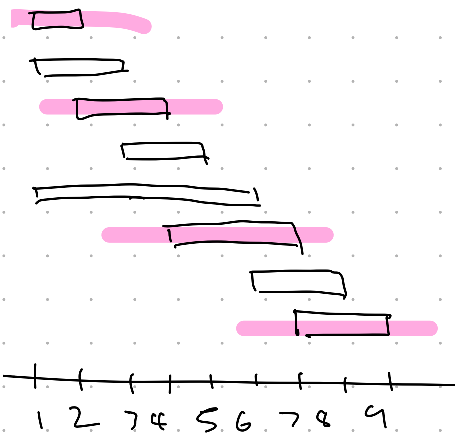
HW 6

<u>12.8.6</u>   * start, fin, benefit

$L = \{ \overset{1}{(1,2,5)}, \overset{2}{(1,3,4)}, \overset{3}{(2,4,7)}, \overset{4}{(3,5,2)}, \overset{5}{(1,6,3)}, \overset{6}{(4,7,5)}, \underset{7}{(6,8,7)}, \underset{8}{(7,9,4)} \}$

Benefit	Predecessor
5	0
4	0
7	1
2	2
3	0
5	3
7	5
4	6



$5+7+5+4 = 21$

Solution: 1, 3, 6, 8

Benefit: 21

1a) $O(n\log n)$

1b) You'll find the middle of the matrix $A[\frac{n}{2}][\frac{n}{2}]$. If the value at the spot is higher, you'll recursively search the right half, if it was smaller, you'll search the left, or the value would be what you were searching for.

1c) $T(n) = T(n/2) + O(n)$

1d) $a=1 \quad b=2 \quad a<b \rightarrow$ case 3

$O(n)$

1e) Divide and Conquer is faster b/c $O(n)$ is faster $O(n\log n)$.

2) For this scenario, to divide and conquer you will divide the array in half. Then using the bit(i, j) function, you will iterate through each bit of each value. You will also calculate expected number of 0 and 1 in the section. the numbers don't line up, you will recursively go through the array to find the missing value.

$$T(n) = O(n) + T(n/2)$$

$$a=1 \quad b=2$$

$$a<b \rightarrow \text{master theorem}$$

$$O(n)$$

3) $T(n) = 10T(n/2) + O(n^2)$

$a = 10 \quad b = 2 \quad k = 2$

$a > b \rightarrow$ Case 1 $\quad O(n^{\log_2 10})$

4) $T(n) = rT(n/2) + O(n^2)$

$O(n^{\log_2 10}) = rT(n/2) + O(n^2)$

$O(n^{\log_2 10} - n^2) = r\cancel{T(n/2)}$

$O(n^{2.81} - n^2) = r$

$O(n^{.81}) = r$

$\boxed{r < n^{.81}}$

Collaborators: Aliza Reshamwalla
Vincent Carluccio

Additional Homework 6 Problems
CompSci 161—Fall, 2023—Dillencourt

1. Consider the problem of searching in a sorted matrix. That is, you are given an $n \times n$ matrix A, where each entry is an integer. Each row of the matrix is sorted in ascending order, and each column is also sorted in ascending order. Given a value x, the problem is to decide whether x is stored somewhere in the array (i.e., whether there is some i and j such that $A[i][j] = x$).

   (a) One way to solve this problem is to do binary search in each row. What is the worst-case running time of this algorithm (in terms of n)?

   (b) Give a divide-and-conquer algorithm for this problem. (Hint: Your algorithm needs to call itself recursively, so think carefully about the parameters required. First compare x with the element in the "middle" of your array (i.e., middle row, middle column). Then ...?).

   (c) Write down a recurrence equation describing the running time of your algorithm from (b).

   (d) Solve your equation from (c), using the formula for solving divide-and-conquer recurrences discussed in class.

   (e) Based on your answers to the above questions, which algorithm for solving the problem is faster: binary search in each row or the divide-and-conquer algorithm from part (b)?

2. Suppose you are given an unsorted array $A[1..n]$, which contains all but one of the integers in the range $0, \ldots, n$ (so exactly one of these elements is missing from A). The problem is to determine the missing integer in $O(n)$ time. Each element of A is represented in binary, and the only operation available is the function `bit`(i, j), which returns the value of the jth bit of $A[i]$ and takes constant time. Show that using only this operation, it is still possible to determine the missing integer in $O(n)$ time, by giving a divide-and-conquer algorithm. Be sure to explicitly state and then solve the recurrence equation for the running time of your algorithm.

   Hint/Suggestion: It may simplify your answer if you assume that n is of the form $2^k - 1$ for some integer k. If it helps, you you are free to make this assumption.

3. Suppose we could multiply two 3×3 matrices using 25 scalar multiplications and a constant number of scalar additions and subtractions. Set up and solve the recurrence relations to analyze the resulting divide-and-conquer algorithm for matrix multiplication.

   Hint/suggestion: Consider how Strassen's algorithm uses a scheme for multiplying 2×2 matrices using 7 scalar multiplcations and a constant number of scalar additions and subtractions. Think carefully how this would change under the assumptions of this question.

4. Suppose we could multiply two 3×3 matrices using r scalar multiplications and a constant number of scalar additions and subtractions. How small would r have to be to make the resulting divide-and-conquer algorithm for matrix multiplication asymptotically faster than Strassen's matrix multiplication algorithm? Justify your answer.

   Hint/suggestion: Solve the previous problem before trying to solve this one.