CSE548, AMS542: Analysis of Algorithms, Spring 2014 **Date:** Feb 25

Homework #1

(Due: Mar 11)

Task 1. [20 Points] Multiplying Long Integers

Karatsuba's algorithm multiplies two n-bit integers by recursively computing three (3) pairwise products of $\frac{n}{2}$-bit integers (see Lecture 2).

(a) [**5 Points**] Show that the problem can be solved by recursively computing six (6) pairwise products of $\frac{n}{3}$-bit integers. Analyze the running time of the algorithm.

(b) [**15 Points**] Can you reduce the number of products of $\frac{n}{3}$-bit integers required in part (a) to five (5)? If not, why? If yes, describe the algorithm and analyze its running time.

Task 2. [20 Points] Deterministic Select

The algorithm we analyzed in the class (in Lecture 7) divided the input array into groups of size 5.

(a) [**10 Points**] Repeat the analysis with groups of size 3, and report the best bound you get.

(b) [**10 Points**] What bound do you get for groups of size 7?

Task 3. [20 Points] Searching in a Random Binary Search Tree

Consider constructing a binary search tree from n numbers as follows. Pick a number uniformly at random from the n input numbers as a pivot, put the pivot in the root node, put all numbers smaller than the pivot in the left subtree and the rest in the right subtree. Apply the same partitioning step recursively on the two subtrees.

(a) [**5 Points**] Consider a node in the tree containing m numbers. Argue that for any constant $\alpha \in \left(0, \frac{1}{2}\right)$ and large enough m, with probability at least $1 - 2\alpha$, each of the two subtrees of the node will contain between αm and $(1 - \alpha)m$ numbers.

(b) [**15 Points**] Show that the expected search time in such a tree can be upper bounded by $T(n)$ which is described by the following recurrence relation:

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n < 10, \\ \frac{1}{3}T\left(\frac{2n}{3}\right) + \left(\frac{1}{2} - \frac{1}{3}\right)T\left(\frac{3n}{4}\right) + \left(\frac{3}{5} - \frac{1}{2}\right)T\left(\frac{4n}{5}\right) + \frac{2}{5}T(n) + \Theta(1) & \text{otherwise.} \end{cases}$$

Solve for $T(n)$.

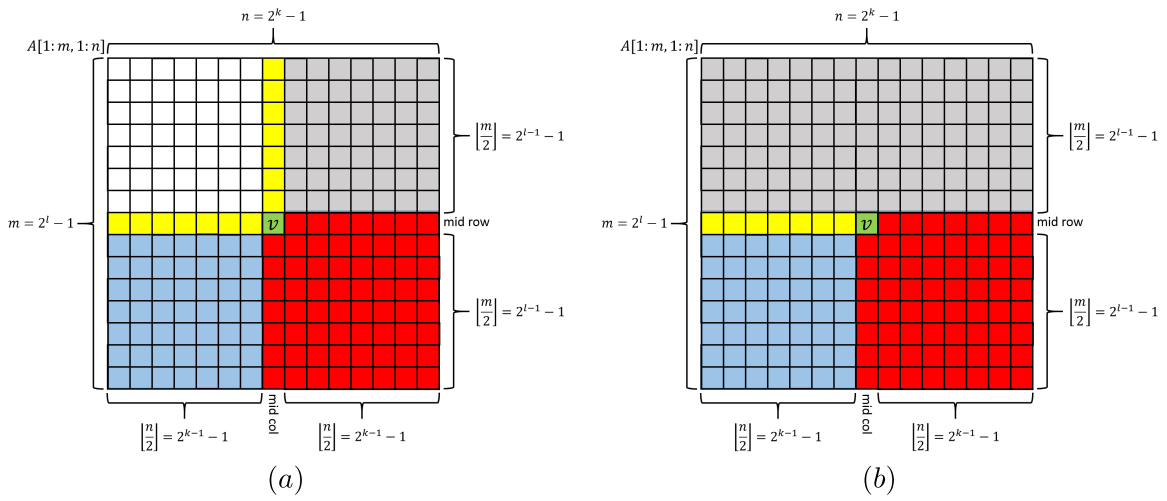
$n = 2^k - 1$
$A[1{:}m, 1{:}n]$
$\lfloor \frac{m}{2} \rfloor = 2^{l-1} - 1$
$m = 2^l - 1$
mid row
v
$\lfloor \frac{m}{2} \rfloor = 2^{l-1} - 1$
mid col
$\lfloor \frac{n}{2} \rfloor = 2^{k-1} - 1$
$\lfloor \frac{n}{2} \rfloor = 2^{k-1} - 1$
(a)
$n = 2^k - 1$
$A[1{:}m, 1{:}n]$
$\lfloor \frac{m}{2} \rfloor = 2^{l-1} - 1$
$m = 2^l - 1$
mid row
v
$\lfloor \frac{m}{2} \rfloor = 2^{l-1} - 1$
mid col
$\lfloor \frac{n}{2} \rfloor = 2^{k-1} - 1$
$\lfloor \frac{n}{2} \rfloor = 2^{k-1} - 1$
(b)

Figure 1: Searching in a sorted 2D array.

