Basic Algorithms — Fall 2020 — Problem Set 7
Due: Wed, Dec 2, 11am

1. **Mastering the Master Theorem.** Use the "Master Theorem," as discussed in class, to solve the following recurrences:

   (a) $T(n) \leq 2T(\lfloor n/4 \rfloor) + O(\sqrt{n})$
   (b) $T(n) \leq 2T(\lfloor n/4 \rfloor) + O(n^{0.51})$
   (c) $T(n) \leq 3T(\lfloor n/3 \rfloor) + O(n)$
   (d) $T(n) \leq 16T(\lfloor n/4 \rfloor) + O(n^{1.5})$
   (e) $T(n) \leq 7T(\lfloor n/2 \rfloor) + O(n^2)$

2. **Weighing coins.** You are given n coins — they all look identical, and all have the same weight except one, which is heavier than all the rest. You also have a balance scale, on which you can place one set of coins on one side, and another set of coins on the other, and the scale will tell you whether the two sets have the same weight, and if not, which is the heavier set.

   Suppose that performing one such weighing takes one minute, and in addition, you have to pay a fee of m dollars, where m is the total number of coins placed on the scale in that weighing.

   Design and analyze a strategy that will identify the heavy coin in $O(\log n)$ minutes and at a cost of $O(n)$ dollars. You may analyze the your strategy using the Master Theorem. You may not assume that n has any special form (although you might want to first think about the case where n is a power of 2).

3. **Quasi-linear recursion tree analysis.** Suppose we have an algorithm that on inputs of size n, recursively solves two problems of size $n/2$, with a "local running time" $t(n)$ that satisfies

   $$c_1 n \log_2(n) \leq t(n) \leq c_2 n \log_2(n)$$

   for some constants c_1, c_2 and all $n \geq 2$. For simplicity, assume that n is a power of 2. Also, assume that $n \geq 2$ and the recursion stops when $n = 2$.

   Let $T(n)$ be the total running time of the algorithm on inputs of size n. Prove that

   $$d_1 n(\log_2(n))^2 \leq T(n) \leq d_2 n(\log_2(n))^2$$

   for some constants d_1, d_2 and all $n \geq 2$.

   Note that the Master Theorem is not directly applicable here. Instead, you should prove this using a recursion tree analysis, similar to the one that was used to prove the Master Theorem in class.

   Hint: for each level j, estimate the number of subproblems at that level, and the local running time for each subproblem at that level.

4. **Uneven divide and conquer (1).** Suppose we have an algorithm that on problems of size n, if $n \geq 10$, it recursively solves two subproblems, one of size $\lfloor 0.7n \rfloor$ and one of size $\lfloor 0.2n \rfloor$. Furthermore, the "local running time" is $O(n)$.

   Prove that $T(n) = O(n)$ using a recursion tree analysis.

   Hint: Let S_j be the sum of all subproblem sizes at level j. Show that $S_{j+1} \leq 0.9 S_j$ for all $j \geq 0$. To get started, suppose the subproblems at level j have sizes $n_1, \ldots, n_k$, so $S_j = \sum_i n_i$.

5. **Uneven divide and conquer (2).** Suppose we have an algorithm that on problems of size n, if $n \geq 10$, it recursively solves two subproblems, one of size $\lfloor 0.9n \rfloor$ and one of size $\lfloor 0.1n \rfloor$. Furthermore, the "local running time" is $O(n^2)$.

   Prove that $T(n) = O(n^2)$ using a recursion tree analysis.

   Hint: Let S_j be the sum of the *squares* of the subproblem sizes at level j. Show that $S_{j+1} \leq 0.82 S_j$ for all $j \geq 0$. To get started, suppose the subproblems at level j have sizes $n_1, \ldots, n_k$, so $S_j = \sum_i n_i^2$.

6. **Roots to coefficients.** You are to design an efficient algorithm for the following problem. The input is a list of elements $a_1, a_2, \ldots, a_n \in F$, where F is some field. The output is the coefficient vector of the polynomial

$$(X - a_1)(X - a_2) \cdots (X - a_n) \in F[X].$$

For example, if the input is the list $1, 2, 3, 4$, the output would be $(24, -50, 35, -10, 1)$, which is the coefficient vector of the polynomial

$$(X-1)(X-2)(X-3)(X-4) = 24 - 50X + 35X^2 - 10X^3 + X^4.$$

A simple algorithm runs as follows:

$f \leftarrow 1$
for $i \in [1\,..\,n]$ do
    $f \leftarrow f \cdot (X - a_i)$
output f

Note that in general, if we have

$$f = c_0 + c_1 X + \cdots + c_d X^d,$$

and $a \in F$, then we have

$$f \cdot (X - a) = -ac_0 + (c_0 - ac_1)X + \cdots + (c_{d-1} - ac_d)X^d + c_d X^{d+1}.$$

That is, given the coefficient vector of f, which is $(c_0, c_1, \ldots, c_d)$, we can compute the coefficient vector of $f \cdot (X - a)$, which is $(-ac_0, c_0 - ac_1, \ldots, c_{d-1} - ac_d, c_d)$, using $\Theta(d)$ operations in F.

So in the above algorithm, each loop iteration costs $\Theta(i)$ operations in F, and hence the total cost of the above algorithm is $\Theta(n^2)$ operations in F.

Your task is to design a faster algorithm for this problem. That is, your algorithm should use $o(n^2)$ operations in F. To this end, you may use, as a subroutine, a fast polynomial multiplication algorithm *Mul* that takes as input two polynomials in $F[X]$ and outputs their product (inputs and outputs are represented by their coefficient vectors). You may assume that the algorithm *Mul* runs in time $O(d^\alpha)$, where d is a bound on the degrees of the input polynomials, and α is a constant, where $1 < \alpha < 2$.

Use a divide and conquer strategy to design your algorithm. You may analyze the running time of your algorithm using the "Master Theorem" from class.