CSCI-GA.1170-001/002 Fundamental Algorithms September 29, 2019

Solutions to Problem 1 of Homework 3 (24 points)

Name: jingshuai jiang (jj2903) *Due: 5pm on Thursday, September 26*

Collaborators: NetID1, NetID2

The sequence $\{F_n \mid n \geq 0\}$ are defined as follows: $F_0 = 1$, $F_1 = 1$, $F_2 = 2$ and, for $i > 2$, define $F_i := F_{i-1} + F_{i-2} + 2F_{i-3}$.

(a) (2 Points) We can think of the above recurrence relation as a matrix equation. More specifically, the relation can be represented as an equation of the following form:

$$\mathbf{A} \cdot \begin{pmatrix} F_i \\ F_{i-1} \\ F_{i-2} \end{pmatrix} = \begin{pmatrix} F_{i+1} \\ F_i \\ F_{i-1} \end{pmatrix}$$

What is the satisfying value of **A**? (**Hint**: Consider the simpler case of a Fibonacci Sequence, i.e, $F_i := F_{i-1} + F_{i-2}$ for $i > 1$ and $F_0 = 0, F_1 = 1$. How would you set up the matrix equation?)

Solution: $\mathbf{A} =$

$$\begin{bmatrix} 1 & 1 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

□

(b) (6 Points) Use the equation from part (a) and the ideas of fast $O(\log n)$ time exponentiation to build an efficient algorithm for computing F_n. Analyze it's runtime in terms of the number of 3×3 matrix multiplications performed (each of which takes a constant number of integer additions/multiplications).

Solution:

□

(c) (6 Points) Prove by induction that, for some constant $a > 1$, $F_n = \Theta(a^n)$. Namely, prove by induction that for some constant c_1 you have $F_n \leq c_1 \cdot a^n$, and for some constant c_2 you have $F_n \geq c_2 \cdot a^n$. Thus, F_n takes $O(n)$ bits to represent. What is the right constant a and the best c_1 and c_2 you can find. (**Hint**: Pay attention to the base case $n = 0, 1, 2$. Also, you need to do *two* very similar inductive proofs.)

Solution:

Conclusion $c_1 = 1, a = 2, c_2 = \frac{1}{2}$

$$F_n \leq c_1 \cdot a^n$$

then

$$F_0 = 1 \leq C_1 \cdot a^0, F_1 = 1 \leq C_1 \cdot a^1, F_2 = 2 \leq C_1 \cdot a^2$$

then we get that

$$C_1 \geq 1, 1 \leq C_1 \cdot a^1 and 2 \leq C_1 \cdot a^2$$

assume it is true for F_{n-1} then

$$F_n = F_{n-1} + F_{n-2} + 2F_{n-3} \leq C_1 \cdot a^{n-1} + C_1 \cdot a^{n-2} + 2 \cdot C_1 \cdot a^{n-3} \leq C_1 \cdot a^n$$

then we get

$$(a-2) \cdot (a^2 - a + 1) \geq 0$$

and we get $a \geq 2$

$$F_n \geq c_2 \cdot a^n$$

then

$$F_0 = 1 \geq C_2 \cdot a^0, F_1 = 1 \geq C_2 \cdot a^1, F_2 = 2 \geq C_2 \cdot a^2$$

then we get that

$$C_2 \leq 1, 1 \geq C_2 \cdot a^1, 2 \geq C_2 \cdot a^2$$

assume it is true for F_{n-1} then

$$F_n = F_{n-1} + F_{n-2} + 2F_{n-3} \geq C_2 \cdot a^{n-1} + C_2 \cdot a^{n-2} + 2 \cdot C_2 \cdot a^{n-3} \geq C_2 \cdot a^n$$

then we get

$$(a-2) \cdot (a^2 - a + 1) \leq 0$$

and we get $a \leq 2$

since $a = 2$ then we get $C_1 \geq 2$ and $C_2 \leq \frac{1}{2}$ then we get $c_1 = 1, a = 2, c_2 = \frac{1}{2}$ □

