Elliot F. Poirier
January 19th 2022

McGill University
Honours Algorithm and Data Structures

Assignment 1

Exercise 1. A QUESTION RELATED TO FIBONACCI SEQUENCES. A Fibonacci-like sequence of integers is defined as follows: $x_0 = x_1 = 0, x_2 = 1, x_n = 3x_{n-1} - 6x_{n-2} + 4x_{n-3}, n \geq 3$.

(i) Show that $|x_n| = \Theta(c^n)$ for some positive constant c and determine c.

Proof. We will get the characteristic polynomial of the recurrence relation.

$$\begin{aligned} 0 &= 3x_{n-1} - 6x_{n-2} + 4x_{n-3} - x_n \\ &= x^3 - 3x^2 + 6x - 4 \\ &= (x-1)(x^2 - 2x + 4) \\ &= (x-1)(x - 1 - \sqrt{3}i)(x - 1 + \sqrt{3}i) \end{aligned} \tag{1}$$

So the roots are 1 and $1 \pm \sqrt{3}i$. And from this we get the equation,

$$\begin{aligned} x_n &= 0 \cdot 1^n + 0 \cdot (1 - \sqrt{3}i) + 1 \cdot ((1 + \sqrt{3}i))^n \\ &= ((1 + \sqrt{3}i))^n \end{aligned} \tag{2}$$

Thus, we have that $|x_n| = \Theta\left((1 + \sqrt{3}i)^n\right)$. ■

(ii) Give an algorithm that, under the Ram model, computes x_n in time $O(\log n)$.

Proof. From the recurrence equation, we will build a matrix of the form

$$\begin{aligned}
\begin{bmatrix} x_n \\ x_{n-1} \\ x_{n-2} \end{bmatrix} &= A \begin{bmatrix} x_{n-1} \\ x_{n-2} \\ x_{n-3} \end{bmatrix} \\
&= \begin{bmatrix} 3 & -6 & 4 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{n-1} \\ x_{n-2} \\ x_{n-3} \end{bmatrix} \qquad (3) \\
\begin{bmatrix} x_n \\ x_{n-1} \\ x_{n-2} \end{bmatrix} &= \begin{bmatrix} 3 & -6 & 4 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\end{aligned}$$

Now, we proceed with a similar algorithm of that we used to compute the fibonacci numbers (Algo 5). We recursively keep squaring the matrix we've squared A $\lfloor \log_2(n-2) \rfloor$ times. Following this, we'll be left with, which is a constant operation.

$$\begin{aligned}
\begin{bmatrix} x_n \\ x_{n-1} \\ x_{n-2} \end{bmatrix} &= A^{n-2} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\
&= A^{2^{2^{\cdots}}} A^{(n-2)-\lfloor \log_2(n-2) \rfloor} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad (4)
\end{aligned}$$

So, our equation for time is

$$\begin{aligned}
T(n) &= O(1) + T(n/2) \\
&= O(1) + \log(n) \cdot O(1) = O(\log n) \qquad (5)
\end{aligned}$$

Thus, our algorithm has $O(\log n)$. ■

Exercise 2. Bit model versus ram model. Let us define the Nanami function $N(n)$ for positive integer argument n as the smallest nonnegative integer i such that $\log_7$ of n, repeated i times, is ≤ 1, i.e.,

$$\overbrace{\log_7 \log_7 \cdots \log_7}^{i \text{ times}}(n) \leq 1.$$

So, $N(1) = 0$, and $N(2) = \cdots = N(7) = 1, N(8) = \cdots = N(49) = 2$, and so forth.

(i) Find a good algorithm for computing $N(n)$ in the RAM model and determine its complexity in $O(\cdot)$ notation.

Proof. We can see that

$$\begin{aligned} &N(n) = 0 \quad n \in \{1\} = \{7^0\} \\ &N(n) = 1 \quad n \in \{2, \cdots, 7\} = \{7^0 + 1, \cdots, 7^1\} \\ &N(n) = 2 \quad n \in \{7^1 + 1, \cdots, 7^7\} \\ &N(n) = 3 \quad n \in \{7^7 + 1, \cdots, 7^{7^7}\} \cdots \end{aligned} \tag{6}$$

by observing logarithm rules such that,

$$\begin{aligned} log_7(\cdots log_7(log_7(log_7(7^{7^{7^{\cdots}}}))) &= log_7(\cdots log_7(log_7(7^{7^{\cdots}} log_7(7)))) \\ &= log_7(\cdots 7^{\cdots} log_7(7 log_7(7 log_7 7)))) \\ &= 1 \end{aligned} \tag{7}$$

