Assignment 4

Unmarked Questions

Remember to comment out this section when submitting your assignment.

Here are a few simple warm-up problems. Make sure you are able to do them before proceeding to the marked questions.

Q1. A Recursively Defined Function

Consider the function

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n = 1, \\ 2f(n-2) & \text{if } n \geq 2 \end{cases}$$

Prove that the closed form of f is $f(n) = 0$ if n is odd and $f(n) = 2^{n/2}$ if n is even.

Proof. Note that this recurrence relation is not amenable to the Master Method so we will use the standard "guess-and-check" approach. Luckily, the question already proposes a "guess" so it suffices for us to "check" that it is correct.

1. **Predicate.** Let $P(n)$ be the predicate: $f(n) = 0$ if n is odd and $f(n) = 2^{n/2}$ if n is even. We want to show that $P(n)$ is true for all $n \in \mathbb{N}$.
2. **Base Case.** Note that for $n = 0$ which is even, $f(0) = 1$ and $f(1) = 0$ by definition which is equal to $2^{0/2}$ and 0 respectively.
3. **Inductive Step.** Suppose that $k \geq 1$ and that $P(0) \wedge \cdots \wedge P(k)$ is true. We want to show that $P(k+1)$ is true. There are two cases to consider, either $k+1$ is odd or even. Suppose that $k+1$ is odd. Then, $k-1$ is also odd and by IH, we must have $f(k-1) = 0$. Thus $f(k+1) = 2f(k-1) = 0$ as required.

   Otherwise $k+1$ is even and $k-1$ is even as well. By IH, we have that $f(k-1) = 2^{(k-1)/2}$ it follows that $f(k+1) = 2f(k-1) = 2 \cdot 2^{(k-1)/2} = 2^{(k+1)/2}$ as required.

Thus, by the Principle of Structural Induction, $P(n)$ is true for all $n \in \mathbb{N}$. □

Q2. k^{th} Minimum

The goal of this problem is to show that Algorithm 1 finds the k^{th} *order statistic* of a list A. Order statistics are a generalization of the notions of minimum and maximum. The minimum value of A is the 1^{st} order statistic while the maximum is the n^{th} order statistic where $|A| = n$.

Note: For simplicity, A will be *one*-indexed.

a. Prove that Algorithm 1 called on $\mathsf{mink}(A, 1, n, k)$ returns the k^{th} order statistic of A.

Algorithm 1 mink(A, i, j, k)

```
Require: $A$ is a one-indexed list and $k$ is a positive integer satisfying $k \leq j - i + 1$.
Ensure: Returns $k$TH smallest element of $A[i, j]$ inclusive.
1: if $i = j$ then return $A[i]$
2: end if
3: $p \leftarrow$ partition$(A, i, j + 1)$        ▷ This is the partition algorithm we saw in-class
4: if $p - i + 1 = k$ then return $A[p]$
5: else if $p - i + 1 > k$ then return mink$(A, i, p - 1, k)$
6: else return mink$(A, p + 1, j, k - p)$
7: end if
```

Proof. Our predicate will take as input a subsequence $A[i, j]$ of A containing all entries of A from index i to j where $i \leq j$. $P(i, j, k)$ will be: mink(A, i, j, k) returns the k^{th} order statistic of $A[i, j]$.

In the base case $i = j$ and $k = 1$. Line 1 returns the 1^{st}, and only, element in $A[i, j]$.

For the inductive step assume that $i < j$ and we apply partition to the subsequence $A[i, j]$ with the pivot being the entry at $A[j]$. By the post-condition of partition, we know that the index of the pivot p return will separate $A[i, j]$ into two parts: for all indices u such that $i \leq u < p$, we have that $A[u] \leq A[p]$ and for all indices v such that $p < v \leq j$, we have that $A[v] \geq A[p]$. Note that if $p - i + 1 = k$, then the algorithm returns the correct value in line 3. Otherwise if $p - i + 1 \neq k$, then it could be greater or less than k. If it is greater than k then we know that the k^{th} order statistic of $A[i, j]$ is the k^{th} order statistic of $A[i, p - 1]$. By IH, this is exactly what mink$(A, i, p - 1, k)$ returns. Conversely, if $p - i + 1 < k$, then the k^{th} order statistic of A is in $A[p + 1, j]$ and is in-fact the $k - p^{\text{th}}$ order statistic in this interval (e.g. if $i = 1$, $j = 10$, $p = 3$, and $k = 5$, then we want to look for the 2^{nd} order statistic in the interval [4..10]). Again, by IH, this is exactly what mink$(A, p + 1, j, k - p)$ gives us.

