Algorithms and Data Structures
MSIS 26:198:685
Homework 2

Instructor: Farid Alizadeh
Due Date: Monday October 21, 2019, at 11:50PM

last updated on November 26, 2019

Please answer the following questions in **electronic** form and upload and submit your files to Sakai site, before the due date. Make sure to push the **submit** button.

You can typeset your answer using MS Word (and MS equation for mathematical formulas), or LaTeX, or you may simply handwrite your note and scan and turn it not a **single pdf format file** and upload to Sakai.

26:198:685, Fall 2019

Homework 2 **Due date:** 10/21/19

1. Solve the following Recurrence relations. Show your work. Express the solution in terms of big Θ of familiar functions. You may use the Mater Method, if applicable.

1a) $F(n) = 3F(n/3) + n^2$, with $F(1) = 1$

Answer: By the Master Theorem, since $g(n) = n^2 = \Omega\big(n^{\log_3(3)}\big)$ it follows that $f(n) = \mathcal{O}(n^2)$. We could also derive it:

Set $n = 3^k$, so $n^2 = 3^{2k}$. Now define $g(k) = F(3^k)$, and so $g(0) = 1$. Then we have

$$\begin{aligned} g(k) &= 3g(k-1) + 3^{2k} \\ &= 3\big(3g(k-2) + 3^{2(k-1)}\big) + 3^{2k} = 3^2 g(k-2) + 3^{2k-1} + 3^{2k} \\ &= 3^2\big(3g(k-3) + 3^{2(k-2)}\big) + 3^{2k-1} + 3^{2k} = 3^3 g(k-3) + 3^{2k-2} + 3^{2k-1} + 3^{2k} \\ &= \vdots \\ & 3^k g(0) + 3^{k+1} + 3^{k+2} + \cdots + 3^{2k} = 3^k + 3^{k+1} + 3^{k+2} + \cdots + 3^{2k} \end{aligned}$$

The above sum is a geometric progression, starting with 2^k and each term multiplied by 3. So based on the formula foe geometric progression we have

$$g(k) = 3^k \frac{3^{k+1} - 1}{3 - 1} = \frac{3^{2k+1} - 3^{2k}}{2}$$

Now replacing for $g(k) = F(3^k) = F(n)$, we get

$$\begin{aligned} F(n) &= \frac{1}{2}\big(3 \times (3^k)^2 - (3^k)^2\big) \\ &= \frac{1}{2}(3n^2 - n^2) = n^2 \end{aligned}$$

1b) $G(n) = 2G\big(\sqrt{n}\big) + \log(n)$, with $G(1) = 1$

Answer: (We should assume $G(2)$ =constant, if we set it equal to 1, we get into trouble.) We set $n = 2^k$ and define $H(k)$ so that $G(2^k) = H(k)$. Then $\sqrt{n} = 2^{k/2}$. So the recursion in terms of $H(\cdot)$ as

$$H(k) = 2H(k/2) + k \quad H(1) = \text{constant}$$

Using the Master Theorem since $g(k) = k = \Theta\big(k^{\log_2(2)}\big)$, we have $H(k) = \mathcal{O}(k \log k)$. Substituting back we get $G(n) = H(\big(\log(n)\big) = \mathcal{O}\big(\log(n) \log\log(n)\big)$.

1c) $H(n) = \frac{H(n-1) + H(n-2) + \cdots + H(1)}{n} + 2$, with $H(1) = 1$

Answer: We get

$$nH(n) = H(n-1) + \cdots + H(1) + 2n$$

And thus,

$$(n-1)H(n-1) = H(n-2) + \cdots + H(1) + 2(n-1)$$

Subtracting, we get

$$nH(n) - (n-1)H(n-1) = H(n-1) + 2n - 2(n-1)$$

which implies

$$nH(n) = nH(n-1) + 2 \quad \Rightarrow \quad H(n) = H(n-1) + \frac{2}{n}$$

From here it is clear that this function is twice the harmonic function.

$$H(n) = 2\Big(1 + \frac{1}{2} + \cdots + \frac{1}{n}\Big) = \mathcal{O}\big(\big(\log(n)\big)$$

1d) $f(n) = 4f(n/2) + \frac{n^2}{\log(n)}$.

