), add($B21, B22$) P7 = strassenMult(subtract($A11, A21$), add($B11, B12$)

// from the matrix grab the resulting 1/4's $C11 = add(subtract(add(P5, P4), P2), P6)$ $C12 = add(P1, P2)$ $C21 = add(P3, P4)$ $C22 = subtract(subtract(add(P5, P1), P3), P7)$

// Combine $C = combine(C11, C12, C21, C22)$ return C

PROBLEM 4 *Receding Airlines*

You have been hired to plan the flights for Professor Floryan's brand new passenger air company, "Receding Airlines." Your objective is to provide service to n major cities within North America. The catch is that this airline will only fly you East.

You recognize that in order to enable all your passengers to travel from any city to any other city (to the East) with a single flight requires $\Omega(n^2)$ different routes. Prof. Floryan says that the airline cannot be profitable when supporting so many routes. Another option would be to order the cities in a list (from West to East), and have flights that go from the city at index i, to the city at index $i+1$. This requires $\Theta(n)$ routes, but would mean that some passengers would require $\Omega(n)$ connections to get to their destination.

1. Devise a compromise set of routes which requires no passenger have more than a single connection (i.e. must take at most two flights), and requires no more than $O(n \log n)$ routes. Prove that your set of routes satisfies these requirements.

2. After a few years, passengers start demanding routes from East to West, and you decide to support new routes from East to West (in addition to supporting routes from West to East). Show that with routes in *both* directions, it is possible to connect all n cities with just $O(n)$ routes such that no passenger needs more than a single connection to get to their destination.

Note: In class so far, we've used recurrence relations to count an algorithm's time complexity, i.e., how many basic operations are executed. Here we're using one to count the number of flights. While this is a bit different than measuring an algorithm's time-complexity, we can still use a divide and conquer approach and a recurrence relation to answer these questions.

Solution:

Divide and Conquer Approach: We want to divide the Cities equally into 2 sub sections A and B. We can then map out a direct route from city A(i) to corresponding city B(i) and do this for each city. Because we are mapping each city to one direct route to its corresponding destination this operation should require O(n) routes. We can continue to apply this recursively so that every corresponding city can correspond to a city further to the east by continuing to break down the cities in 2 groups. With each City in A having a corresponding city in B we can create indirect routes from every city in A to every city in B.

COMBINE

- For every city in A, create indirect flights to every city in A. This requires O(n)routes.

This approach guarantees that no passenger needs more than two flights to reach their destination, and all flights are in the eastward direction.

See the routes created and let T(n) represent the total number of routes for n cities. The recursive formula for T(n) is:

$T(n) = T\left(\frac{n}{2}\right) + n$

Solving this recurrence relation using the Master Theorem, we find that $T(n) = O(n \log n)$. This satisfies the requirement that the number of routes is $O(n \log n)$ while ensuring no passenger needs more than two flights to reach their destination.

for the second part of the question being able t move west to east we can connect all n cities with just $O(n)$ routes. Here's how:

Divide the cities into two equal halves, A and B).

conquer

- For each city in A, create a direct flight to the corresponding city in B.

- Recursively apply the same approach to subproblems A and B.

combine outcomes

- For every city in A, create indirect flights to every city in B. This requires O(n) routes.
- For every city in B, create indirect flights to every city in A. This also requires $O(n)$ routes.

This approach guarantees that no passenger needs more than two flights to reach their destination, regardless of the direction. The total number of routes for this strategy is $O(n + n^2) = O(n^2)$. However, since n^2 is $O(n)$, we can say that the total number of routes is $O(n)$, satisfying the requirement and making sure 1 connection maxiumum is kept.

