CS 515 Take-Home Worksheet #1—Algorithm Design Techniques

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name: | Liam Warren | UNH User ID: |  | Total: |  | /100 pts |

Lecture Section: 01

**[50 pts]** Consider the following “peak element” problem.

Given an array of integers, an array element is considered ***peak*** if it is NOT smaller than its neighbors. For example, if the input array is sorted in increasing order {10, 20, 30, 40} then the last element 40 is the only peak element; if the array is sorted in decreasing order {90, 70, 50, 30} then the first element 90 is the only peak element. In an array of all the same element, like {40, 40, 40, 40}, they’re all peak.

Finding a peak element in an array can easily be done in *O*(*N*) time, but there is a way to do it in *O*(log *N*) time. Pick a strategy that we talked about today, tell me which strategy you're using, and try to describe an algorithm that would find any peak element in an array in *O*(log *N*) time (the algorithm doesn’t have to find the biggest one, just any one). Also explain why you think it would take only *O*(log *N*) time.

For this Algorithm the method of divide and conquer would result in a efficiency of O(log N). The recursive divide and conquer would follow this outline:

Base Case: Array size = 2

Return larger of 2 leftover numbers

Recursive case: else

Recurse on right half of array

Recurse on left half of array

Return the larger of the 2

This can be shown to have a time efficiency of O(log N) because for an array of size 2^n, only half of the array needs to be compared, leading to 2^(n – 1) comparisons. Applying this recursively reduces to log N comparisons and thus the time efficiency is O(log N).

*More on the back…*

**[50 pts]** The Fibonacci sequence is defined by the recursive function *Fn* = *Fn*-1 *+ Fn*-2*.* A naïve implementation of this function would take an amount of time to complete that is exponential on the value of the input. Pick a strategy discussed in the lecture about “Algorithm Design Techniques”, tell me which technique(s) you're using, and describe an algorithm that would be able to calculate a given Fibonacci number in *O*(*N*) time. Describe why you think it would only take *O*(*N*) time, and also describe how much space you think it would take.

This algorithm can be achieved using Dynamic Programming. All that needs to be considered is the nth number, and the previous sum of n – 1 numbers, so the recursive algorithm would look like this:

Fib(N, calls, sum)

Base case: calls == N

Return sum + calls

Recursive case: else

Sum += calls

Return fib(N, calls + 1, sum)

This algorithm only uses one iteration per call, thus having a time efficiency of O(n), by dynamically storing the result of each function call and passing it along to the next call of the function. While this algorithm is time efficient compared to the naïve implementation, it uses significantly more space because it requires N recursive calls which allocates N calls to the call stack and allocates space for 3 variables which can’t be freed by the garbage collector.