So this is the algorithm I came up with, written in python:

```
def NanamiFunction(n):
    exponent = 0
    while (n > 1):
        n = math.ceil(n/7)
        exponent+ = 1
    if (exponent> 7):
        return NanamiFunction(exponent) + 1
    if (exponent> 2):
        return 2
    return exponent
```

We know that everytime the function is called, the loop is performed $k = \min\{k \in \mathbb{N} : n \leq 7^k\} = \lceil log_7(n) \rceil$ which is the important part since the rest is constants. And since I call the function again after looping we have (there are two operations in the loop);

$$\begin{aligned} T(n) &= 2 + 2\lceil log_7(n) \rceil + T(\lceil log_7(n) \rceil) \\ &\leq 4 + 4\lceil log_7(n) \rceil \end{aligned} \tag{8}$$

Thus, my algorithm has $O(log_7(n))$ in the RAM model. ■

(ii) Find a good algorithm for computing $N(n)$ in the bit model and determine its complexity in $O(\cdot)$ notation.

Proof. Now, the key difference is that in the bit model the computation of " n = math.ceil(n/7) " will take $log_2(n)$, so we would get:

$$\begin{aligned} T(n) &\leq 2 + log_7(n) + \sum_{k=0}^{\lceil log_7(n) \rceil} \lceil log_2 \lceil \frac{n}{7^k} \rceil \rceil + T(\lceil log_7(n) \rceil) \\ &\leq 2 + \lceil log_7(n) \rceil + \lceil log_7(n) \rceil \lceil log_2 n \rceil + T(\lceil log_7(n) \rceil) \\ &\leq 4 + 4 \lceil log_7(n) \rceil \lceil log_2 n \rceil \end{aligned} \tag{9}$$

Thus, the complexity in the bit model is $O(log_2(n) log_7(n))$. ■

Exercise 3. Algorithm design: Finding the maximal POINTS. Consider n points $(x_i, y_i), 1 \leq i \leq n$, with strictly positive coordinates such that no two coordinates coincide. We call a point (x_i, y_i) maximal if there does not exist a j such that $x_j \geq x_i$ and $y_j > y_i$. Assuming a RAM model of computation, give an $O(n \log n)$ divide-and-conquer algorithm for solving this problem.

Proof. Let $S = \{(x_i, y_i) | 1 \leq i \leq n\}$ with strictly positive coordinates such that no two coordinates coincide. Now, let's create two lists, one with sorted with x_i, X; one sorted with y_i, Y. This will take $O(n \log_2 n)$ twice.

Then go through X and perform binary search for the given (x_i, y_i) in the subset of Y, $Y' = Y \setminus \{(x_j, y_j) : x_i \geq x_j\}$ and we check to see if it is the greatest element in Y', if it is, then it is a maximal element, otherwise it isn't. Binary search takes $O(\log_2 k)$ for $|Y'| = k$ and we perform it $|X| = n$ times, so $O(n \log_2 k)$. Therefore,

$$\begin{aligned} T(n) &\leq 2(n \log_2 n) + n(\log_2 k) \\ &\leq 3(n \log_2 n) \end{aligned} \tag{10}$$

Thus, we have a complexity of $O(n \log_2 n)$. ■

Exercise 4. Recursive algorithm Design. We are given a collection of n integers in binary format (each number being given as a bit vector). These integers are a permutation of the set $A = \{0, 1, 2, \ldots, n+ 1\}$ after two integers, i and j, are removed. We do not know i and j. We have the following oracle at our disposal: we can ask the oracle for the k-th bit (from the right) of the ℓ-th number for any k and ℓ. The objective is to find i and j in $O(n)$ time, where time is measured in terms of how often the oracle is used. Give an algorithm and argue why it takes $O(n)$ time.

Proof. We know that all integers in A fit in binary strings of size $\lceil log_2(n + 1)\rceil$, so we go over all strings in A', our collection of n integers and we ask the oracle about the first two digits (from the right) of all the numbers $(2n)$. This will divide the set in four (10,01,00,11), then either two sets are missing 1 or one is missing 2 compared to their respective counterparts in A. We then repeat the process for the next two digits with the set or sets that are missing elements. So we at most have half the input size at every subsequent iteration. Hence,

$$\begin{aligned} T(n) &\leq 2n + T(n/2) \\ &= 2n + n + n/2 + n/4 + \cdots \\ &\leq 2n \sum_{k=0}^{\frac{\lceil \log_2 n \rceil}{2}} \frac{1}{2^k} \\ &\leq 4n \end{aligned} \tag{11}$$

Thus, we can see that this algorithm has $O(n)$ time. ■