(d) (6 points) In your algorithm of part (b) you only counted the number of 3×3 matrix multiplications. However, the integers used to compute (F_i, F_{i-1}, F_{i-2}) are $O(i)$ (in fact, $\Theta(i)$) bits long, by part (b). Thus, the 3×3 matrix multiplication used at that level of recursion will not take $O(1)$ time. In fact, using Karatsuba's multiplication, let us assume that the last matrix multiplication you use to compute (F_i, F_{i-1}, F_{i-2}) takes time $O(i^{\log_2 3})$. Given this more realistic estimate, analyze the actual running time $T(n)$ of your algorithm in part (b).

Solution: INSERT YOUR SOLUTION HERE □

(e) (4 points) Finally, let us look at the naive sequential algorithm which computes $F_3, F_4, \ldots, F_n$ one-by-one. Assuming each F_i takes $\Theta(i)$ bits to represent, and that integer addition/subtraction takes time $O(i)$ (multiplication by two can be implemented by addition), analyze the actual running time of the naive algorithm. How does it compare to your answer in part (d)?

Solution: INSERT YOUR SOLUTION HERE □

CSCI-GA.1170-001/002 Fundamental Algorithms September 29, 2019

Solutions to Problem 2 of Homework 3 (8 Points)

Name: jingshuai jiang (jj2903) *Due: 5pm on Thursday, September 26*

Collaborators: NetID1, NetID2

Find a *divide-and-conquer* algorithm that finds the maximum and the minimum of an array of size n using at most $3n/2$ *comparisons* (between elements of the array). (Note that we are not asking for an iterative algorithm. We are asking for you to *explicitly use recursion*.) Derive an *exact* recurrence for the number of *comparisons* of your algorithm and prove it using induction.
(**Hint**: Your conquer step should make a constant number of comparisons. Be careful for what n you stop recursing.)

Solution: □

CSCI-GA.1170-001/002 Fundamental Algorithms September 29, 2019

Solutions to Problem 3 of Homework 3 (12 points)

Name: jingshuai jiang (jj2903) *Due: 5pm on Thursday, September 26*

Collaborators: NetID1, NetID2

An array $A[0 \dots (n-1)]$ is called *rotation-sorted* if there exists some some cyclic shift $0 \le c < n$ such that $A[i] = B[(i + c \bmod n)]$ for all $0 \le i < n$, where $B[0 \dots (n-1)]$ is the sorted version of A.[1] For example, $A = (2, 3, 4, 7, 1)$ is rotation-sorted, since the sorted array $B = (1, 2, 3, 4, 7)$ is the cyclic shift of A with $c = 1$ (e.g. $1 = A[4] = B[(4+1) \bmod 5] = B[0] = 1$). For simplicity, below let us assume that n is a power of two (so that can ignore floors and ceilings), and that all elements of A are distinct.

(a) (4 points) Prove that if A is rotation-sorted, then one of $A[0 \dots (n/2-1)]$ and $A[n/2 \dots (n-1)]$ is fully sorted (and, hence, also rotation-sorted with $c = 0$), while the other is at least rotation-sorted. What determines which one of the two halves is sorted? Under what condition *both halves* of A are sorted?

Solution: Since A is roatation-sorted, then exits some C that $A[i] = B[(i + c \bmod n)]$

if $0 \le c \le \frac{n}{2}$ then $A[0 \dots (n/2-1)] = B[c \dots \frac{n}{2} - 1 + c]$ since B is fully sorted, and $0 \le c \le \frac{n}{2}$ then $c \dots \frac{n}{2} - 1 + c$ is consecutive which means $A[0 \dots (n/2-1)]$ is fully sorted.

the value of c determines which part is sorted. if $0 \le c \le \frac{n}{2}$ then the first part is sorted. if $\frac{n}{2} \le c \le n$ the second part is fully sorted.

if $c = 0$ or $c = \frac{n}{2}$ then they will all be sorted. □

(b) (8 points) Assume again that A is rotation-sorted, but you are not given the cyclic shift c. Design a divide-and-conquer algorithm to compute the minimum of A (i.e., $B[0]$). Carefully prove the correctness of your algorithm, write the recurrence equation for its running time, and solve it. Is it better than the trivial $O(n)$ algorithm? (**Hint**: Be careful with $c = 0$ an $c = n/2$; you might need to handle them separately.)

