Mahitha Valluru

9/17/17

CS4800 Algorithms and Data

Professor Or Beit Aharon

Homework 2

* 1. The recurrence relation is:

And the solution would be:

.

To prove that, I will have to first prove the base case and then the

recursive case.

Base Case:

Induction Step:

We will use strong induction and assume is true for all

. Prove true for n. Also, we will make n > 1.

Now, if we plug in and into my solution, , we should get

the same answer as if we had plugged them into T(n).

The original recursion:

Using the solution: and

Substitute in those values for the ceiling and floor:

Factor out the 2:

There is a theorem that states that the floor and ceiling of any number

equals n, so if we apply it here, the equation becomes:

Simplify:

The induction shows that is indeed a formula that is valid for all

n.

* 1. The brute force algorithm, which would function as:
* Let the first element in an array be set as the minimum
* Compare the minimum to the second element and set the new minimum based on which is lower in value
* Compare this new minimum to the third element
* Repeat until you reach the end of the array

This would result in operations, which means that the divide and conquer algorithm found in part a is , so the brute force algorithm would be computationally less expensive.

{

* 1. Assuming for simplicity that , we obtain the following recurrence for the number of element comparisons T(n):

* 1. The brute force algorithm, which would function as:
* Let the first element in an array be set as the minimum
* Compare the minimum to the second element and set the new minimum based on which is lower in value
* Compare this new minimum to the third element
* Repeat until you reach the end of the array
* Start over and let the second element in the array be set as the minimum
* Do the same thing as with the minimum
* Repeat until the end of the array

This would result in operations, which means that the divide and conquer algorithm found in part a is , so the bru

te force algorithm would be computationally more expensive.

1. Strassen’s algorithm uses multiplications to multiply an matrix.

The algorithm that the student found would then perform multiplications to multiply an matrix.

Using theta-notation to express the estimate, we would get: .

1. This is the algorithm:
   1. Find(A)
      1. Maximal = 0
      2. FirstIndex = 0, SecondIndex = 0
      3. Let a = 1, b = 2
      4. While (b <= A.size())
         1. If (A[a] – A[b]) > maximal {maximal = (a - b), FirstIndex = a,

SecondIndex = b}

* + - 1. Else { a + 1 & b + 1 }
    1. Let x = A.size(), y = A.size() – 1
    2. While (y >= 0)
       1. If (A[x] – A[y]) > maximal {maximal = (x - y), FirstIndex = x, SecondIndex = y}
       2. Else { x – 1 & y – 1 }
    3. Return FirstIndex, SecondIndex

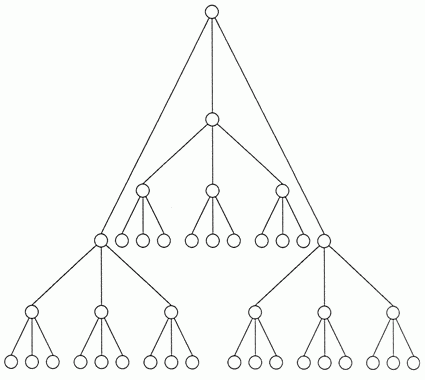
Because this algorithm does not recurse, its runtime will be . This is because , as it is multiplied by . Therefore, .

1. The comparison algorithm requires that the two arrays A and B must be sorted first.

The running time of this algorithm is .

The worst-case running time for this comparison algorithm is .

1. To better visualize this problem, imagine a ternary tree:



The maximum number of results this would have is , while the number of edges there could possibly be would be .

This would simplify to: , which would then simplify to .

Asymptotically, this would be equal to .

This may seem random, but I promise this is on topic:

With a binary tree, the maximum number of results is , while the number of edges there could possibly be would be .

This would simplify to: , which would then simplify to .

Now, .

Asymptotically speaking, , and therefore .

, so we can say that .

As result, we can say that the algorithm cannot do better than .