PROBLEM 5 *Bazinga!*

Theoretical Physicist Sheldon Cooper has decided to give up on String Theory in favor of researching Dark Matter. Unfortunately, his grant-funded position at Caltech is dependent on his continued work in String Theory, so he must search elsewhere. He applies and receives offers from MIT and Harvard. While money is no object to Sheldon, he wants to ensure he's paid fairly and that his offers are at least the median salary among the two schools' Physics departments. Therefore, he hires you to find the median salary across the two departments. Each school mantains a database of all of the salaries for that particular school, but there is no central database.

Each school has given you the ability to access their particular data by executing *queries*. For each query, you provide a particular database with a value k such that $1 \leq k \leq n$, and the database returns to you the k^{th} smallest salary in that school's Physics department.

You may assume that: each school has exactly n physicists (i.e. $2n$ total physicists across both schools), every salary is unique (i.e. no two physicists, regardless of school, have the same salary), and we define the *median* as the n^{th} highest salary across both schools.

1. Design an algorithm that finds the median salary across both schools in $\Theta(\log(n))$ total queries.
2. State the complete recurrence for your algorithm. You may put your $f(n)$ in big-theta notation. Show that the solution for your recurrence is $\Theta(\log(n))$.
3. Prove that your algorithm above finds the correct answer. *Hint: Do induction on the size of the input.*

Solution:

ALGO FOR MEAN SALARY AL = 1; AH = n; BL = 1; BH = n. While AL ¡= AH and BL ¡= BH:
midA = (AL + AH) / 2
midB = (BL + BH) / 2
if A[midA] + B[midB] ¡ n:
AL = midA + 1
BL = midB + 1
else:
AH = midA - 1
BH = midB - 1
Return max(A[AL], B[BL])

Now Recurrence - split the problem into half each iteration so the recurrence relation can be represented as:
$T(n) = T(\frac{n}{2}) + \Theta(1)$
Where $\Theta(1)$ represents the constant time it takes to execute a single query.

MASTER THEOREM $T(n) = aT(\frac{n}{b}) + f(n)$
Where:
a = 1, b = 2, and $f(n) = \Theta(1)$
The critical exponent for this case is:
$critE = log_b(a) = log_2(1) = 0$

Given that $f(n) = \Theta(1) = \Theta(n^0), theMasterTheoremimpliesthatT(n) = \Theta(log(n))$

PROOF
Base case for n = 1 Compare all smaller salries and return the highest salary; Must Assume the algorithm works correctly for k physicists
so: Inductive Step: Consider $k + 1$ physicists.

if $(midA + midB)th$ combined salary is ¡ n, it means that the median is in the upper half of A OR B.
so then well have to update the "AL" and "BL" to look in the upper halves.
else, median is in the lower half of A OR B -¿ or both. So we can update "AH" and "BH" to search in the lower halves.

By induction, it follows that our algorithm correctly identifies the median salary across both schools in $\Theta(log()n)$ total queries

PROBLEM 6 *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 $\lceil n/2 \rceil$.
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.

Solution:

use a divide-and-conquer strategy similar to receading airlines problem.
Divide the group of agents into two equal halves, A and B. Ask each agent in A about every agent in B. If there is at least one pair where both agents claim the other is a double agent, eliminate both agents from consideration. pew pew
If there is a pair where one agent claims the other is a double agent, eliminate the agent making the claim. If there are pairs where both agents are claimed to be loyal, keep both. Recurse on the remaining agents in each half.

This algorithm uses $\lfloor n/2 \rfloor$ pairwise tests and outputs a smaller "acceptable" group of agents of size at most $\lceil n/2 \rceil$.
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.

Solution:

Implement the divide-and-conquer algorithm as described in Part 1. This algorithm will make $\lfloor n/2 \rfloor$ pairwise tests in each recursive step. In each step, it reduces the number of agents by at least half. Therefore, the total number of tests made by this algorithm is:

$$\lfloor n/2 \rfloor + \lfloor n/4 \rfloor + \lfloor n/8 \rfloor + \ldots + 1 = \Theta(n)$$

algorithm uses $\Theta(n)$ pairwise tests to identify which agents are loyal and which are double agents.