Since the post-condition of the algorithm is satisfied, when we call mink$(A, 1, n, k)$, we will obtain the k^{th} order statistic of k as required. □

b. Analyse the worse-case running time of this algorithm if partition always returns the middle index of the input list.

Solution. Let $T(n)$ be the running time of Algorithm 1 on a list A of length n. We observe that the running time can be recursively defined as constant for $T(1)$ and for $n > 1$,

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

Using the Master Method, we see that this translates into a running time of $\Theta(n)$ as $a = 1$, $b = 2$ and the work being done during an iteration is on the order of n^1.

Marked Questions

Q1. [10 Points] Multiplying Upper-Triangular Matrices *(maximum 3 pages)*

In class, we saw Karatsuba's algorithm which was a way to speed up the computation of $a \cdot b$ for two n-bit integers a and b. In this problem, we analyse an algorithm for multiplying two *upper-triangular*[1] matrices abbreviated *UT matrices.* See Algorithm 2. The standard approach for multiplying two matrices is shown below for two 2×2 matrices.

$$A \cdot B = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{bmatrix}$$

For ease of explanation, if the inputs A and B to Algorithm 2, Algorithm 3, and Algorithm 4 have dimension $n \times n$, then let n be a power of two. In practice, we can always pad the matrix with rows and columns of all zeros so that this is the case.

Note that the notation $A[i : j, \ell : k]$ represents the submatrix of A consisting of the rows i to j inclusive and columns ℓ to k inclusive. For A and B, let $A_{11} = A[1 : n/2, 1 : n/2]$, $A_{12} = A[1 : n/2, (n/2 + 1) : n]$, $A_{21} = A[(n/2 + 1) : n, 1 : n/2]$, and $A_{22} = A[(n/2 + 1) : n, (n/2 + 1) : n]$. B_{ij} are defined similarly for $i, j \in [2]$.

The time it takes to multiple two constant dimension matrices is constant time.

Algorithm 2 multiplyTT(A : UT matrix, B : UT matrix) $\to$ UT matrix

```
Require: A and B are two upper-triangular matrices.
Ensure: Outputs $A \cdot B$.
1: if $n = 1$ then return $A \cdot B$
2: end if
3: $M_1 \leftarrow$ multiplyTT($A_{11}, B_{11}$)
4: $M_2 \leftarrow$ multiplyTM($A_{11}, B_{12}$) + multiplyMT($A_{12}, B_{22}$)
5: $M_3 \leftarrow$ multiplyTT($A_{22}, B_{22}$)
6: return $\begin{bmatrix} M_1 & M_2 \\ 0 & M_3 \end{bmatrix}$
```

Algorithm 3 multiplyTM(A : UT matrix, B : matrix) $\to$ matrix

```
Require: A is an upper-triangular matrix and B is a matrix.
Ensure: Outputs $A \cdot B$.
1: if $n = 1$ then return $A \cdot B$
2: end if
3: $M_1 \leftarrow$ multiplyTM($A_{11}, B_{11}$) + multiply($A_{12}, B_{21}$)
4: $M_2 \leftarrow$ multiplyTM($A_{11}, B_{12}$) + multiply($A_{12}, B_{22}$)
5: $M_3 \leftarrow$ multiplyTM($A_{22}, B_{21}$)
6: $M_4 \leftarrow$ multiplyTM($A_{22}, B_{22}$)
7: return $\begin{bmatrix} M_1 & M_2 \\ M_3 & M_4 \end{bmatrix}$
```

a. Suppose that the time it takes to multiply two numbers or add two numbers is $O(1)$. What

[1]A square matrix A is *upper-triangular* if all entries $a_{i,j}$ are equal to zero for $i > j$ (the top left entry of A is $a_{1,1}$ the bottom right entry is $a_{n,n}$). Here are some upper-triangular matrices:

$$A_1 = \begin{bmatrix} 5 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 5 & 1 \\ 0 & 2 \end{bmatrix} \qquad A_3 = \begin{bmatrix} 0 & 3 & 1 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Algorithm 4 multiplyMT(A : matrix, B : TU matrix) $\to$ matrix

