HW3.CAB8XD

Honor Code Notice This document is exclusively for Spring 2019 CS2501 DSA2 students with Professor Raymond Pettit at The University of Virginia. Any student who references this document outside of that course during that semester (including any student who retakes the course in a different semester), or who shares this document with another student who is not in that course during that semester, or who in any way makes copies of this document (digital or physical) without consent of Professor Raymond Pettit is guilty of cheating, and therefore subject to penalty according to the University of Virginia Honor Code.

PROBLEM 1 *Fast Exponentiation*

1. Given a pair of positive integers (a, n), devise (and prove) a divide and conquer algorithm that computes a^n using only $O(\log n)$ calls to a multiplication routine.

   Proof. The following algorithm provides a power method given the base number a and the exponent n in $O(\log n)$ time.

   (a) Given int a and int n, if y is zero, return 1 (as the base case).

   (b) If n is even, the multiply the recursive call of the power of a and floor(n/2) itself and return that value.

   (c) If n is odd, follow the same step as above then multiply by x and return that value.

   The above algorithm runs in $O(\log n)$ time because the function pow calls itself $\log_2 n$ times by recursively dividing n by two until reaching the base case. The multiply operation is constant time, so the run time of the algorithm presents $O(\log n)$. □

PROBLEM 2 *Largest Element in Array*

1. Write pseudocode for a divide-and-conquer algorithm for finding the position of the largest element in an array of n numbers.

   Proof. The algorithm for finding the position of the largest element:

   (a) Given an array of comparable values, if the size of the array is 1, compare the value to the current maximum. If the value is greater than the maximum variable, update the variable. Return the maximum variable.

   (b) Call the function itself with an input of the array from the beginning index to the middle index (function parameters can be programmed either as an inputted sub list or given indices) and set equal to a left variable.

   (c) Call the function itself with an input of the array from the middle index + 1 to the last index and set equal to a right variable.

   (d) Return the maximum of the left and right values.

   □

2. What will be your algorithm output (be specific) for arrays with several elements of the largest value?

   The output will be the first index of the largest values that appears as the 'greater than' comparison is closed.

PROBLEM 3 *Closest-Pair, One-dimensional*

1. For the one-dimensional version of the closest-pair problem, i.e., for the problem of finding two closest numbers among a given set of n real numbers, design an algorithm that is directly based on the divide-and-conquer technique. Show pseudocode for your algorithm.

   Proof. The algorithm for the One-dimensional Closest Pair of Points:

   (a) Given a sorted list of n real numbers, the base case of the recursion will check if the size of the array is 2, and (if the array is odd sized) if the array is of size 3. If so, then the base case will calculate the distance between all of the possible point combinations in the array and return the minimum value.

   (b) If it is not the base case, The function will call itself with an input of the first half of its list and set it to a variable right list min.

   (c) The function will again call itself but with an input of the second half of its list and set it to a variable for left list min.

   (d) Lastly, the left and right list min variables will be compared to find their minimum. The global variable for minimums will be set to that value. The minimum value will be returned.

   □

2. Show the complexity class for your algorithm.

   Proof. The runtime of the closest-pair, one-dimensional algorithm breaks down to (1) the number of recursion of the dividing lists and (2) (assuming the list is sorted before the algorithm begins) the number of comparisons between values. The recursions repeat n times as each pair of points must compare to an adjacent point to accurately find the closest pair. Due to the adjacent sort, the actions in the algorithms sum to n-1 actions when finding the closest pair. The constant proves negligent as the sample grows, so the algorithm's runtime is $O(n)$. □