Solution: algorithm:

[1] Intuitively, A is either completely sorted (if $c = 0$), or (if $c > 0$) A starts in sorted order, but then "falls off the cliff" when going from $A[n-c-1] = B[n-1] = \max$ to $A[n-c] = B[0] = \min$, and then again goes in increasing order while never reaching $A[0]$.

```
1     public int Min(int start,int end, int[] a)
2     {
3         int n=a.length;
4         if(a[0]<a[n-1]) return a[0];
5         if(a[n/2-1]>a[n/2]) return a[n/2];
6         if(a[start]<a[end]) return a[start];
7         if(a[start]>a[(start+end)/2-1]) return Min(start,(start+end)/2-1,a);
8         else return Min((start+end)/2,end,a);
9     }
```

$$T(n) = T(\frac{n}{2}) + O(1) = O(logn)$$

it is better than the trivial $O(n)$ algorithm

CSCI-GA.1170-001/002 Fundamental Algorithms September 29, 2019

Solutions to Problem 4 of Homework 3 (18 points)

Name: jingshuai jiang (jj2903) *Due: 5pm on Thursday, September 26*

Collaborators: NetID1, NetID2

A *local minimum* of a two-dimensional array $\{A[i,j] \mid 1 \le i,j \le n\}$ is an index $(i,j) \in \{1,\ldots,n\} \times \{1,\ldots,n\}$ which is less or equal than all of its neighbors, where we say that two nodes are neighboring if they are either vertically or horizontally (but not diagonally) adjacent in the array. Note that every array has at least (and possibly more than) one local minimum, since the "global" minimum of the entire array is also a local minimum. The eventual goal of this problem is to design an efficient divide-and-conquer algorithm to find some local minimum of a given (unsorted) array A of size $n \times n$.

(a) (2 points) Consider the following "greedy" algorithm. Start with any node $v = (i,j)$. If v is a local minimum, then output v. Else take the smallest neighbor of v (break ties arbitrarily) and repeat the above process with the neighbor until you find a local minimum. Prove that this algorithm always terminates in time $O(n^2)$.

Solution: INSERT YOUR SOLUTION HERE □

(b) (3 points) What is the exact length (number of nodes, not "edges") of the "local minimum path" of the greedy algorithm on the following 7×7 grid, starting with the initial point $v = (1,1)$ (equal to 30). By generalizing this picture from $n = 7$ to general n, show that the worst case running time of the greedy algorithm is $\Omega(n^2)$.

$$\begin{pmatrix} 30 & 100 & 16 & 15 & 14 & 100 & 0 \\ 29 & 100 & 17 & 100 & 13 & 100 & 1 \\ 28 & 100 & 18 & 100 & 12 & 100 & 2 \\ 27 & 100 & 19 & 100 & 11 & 100 & 3 \\ 26 & 100 & 20 & 100 & 10 & 100 & 4 \\ 25 & 100 & 21 & 100 & 9 & 100 & 5 \\ 24 & 23 & 22 & 100 & 8 & 7 & 6 \end{pmatrix}$$

Solution:

the length is 31

$$\begin{pmatrix} bignumber & \infty & 16 & 15 & 14 & \to & \infty & smallnumber \\ bignumber-1 & \infty & 17 & \infty & 13 & \to & \infty & smallnumber+1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \to & \downarrow & \downarrow \\ 28 & \infty & 18 & \infty & 12 & \to & \infty & 2 \\ 27 & \infty & 19 & \infty & 11 & \to & \infty & 3 \\ 26 & \infty & 20 & \infty & 10 & \to & \infty & 4 \\ 25 & \infty & 21 & \infty & 9 & \to & \infty & 5 \\ 24 & 23 & 22 & \infty & 8 & \to & 7 & 6 \end{pmatrix}$$

the symbol ∞ here represents some very big numbers larger than the others and the numbers outside the symbol ∞ is just some numbers that help we explain.