Note that the Master Theorem does not apply here since $g(n) = \frac{n^2}{\log(n)} \neq n^{\log_2(4)-\epsilon}$ for any value of ϵ. So we use the direct method of change of variables and expansion. Setting $n = 2^k$, and thus, $k = \log(n)$, we have:

$$\begin{aligned} g(k) &= 4g(k-1) + \tfrac{2^{2k}}{k} \\ &= 4\big(4g(k-2) + \tfrac{2^{2k-2}}{k-1}\big) + \tfrac{2^{2k}}{k} = 4^2 g(k-2) + \tfrac{2^{2k}}{k-1} + \tfrac{2^{2k}}{k} \\ &= \ldots \\ &= 4^k C + 2^{2k}\Big(\tfrac{1}{k} + \tfrac{1}{k-1} + \cdots + 1\Big) \end{aligned}$$

So $H(n) = g(2^k) = C4^k + 2^{2k}\Big(1 + \frac{1}{2} + \cdots + \frac{1}{k}\Big)$. Replacing $k = \log(n)$ and noting that $1 + \frac{1}{2} + \cdots + \frac{1}{k} = \mathcal{O}(\log k) = \mathcal{O}(\log\log n)$ we get

$$H(n) = n^2 \log\log(n)$$

2. Recall that in boolean algebra we only have the two values of 1 (also known as `True`) and 0 (also known a `False`.) The two main operations on boolean values are product '$\cdot$' (or `And`) and sum '+' (or `Or`) defined as follows:

$$a \cdot b = 1 \text{ if both } a = 1 \text{ and } b = 1, \text{ otherwise } a \cdot b = 0$$
$$a + b = 1 \text{ if at least one of } a \text{ or } b \text{ is 1, otherwise } a + b = 0.$$

With these definitions we can define the *boolean matrix product* $AB = C$ where

$$C_{ij} = A_{i1} \cdot B_{1j} + \cdots + A_{in} \cdot B_{nj}.$$

Consider two $n \times n$ matrices A and B with boolean, that is only 0 or 1 entries. Define the *boolean product* of these two matrices the same way that ordinary numerical matrices are multiplied, except that instead of multiply we use the above boolean product, and instead of plus, we use the above boolean sum.

2a) To compute the boolean matrix product AB, explain why the Strassen's algorithm cannot be used directly on the boolean matrices using only the boolean operations.

If we wish to mimic the Strassen's method directly, we have to find an analog of "subtraction", inverse of the "or" operation. But there is no such inverse. The "boolean equation $x + a = 1$ has two solutions if $a = 1$. So there is no inverse and no subtraction.

2b) Interpret 0's and 1's as *integers*, and use ordinary multiplication. How can you extract boolean multiplication from the result of ordinary integer multiplication? Strassen's algorithm for this case?

We interpret values of a boolean matrix as a *numerical* matrix. So for "or" we use ordinary addition, and for "and" we use ordinary multiplication. The result of matrix product becomes a numerical matrix with the largest number in the matrix not exceeding n. Now we interpret every zero element in the product as False or logical 0, and every non-zero entry as True or logical 1.

2c) Arithmetic complexity is not appropriate here because the original input is a matrix of bits. Give a complete *bit* complexity analysis. Assume in part a) you used a Strassen-like algorithm with arithmetic complexity $O(n^{2+\omega})$ where $0 < \omega < 1$.

The number of arithmetic operations, using Strassen-like algorithms, requires $O\big(n^{2+\omega}\big)$. Each operation involves multiplying at most n-bit numbers. Since each n-bit integer multiplication costs $O(n^2)$ (or $O\big(n \log(n) \log\log(n)\big)$ if we use FFT) the total number of bit operations is $O(n^{2+\omega} \times n^2) = O\big(n^{4+\omega}\big)$.

3. A list of numbers is *unimodal* if it starts with a sequence of *non-decreasing* numbers followed by a sequence of *non-increasing numbers*. Each subsequence can be empty. For instance, the Following sequences are unimodal: (1, 2, 7, 6, 4), (1, 1, 2, 2, 5,4), (2,2,2,2), (4,3,2,1). However, the following sequences are *not* unimodal: (1,2,3,2,4), (1,1,2,2,1,2).

   Given an unimodal sequence, give the most efficient algorithm you can to find its maximum element, both its value, and its index in the list. If there are more than one occurrence of the maximum, return the smallest index in which it occurs. Provide a complete complexity analysis, deriving any difference equation needed. Solve the difference equation you derived to find the time complexity of your algorithm.

