Homework 1: Divide and Conquer
CS 6161: Design and Analysis of Algorithm (Fall'20)

Due: 6 pm, September 16 (Wednesday)

General Instructions The assignment is meant to be challenging. Feel free to discuss with fellow students, however please write up your solutions independently (e.g., start writing solutions after a few hours of any discussion) and acknowledge everyone you discussed the homework with on your writeup. The course materials are all on UVA Collab. You may refer to any materials covered in our class. However, any attempt to consult outside sources, on the Internet or otherwise, for solutions to any of these homework problems is *not* allowed.

Whenever a question asks you to design or construct an algorithm, please formally write the pseudo-code in an algorithm box with all the essential elements (the input, output, and operations); whenever a question asks you to "show" or "prove" a claim, please provide a formal mathematical proof. Unless specified otherwise, in every problem set, when we ask for an answer in asymptotic notation, we are asking either for a $\Theta(\cdot)$ result, or else the tightest $O(\cdot)$ result you can come up with.

Submission Instructions Please write your solutions in LaTeXusing the **provided template**, hand-written solutions will not be accepted. **You must submit your answer in PDF version via Gradescope.** You should have received a notice about signing up for Gradescope — if not, please let the TAs know as soon as possible. Hope you enjoy the homework!

Problem 1

1. **[8 Points]** Review the definition of asymptotic notions. Now given a set of functions, $S = \{h, k, m\}$ where $h(n) = \sqrt{n} + \log n$, $k(n) = 2^n/n^9$, $m(n) = (\log n)^9 + n^{1/2}$.

   For any pair of functions f, g from the above set of functions, say whether it holds that $f = O(g)$, $f = \Omega(g)$, $f = o(g)$, $f = \omega(g)$ or $f = \Theta(g)$. If multiple relations between f and g hold, state the stronger one. For example if $f = o(g)$ and also $f = O(g)$, then only say $f = o(g)$, or alternatively $g = \omega(f)$. **Note:** *Give a short argument for each of your answers in this problem.*

2. **[10 Points]** Consider the following functions, and rearrange them in ascending order of growth rate. That is, if function $g(n)$ comes after function $f(n)$ in your list, then it should be the case that $f(n) =$

$O(g(n))$.

$$\begin{aligned} f_1(n) &= n^{\sqrt{\log n}} \\ f_2(n) &= 7n + o(n) \\ f_3(n) &= 100\sqrt{n} + 2n^{1/4} + 100000 \\ f_4(n) &= 2^{\log^2 n} \\ f_5(n) &= 2^{2\log n} + n^{1.5} \\ f_6(n) &= 3^n \\ f_7(n) &= \ln(n!) \\ f_8(n) &= 2^{1.5^n} \end{aligned}$$

Note: *Be careful about the superscript in the mathematical expressions*

3. **[12 Points]** Assume you have functions f and g such that $f(n) = O(g(n))$. For each of the following statement, decide whether you think it is true of false and give a proof or counterexample. You may assume that all functions mentioned are monotone non-decreasing and non-negative.

   (a) $\log_2 f(n) = O(\log_2 g(n))$

   (b) $2^{f(n)} = O(2^{g(n)})$

   (c) $f(n)^2 = O(g(n)^2)$

Problem 2

[10 Points] Consider the recurrence relation $T(n) = T(n-10) + n$ for $n > 10$, with $T(n) = 1$ for all $n \leq 10$. Your friend claims that $T(n) = O(n)$, and offers the following justification:

Let's use the Master Theorem with $a = 1, b = \frac{n}{n-10}$, and $d = 1$. This applies since, for any $n > 10$, we have

$$\frac{n}{b} = n \cdot \left(\frac{n-10}{n}\right) = n - 10$$

Then for any $n > 10$, we have $a < b^d$, so the Master Theorem says that $T(n) = O\left(n^d\right) = O(n)$

Do you agree with your friend's argument? If not, what is the correct answer? If it helps, you may assume that n is a multiple of 10.

Note: *To convince your friend, you need to clearly identify the faulty logic above, then give your solution to this recurrence along with a short but convincing justification (no formal proof needed).*

Problem 3

In a set T of $2n-1$ integers, the *EGZ* subset $S \subset T$ is of size exactly n and that the sum of all its elements is a multiple of n, i.e., $n \mid \sum_{x \in S} x$. As the Erdös–Ginzburg–Ziv theorem implies that the *EGZ* subset must exists in any set, the remaining task for you is to determine all of the elements in this subset efficiently.

1. **[15 Points]** Suppose $n = 2^{\mathrm{k}}$ is a power of 2. Please design an algorithm to find one such *EGZ* subset for any set of $2n-1$ integers in $o(n^2)$, and prove its correctness and time complexity. Your algorithm should take a set T as input and output one valid *EGZ* subset S.

   Hint: *Try to solve a few small cases by hand.*

2. **[15 Points]** Suppose n is an arbitrary composite number. Given an oracle subroutine that finds in $O(p)$ time an *EGZ* subset from a set of $2p-1$ integers for any prime number p, can you design a polynomial time (w.r.t. n) algorithm for the case of any composite n? Prove the correctness and time complexity of your algorithm.

Problem 4

A polynomial of degree-n, $A(x) = \sum_{i=0}^{n-1} a_i x^i$, can be represented as a vector of its coefficients $\boldsymbol{a} = (a_0, a_1, \cdots, a_{n-1}) \in \mathbb{R}^n$.

1. **[10 Points]** Given a polynomial $A(x)$ of degree-n with its representation vector $\boldsymbol{a}$, you want to compute $A(x_0)$ for any given real number $x_0 \in \mathbb{R}$. Suppose that you are only allowed to use the four "basic operations": $+, -, \times, /$, each taking a unit of time. However, you cannot compute x^n in constant time as it requires n multiplications. Design an algorithm to compute $A(x_0)$ in $O(n)$ time. Your algorithm should take $\boldsymbol{a}$ as input and correctly outputs the value $A(x_0)$.

   Moreover, prove that $O(n)$ is the optimal time complexity lower bound — i.e., there is no algorithm that can correctly compute $A(x_0)$ in $o(n)$ time.

2. **[8 Points]** During Lecture 3, we described the Fast Fourier Transform (FFT) algorithm which can be viewed as computing the values of a degree-$(n-1)$ polynomial at ω_n^j for $j = 0, 1, \cdots, n-1$ (recall $\omega_n = e^{-2\pi i/n}$). The algorithm we described requires n to be a power of 2. In this question, you are asked to show that the algorithm can be adapted to work for any n and ω_m.

   In particular, show how to adapt the FFT algorithm to compute the values of a degree-$(n-1)$ polynomial at ω_m^j for $j = 0, 1, \cdots, m-1$ for any m, n (n is not necessarily a power of 2 any more).

3. **[12 Points]** The key idea underlying FFT is to divide a polynomial into odd and even terms. However, this is not the only way to apply the idea of divide-and-conquer. In fact, cutting a polynomial right in the middle appears a more straightforward way. Please design a divide and conquer algorithm for polynomial multiplication using this idea and show that it can still improve upon the time efficiency $O(n^2)$. For simplicity purposes, you may assume n to be a power of 2.

   Hint: *Try to write* $A(x) = \sum_{i=0}^{n-1} a_i x^i$ *as* $\sum_{i=0}^{(n-2)/2} a_i x^i + \sum_{i=n/2}^{n-1} a_i x^i$ *and see what happens.*