by arranging the matrix like this we can get that if n is an odd,then the time will be

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

if n is an even ,then the time will be

$$\frac{n(n+1)}{2}$$

by showing the worst case ,it can be conclude that their running time is $\Omega(n^2)$. □

(c) (3 points) For simplicity of calculation, assume that $n = 2^k - 1$ for some integer $k \geq 1$. Consider the following divide-and-conquer algorithm: at every step, find the minimum element v of the middle row 2^{k-1} and the middle column 2^{k-1}. If v is a local minimum, output v. Else take the smallest neighbor of v (call it w), and recurse in the quadrant (north-east, north-west, south-east or south-west) of A of size $(n-1)/2 = (2^{k-1} - 1)$ where w lies (not counting the middle row/column in the quadrant, so one row/column is eliminated before dividing by 2).

While this algorithm looks appealing, you will prove that it is *not* correct in general. For this, consider the following grid containing all the numbers from 1 to 49 exactly once, except three weights $10, 31, 39$ are marked with the ?. Fill the missing numbers marked with ? with $10, 31, 39$ in a way such that the algorithm given above gives the wrong answer, and state what this wrong answer is.

$$\begin{pmatrix} 45 & 48 & ? & 20 & ? & ? & 32 \\ 42 & 41 & 47 & 30 & 36 & 34 & 37 \\ 46 & 43 & 44 & 40 & 38 & 33 & 35 \\ 21 & 22 & 23 & 24 & 25 & 26 & 49 \\ 2 & 3 & 9 & 27 & 13 & 12 & 16 \\ 4 & 1 & 8 & 28 & 11 & 15 & 19 \\ 5 & 6 & 7 & 29 & 14 & 17 & 18 \end{pmatrix}$$

Solution:

$$\begin{pmatrix} 45 & 48 & 31 & 20 & 10 & 39 & 32 \\ 42 & 41 & 47 & 30 & 36 & 34 & 37 \\ 46 & 43 & 44 & 40 & 38 & 33 & 35 \\ 21 & 22 & 23 & 24 & 25 & 26 & 49 \\ 2 & 3 & 9 & 27 & 13 & 12 & 16 \\ 4 & 1 & 8 & 28 & 11 & 15 & 19 \\ 5 & 6 & 7 & 29 & 14 & 17 & 18 \end{pmatrix}$$

□

(d) (8 points) For simplicity of calculation, assume that $n = 2^k + 1$ for some integer $k \geq 0$ and all the numbers $A[i, j]$ are distinct. Consider a slightly more complex divide-and-conquer algorithm. In addition to the middle row $(2^{k-1}+1)$ and column $(2^{k-1}+1)$, also look at the boundary nodes $\{(1, i), (n, i), (i, 1), (i, n) \mid i = 1 \ldots n\}$, and find the global minimum a (located at node v of all these $6n-9$ numbers (3 rows and 3 columns of size n, except 9 "intersection points" are counted twice) in $O(n)$ time. If v is a local minimum, output v. Else take the smallest neighbor of v (call it w), and recurse in the quadrant (north-east, north-west, south-east or south-west) of A of size $(n+1)/2 = 2^{k-1}+1$ (where the quadrant *includes the boundary*, so you effectively duplicate the middle row/colum before dividing by 2) where w lies.

Prove the correctness of this modified algorithm. To do that, prove the following stronger inductive statement: the algorithm computes an answer which is (a) correct local minimum (i, j) *and* (b) $A[i, j]$ is less or equal than every number on the boundary of the square. Make sure you stress where (i) you use the fact that a *smaller* neighbor w is selected; (ii) that all the numbers are *distinct*.

Solution: INSERT YOUR SOLUTION HERE □

(e) (2 points) Write the recurrence equation and solve it to compute the running time of the algorithm in part (d).

Solution:

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

$$T(n) = T(\frac{n}{4}) + c\frac{n}{2} + cn$$

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

□