Task 4. [30 Points] Searching in a Sorted 2D Array

Suppose you are given an $m \times n$ array A containing mn distinct numbers (see Figure 1), and you are required to find if a given number x appears in A. Assume for simplicity that $m = 2^l - 1$ and $n = 2^k - 1$, where l and k are positive integers. The numbers in each row of A is sorted in increasing order of value from left to right, and those in each column is sorted in increasing order from top to bottom.

(a) [**5 Points**] Consider the following divide-and-conquer algorithm for finding x in A as outlined in Figure $1(a)$. Suppose v is the number that appears at the intersection of the middle row and the middle column of A. If $x = v$, we are done. If $x < v$, then x cannot appear in the bottom-right (red) quadrant of A. But it can appear in any of the other three quadrants (top-left/white, top-right/gray and bottom-left/blue), or in the left (yellow) part of the mid row, or the top (yellow) part of the mid column. One can recursively search for x in the white, gray and blue quadrants, and peform binary search in the yellow areas. Similarly for $x > v$. Write a recurrence relation describing the running time of the algorithm, and solve it.

(b) [**15 Points**] The divide-and-conquer algorithm outlined in Figure $1(b)$ is similar to that in Figure $1(a)$, but when $x < v$, a single recursive function call is made to find x in the top half of A (colored gray), one recursive call finds x in the bottom-left quadrant (colored blue), and a binary search finds x in the left (yellow) part of the mid row. Similarly for $x > v$. Write a recurrence relation giving an upper bound on the running time of the algorithm, and solve it.

(c) [**10 Points**] Can you modify the approach given in this task to achieve better performance?

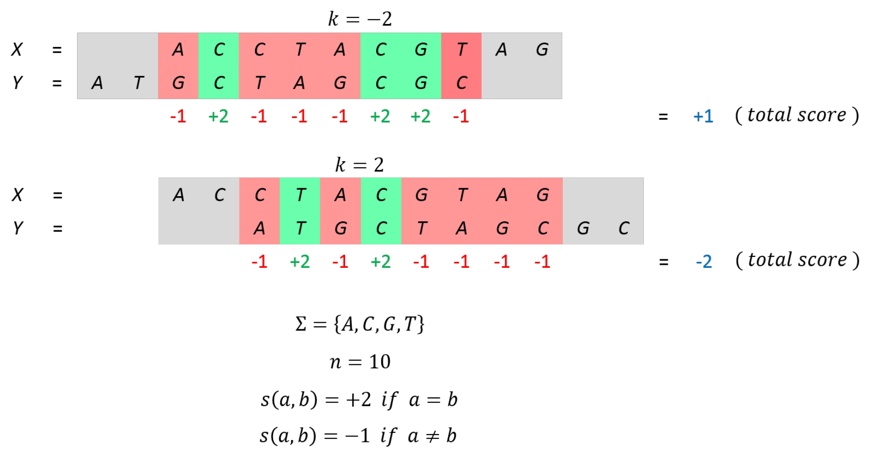

Figure 2: Gapless alignment of two DNA sequences.

Task 5. [20 Points] Gapless Sequence Alignment (Oleksii Starov)

Sequence alignment plays a central role in biological sequence comparison, and can reveal important relationships among organisms. Given two strings[1] $X = x_1x_2\ldots x_n$ and $Y = y_1y_2\ldots y_n$ over a finite alphabet[2] Σ, a *gapless alignment* of X and Y is obtained provided there exists an integral *shift* $k \in (-n, n)$ such that for all $i \in [1 + \max(0, k), \leq n + \min(0, k)]$, x_i is aligned with y_{i-k}. Observe that the alignment does not tear or break the sequences, i.e., does not insert gaps inside any of the sequences for better alignment.

Suppose you are given an *alignment score* $s(a, b)$ for each pair $a, b \in \Sigma$, meaning that your overall alignment score increases by $s(a, b)$ for every aligned pair $\langle x_i, y_{i-k}\rangle$ with $x_i = a$ and $y_{i-k} = b$. Assume $s(a, b) = s(b, a)$. Note that depending on a and b, $s(a, b)$ can take any (fixed) real value (positive, negative, zero). The *total alignment score* for shift k is $\sum_{1+\max(0,k)\leq i\leq n+\min(0,k)} s(x_i, y_{i-k})$. Figure 2 shows an example.

Your goal is to find a shift $k_{opt} \in (-n, n)$ that maximizes the total alignment score. Observe that a naïve algorithm can find k_{opt} in $\Theta\left(n^2\right)$ time. Describe an algorithm that improves over this bound, and analyze its running time.

[1] for simplicity, both strings are assumed to have the same length

[2] for DNA sequences $\Sigma = \{A, C, G, T\}$, for RNA strings $\Sigma = \{A, U, G, C\}$, and for amino acid sequences (proteins) $\Sigma = \{A, R, N, D, C, E, Q, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$