COMP 3804 — Assignment 1

Due: Sunday February 7, 23:59.

Assignment Policy:

- Your assignment must be submitted as one single PDF file through cuLearn.

> Use the following format to name your file:
>
> `LastName_StudentId_a1.pdf`

- **Late assignments will not be accepted. I will not reply to emails of the type "my internet connection broke down at 23:57" or "my scanner stopped working at 23:58", or "my dog ate my laptop charger".**
- You are encouraged to collaborate on assignments, but at the level of discussion only. When writing your solutions, you must do so in your own words.
- Past experience has shown conclusively that those who do not put adequate effort into the assignments do not learn the material and have a probability near 1 of doing poorly on the exams.
- When writing your solutions, you must follow the guidelines below.
  - You must justify your answers.
  - The answers should be concise, clear and neat.
  - When presenting proofs, every step should be justified.

> Some useful formulas:
>
> 1. for any real number $x > 0$, $x = 2^{\log x}$.
> 2. for any real number $x \neq 1$ and any integer $k \geq 1$,
>
> $$1 + x + x^2 + \cdots + x^{k-1} = \frac{x^k - 1}{x - 1}.$$

Question 1: Write your name and student number.

Question 2: Solve the following recurrences using the unfolding method we have seen in class. In each case, give the final answer using Big-O notation.

(2.1)

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 1 + 2 \cdot T(n/3) & \text{if } n \geq 3. \end{cases}$$

You may assume that n is a power of 3.

(2.2)

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 1 + 2 \cdot T(n-1) & \text{if } n \geq 2. \end{cases}$$

(2.3)

$$T(n) = \begin{cases} 1 & \text{if } n = 2, \\ 1 + T(\sqrt{n}) & \text{if } n \geq 4. \end{cases}$$

You may assume that n is a power of a power of 2, so that $\log \log n$ is an integer.

Question 3: In class, we have seen a fast algorithm that computes the product of two large integers. You may think that squaring an integer is "easier" than multiplying two integers. After all, for squaring, you get one input integer x, and have to multiply it by itself, whereas for multiplication, you have to compute the product of two input integers x and y. In this question, you will show that this is not the case: The time complexity of multiplication is the same (except for constant factors) as that of squaring.

1. You are given an algorithm $\mathcal{A}$ that takes as input an integer $x \geq 1$, written in binary, and returns the integer x^2. We denote the running time (i.e., the number of bit operations) of algorithm $\mathcal{A}$ by $S(n)$, where n is the number of bits in the binary representation of x.

2. You learned in elementary school that, given two integers $x \geq 1$ and $y \geq 1$, both written in binary and both having n bits, their sum $x + y$ and difference $x - y$ can be computed in $O(n)$ time.

3. Given an integer $x \geq 2$, written in binary, and given an integer $k \geq 1$, such that x is a multiple of 2^k, we can compute $x/2^k$ in $O(k)$ time, by just removing the rightmost k bits from the binary representation of x.

Describe (in plain English or pseudocode), an algorithm $\mathcal{B}$ that takes as input two n-bit integers x and y, and returns the product xy. Your algorithm $\mathcal{B}$ must use algorithm $\mathcal{A}$ as a subroutine, as well as the results in 2. and 3. above, and its running time must be $O(S(n+1))$.

It is easy to come up with an algorithm $\mathcal{B}$ that calls $\mathcal{A}$ three times. You will get more marks if your algorithm $\mathcal{B}$ calls $\mathcal{A}$ only twice. *Hint:* $(x + y)^2$.

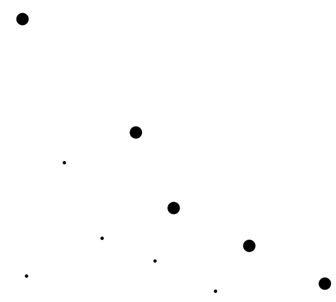

Figure 1: *The •-points are maximal; the ·-points are not maximal.*

Question 4: Consider the following recursive algorithm APPLE(n), which takes as input an integer $n \geq 1$:

> **Algorithm** APPLE(n):
> **if** $n = 1$
> **then** sing "an apple a day keeps the doctor away"
> **else** eat one apple;
>   choose an arbitrary integer m with $1 \leq m \leq n - 1$;
>   APPLE(m);
>   APPLE($n - m$)
> **endif**

(4.1) Explain why, for any integer $n \geq 1$, algorithm APPLE(n) terminates.

(4.2) Let $A(n)$ be the number of apples that you eat when running algorithm APPLE(n). Determine the exact value of $A(n)$.

Question 5: Let S be a set of n points in the plane. Each point p of S is given by its x- and y-coordinates p_x and p_y, respectively.

A point p of S is called *maximal* in S, if there is no point in S that is to the north-east of p, i.e.,

$$\{q \in S \setminus \{p\} : q_x \geq p_x \text{ and } q_y \geq p_y\} = \emptyset.$$

See Figure 1 for an example. Observe that, in general, there is more than one maximal element in S.

Give a *divide-and-conquer*[1] algorithm (in plain English or pseudocode) that computes all maximal elements in S. The running time of your algorithm must be $O(n \log n)$. Explain why the running time of your algorithm is $O(n \log n)$ (you may use the Master Theorem).

Hint: At the start of the algorithm, sort the points of S from left to right. Use this ordering to divide the input set into two subsets. You don't have to give pseudocode for the

[1] Your algorithm must use divide-and-conquer

sorting step. To avoid special cases, you may assume that no two points in S have the same x-coordinate, and no two points in S have the same y-coordinate.

Question 6: *(Binary search in an infinite array)* Let $A[1\ldots]$ be an infinite array of real numbers such that

$$0 = A[1] < A[2] < A[3] < \ldots,$$

and let x be a positive real number.

In this question, you are asked to describe (in plain English or pseudocode) an algorithm that decides whether or not x occurs in A.

Let n be the index such that $A[n] \leq x < A[n+1]$. Hence, if x occurs in A, then $x = A[n]$; if x does not occur in A, then $A[n] < x < A[n+1]$. The running time of your algorithm must be $O(\log n)$. Keep in mind that the input consists *only* of the array A and the real number x; the index n is *not* part of the input. In other words, at the start of the algorithm, you *do not know* the value of n.

You are allowed to use the binary search algorithm as a subroutine, and you may use the fact that this algorithm has a logarithmic running time.

Explain why the running time of your algorithm is $O(\log n)$

Hint: In $O(\log n)$ time, compute an index h that is not "too large" and for which $A[h] > x$. (You may even show that such an index can be computed in $O(\log\log n)$ time, but the rest of the algorithm will still take $O(\log n)$ time.)

Question 7: Let $A[1\ldots n]$ be an array containing numbers in sorted order; you may assume that all these numbers are distinct. The following algorithm is a randomized version of the binary search algorithm. The input consists of the array A, its size n, and a number x. If x is in the array, then the algorithm returns the index p such that $A[p] = x$. Otherwise, the algorithm returns "not present".

Algorithm RANDOMIZEDBINARYSEARCH(A, n, x):
$\ell = 1$; $r = n$;
while $\ell \leq r$
do $p =$ uniformly random element in $\{\ell, \ell+1, \ldots, r\}$;
    if $A[p] < x$
    then $\ell = p + 1$
    else if $A[p] > x$
        then $r = p - 1$
        else return p
        endif
    endif
endwhile;
return "not present"

Let T be the running time of this algorithm. Note that T is a random variable. Prove that the expected value of T is $O(\log n)$.