CS 3510 Section C: Design & Analysis of Algorithms August 27nd 2023

HW 2: Divide & Conquer and Arithmetic

YOUR NAME HERE Due: September 3rd 2023

- Please type your solutions using LaTeX or any other software. Handwritten solutions will not be accepted.
- Your algorithms must be in plain English & mathematical expressions, and the pseudo-code is optional. Pseudo-code, without sufficient explanation, will receive no credit.
- Unless otherwise stated, all logarithms are to base two.
- If we ask for a specific running time, a correct solution achieving it will receive full credit even if a faster solution exists.

1.) Divide & Conquer (25 points)

Let's consider a group of n monkeys lined up in a row. Each monkey holds a banana with a unique number printed on its sticker, and these monkeys are lined up such that the numbers on their banana stickers are increasing. Design an efficient algorithm to determine if there exists some number i such that the monkey standing in position i holds a banana with number i printed on it. Assume that these positions are zero-indexed.

(a.) Describe a simple algorithm to solve this problem that takes $\mathcal{O}(n)$ time.

Solution: Loop over all monkeys keeping track of position i, and check if a monkey holds a banana with sticker label i. If we find such a monkey, return the current i. If no such monkey is found, output that no such i exists.

(b.) Find a more efficient Divide & Conquer algorithm that takes $o(n)$ time. Calculate your algorithm's runtime by finding a recurrence and justify its correctness.

Solution: We use a modified form of binary search. **Note that this solution assumes banana numbers can be negative. If banana numbers are just natural numbers, then just checking the first index is sufficient and takes $\mathcal{O}(1)$ time.**

(a) Start with $i = 0$, $j = n - 1$. Calculate midpoint mid between i and j.
(b) If the monkey at position mid holds a banana labelled mid, return mid.
(c) Otherwise, if the banana label is lower than mid, then it is not possible for us to find a match before position mid. We set $i = mid + 1$ and repeat.
(d) If the banana label is higher than mid, there cannot be a match after position mid. We set $j = mid - 1$ and repeat.

Our recurrence is $T(n) = T(n/2) + \mathcal{O}(1)$, which yields $\mathcal{O}(\log n)$ runtime.

2.) Divide & Conquer (25 points)

You are given two sorted lists of size m and n. Give an $\mathcal{O}(\log m + \log n)$ time algorithm for computing the kth smallest element in the sorted list formed by merging the two lists together (recall `merge` from `merge sort`). For example, given list $A = [0, 1, 2, 5, 6]$ and $B = [-15, 0, 1, 3, 17]$ with $k = 3$, your algorithm should output 0. Remember to justify your algorithm's correctness, show your recurrence, and prove runtime using the Master Theorem.

Solution: We use a modified form of binary search. Consider A to be the sorted list of size m, and B to be the sorted list of size n.

1. Calculate the middle indices elements a_{mid} and b_{mid} of A and B. We first need to determine whether the the midpoints include the kth smallest element.
2. If $a_{mid} + b_{mid} < k$, we know the kth smallest must reside on the right side of either array.
   (a) If $A[a_{mid}] \leq B[b_{mid}]$, then we recur on the second half of A and the same array B. We update k to equal $k - a_{mid} - 1$.
   (b) Otherwise, we recur on the same array A and the second half of B. We update k to equal $k - b_{mid} - 1$.
3. Otherwise, the kth smallest must reside on the left side of either array.
   (a) If $A[a_{mid}] \leq B[b_{mid}]$, then we recur on the same array A and the second half of B. We keep k the same.
   (b) Otherwise, we recur on first half of A and the same array B. We keep k the same.

The runtime of this approach is $T(n, m) = T(n/2, m)$ *or* $T(n, m/2) + \mathcal{O}(1)$. This is upper-bounded by $\log mn = \log m + \log n$.

3.) Divide & Conquer (25 points)

You are given an array of integers A of length n. The *minimum interval sum* is defined as the minimum sum of elements across all intervals $A[i \dots j]$ for $i \leq j$. For example, given the array $A = [-16, 2, -10, 2, 43, -20]$, the minimum interval sum would be $(-16)+2+(-10) = -24$, which would correspond to the interval from $i = 0$ to $j = 2$ (inclusive).

Design an efficient Divide & Conquer algorithm to find the minimum interval sum of A. Justify correctness of your algorithm, provide a recurrence, and prove its runtime.

Solution: We use an approach similar to MergeSort.

1. Divide A into two halves A_L, A_R.
2. We have three cases:
   (a) The minimum interval sum is in the left half. We recur on A_L.
   (b) The minimum interval sum is in the right half. We recur on A_R
   (c) The minimum interval sum is across the left and right half, which means the interval must include the middle element! Find the minimum sum starting from mid point and ending at some point on left of mid, then find the minimum sum starting from mid + 1 and ending with some point on right of mid + 1. Combine the two as the minimum interval sum across both halves.
3. Return the minimum of all above cases.

Our recurrence is $T(n) = 2T(n/2) + \mathcal{O}(n)$, which yields a running time of $\mathcal{O}(n \log n)$.

4.) Arithmetic (25 points)

Using Fermat's Little Theorem and modular exponentiation, solve for x in the following. Show your work; answers by themselves will receive **no credit**.

(a.) $17^{2^c} \equiv x \bmod 3$ (for all constants $c > 0$)

Solution: We can rewrite 17^{2^c} as $17^2 17^2 17^2 \ldots$. Since $17^2 \equiv 1 \bmod 3$ by Fermat's Little Theorem, we have that $17^{2^c} \equiv 1 \bmod 3$.

(b.) $2^{2000} \equiv x \bmod 13$

Solution: By Fermat's Little Theorem, $2^{12} \equiv 1 \bmod 13$. $2^{2000} = (2^{12})^{166} 2^8 \equiv 256 \equiv 9 \bmod 13$.

(c.) $27^{43} \equiv x \bmod 127$

Solution: $27^{43} = 3^{129}$. We have $3^{126} \equiv 1 \bmod 127$, so $3^{129} = 3^3 3^{126} \equiv 27 \bmod 127$.

(d.) $3^{57} \equiv x \bmod 59$

Solution: We have $3^{58} \equiv 1 \equiv 60 \bmod 59$. Therefore, $3^{57} \equiv \frac{60}{3} \equiv 20 \bmod 59$.