Suppose we split the list in the middle: $(a_1, \dots, a_m)$ and $(a_{m+1}, \dots, a_n)$, where $m = \lfloor \frac{n}{2} \rfloor$. If the maximum is in the first (left) group then $a_{m+1} \leq a_m$. If the maximum is in the second (right) group then $a_{m+1} \geq a_m$. So if $a_m > a_{m+1}$ we can recursively search the left list only. If $a_m < a_{m+1}$ we can recursively search the right sequences. The difficulty comes when $a_m = a_{m+1}$. In that case the maximum can be in either sequence. So we get two results: If the sequence is *strongly* unimodal, that is $(a_1 < \dots < a_{max} > \dots > a_n)$, the n the following recursive algorithm will work:

```
Algorithm: StrongUnimodal($a_{low}, \dots, a_{high}$)
    if low == high
        return($a_{low}$)
    set mid = $\left\lfloor \frac{low+high}{2} \right\rfloor$
    if $a_{mid} > a_{mid+1}$
        return StrongUnimodal($a_{mid+1}, \dots, a_{high}$)
    if $a_{mid} < a_{mid+1}$
        return StrongUnimodal($a_{low}, \dots, a_{mid-1}$)
    else
        return max[ StrongUnimodal($a_{mid+1}, \dots, a_{high}$), StrongUnimodal($a_{low}, \dots, a_{mid-1}$), $a_{mid}$]
```

The complexity analysis follows the recurrence relation $f(n) = 2f(n/2) + c$, which, using the Master Theorem, gives us $f(n) = \mathcal{O}(n)$. The worst case happens if the sequence is made of equal numbers, say $(2, 2, \dots, 2)$.
If all numbers are distinct and there is no equality, then the last part of the algorithm (**else**) is not needed. In that case the recurrence will be $f(n) = f(n/2) + c$, which by the Master Theorem yields $f(n) = \mathcal{O}(\log(n))$

4. Apply Euclid's algorithm to:

4a) find the $\gcd(726, 693)$. Also find the integers x, y such that $\gcd(726, 693) = 726x + 693y$.

$$\begin{aligned} \gcd(726, 693) &: 726/693 \quad q = 1 \quad r = 33 \\ \gcd(693, 33) &: 693/33 \quad q = 21 \quad r = 0 \end{aligned}$$

So $\gcd(726, 693) = 33$. To find x, y such that $726x + 693y = 33$, we work backwards:

$$\begin{aligned} 33 &= 33 \times 1 + 693 \times 0 \\ r &= 726 - q \times 693 \\ 33 &= 726 - 693 \times 1 = 726 - 693 \times 1 \\ x &= 1, y = -1 \end{aligned}$$

4b) Either determine that the following equation does not have a solution, or find x: $77x = 4 \mod 20$.

First find $\gcd(77, 20)$:

$$\begin{aligned} \gcd(77,20) :& q = 3, r = 17 \\ \gcd(20,17) :& q = 1, r = 3 \\ \gcd(17,3) :& q = 5, r = 2 \\ \gcd(3,2) :& q = 1, r = 1 \\ \gcd(2,1) :& q = 1, r = 1 \end{aligned}$$

So, $\gcd(77, 20) = 1$. Next we find x, y such that $77x + 20y = 1$:

$$\begin{aligned} x = 0, y = 1 :& 1 = 2 \times 0 + 1 \times 1 \\ x = y, y = x - yq :& 1 = 1 \times 3 + (0 - 1 \times 1) \times 2 = 3 \times 1 - 2 \times 1 : x = 1, y = -1 \\ x = y, y = x - yq :& 1 = 17 \times (-1) + 3 \times (1 - 5 \times (-1)) : x = -1, y = 6 \\ x = y, y = x - yq :& 1 = 20 \times 6 + 17 \times (-1 - 1 \times 6) : x = 6, y = -7 \\ x = y, y = x - yq :& 1 = 77 \times (-7) + (6 - (-7) \times 3)a : x = -7, y = 27 \end{aligned}$$

So, we have $1 = 77 \times (-7) + 20 \times (27)$. Now, reducing modulo 20, the second term, since it is a multiple of 20, equals 0 modulo 20. So we have

$$77 \times (-7) = 1 \mod 20 \quad \Leftrightarrow \quad 77 \times 13 = 1 \mod 20$$

since $-7 = 13 \mod 20$. So $x = 13$ is the solution to the equation $77x = 1 \mod 20$. Multiplying both sides by 4 we get $77 \times (13 \times 4) = 4 \mod 20$. Since $13 \times 4 = 52 = 12 \bmod 20$, we get $77 \times 12 = 4 \mod 20$. So $x = 12$.