Require: A is a matrix and B is an upper-triangular matrix.
Ensure: Outputs $A \cdot B$.

```
1: if $n = 1$ then return $A \cdot B$
2: end if
3: $M_1 \leftarrow$ multiplyMT($A_{11}, B_{11}$)
4: $M_2 \leftarrow$ multiply($A_{11}, B_{12}$) + multiplyMT($A_{12}, B_{22}$)
5: $M_3 \leftarrow$ multiplyMT($A_{21}, B_{11}$)
6: $M_4 \leftarrow$ multiply($A_{21}, B_{12}$) + multiplyMT($A_{22}, B_{22}$)
7: return $\begin{bmatrix} M_1 & M_2 \\ M_3 & M_4 \end{bmatrix}$
```

is the asymptotic running time of the standard algorithm for multiplying two n by n upper-triangular matrices?

Solution.

b. Prove the correctness of Algorithm 2 assuming the correctness of multiply (it takes as input two $n \times n$ matrices A and B and the post-conditions is that it correctly returns $A \cdot B$). You only need to prove *one* of Algorithm 3 and Algorithm 4 correct, but not both.

Proof. □

c. Read about Strassen's algorithm. Compute the asymptotic running time of Algorithm 2 assuming that multiply on lines 3 and 4 of Algorithm 3 and lines 4 and 5 of Algorithm 4 calls Strassen's algorithm.

Solution.

Q2. [10 Points] Candy Claw Machine *(maximum 3 pages)*

You are playing a crane game to win as many tasty candies as possible. There are n candies in a row on a conveyor belt and you assigned a value $v_i \in \mathbb{R}$ to the candy at position i (note the value can be negative). At each time unit the conveyor belt will move to the left by one candy. You can lower the claw at time t and retract the claw at time s where $0 \leq t \leq s \leq n$ (when $t = s = n$, you will not get any candies). *You can only lower and raise the claw once.*

You *collect* all candies at positions t through s inclusive (e.g. if $t = 0$ and $s = n - 1$ then you collect all the candies). The *value* of the candies you collect is $\sum_{i=t}^{s} v_i$. Come up with a *recursive* algorithm to determine when you should lower and raise the claw to collect the most valuable candies. If there are multiple time intervals obtaining the maximum value, you can output any.

Your algorithm should run in $O(n)$ time. If it runs in $O(n \log n)$ and you can prove this, you will get part marks.

As an example, suppose that you have the candies on the conveyor belt have the following values:

$$[1, 2, -9, 3, 3, 1, -2, 1]$$

Then the most valuable candies can be obtained by lowering the claw at time 3 (picking up the first three value candy) and raising it at time 5 (picking up the one value candy). The total value of this collection is 7.

If all the candies have negative value, as is the case for this conveyor belt:

$$[-1, -4, -2, -2, -3],$$

you can lower and raise the claw at time 5 to collect no candies.

a. Write the pseudo-code in Algorithm 5.

Algorithm 5 Claw(L)

```
Require: Fill-in the precondition.
Ensure: Fill-in the postcondition.
 1: $u \leftarrow 1$                          ▷ this is how you assign variables
 2: for $i \in [n]$ do                        ▷ this is a for-loop
 3:     while $i > 0$ do                      ▷ this is a while-loop
 4:         if $i < 10$ then                  ▷ this is an if-else-statement
 5:             $u \leftarrow 2$
 6:         else
 7:             $u \leftarrow 3$
 8:         end if
 9:     end while
10: end for
11: return $A, B$                             ▷ this how you return something
```

b. Prove the correctness of your algorithm. Remember to *state the post-condition.*

Proof. □

c. Evaluate the running time of your algorithm and justify your answer.

Solution.

Additional Questions

Remember to comment out this section when submitting your assignment.

If you would like more exercises consider trying the following problems from your primary and supplementary textbooks. *We will not be providing solutions to these questions* though you are free to find the solution online and discuss them with your peers.

1. David Liu's notes Chapter 3: Exercises 8, 9, 10, 11, 13.
2. David Liu's notes Chapter 4: Exercises 1