CSCI 570 Homework 3

Due Sunday Mar. 08 (by 23:30)

For all divide-and-conquer algorithms follow these steps:

1. Describe the steps of your algorithm in plain English.
2. Write a recurrence equation for the runtime complexity.
3. Solve the equation by the master theorem.

For all dynamic programming algorithms follow these steps:

1. Define (in plain English) subproblems to be solved.
2. Write the recurrence relation for subproblems.
3. Write pseudo-code to compute the optimal value
4. Compute its runtimecomplexity in terms of the input size.

1. Suppose we define a new kind of directed graph in which positive weights are assigned to the vertices but not to the edges. If the length of a path is defined by the total weight of all nodes on the path, describe an algorithm that finds the shortest path between two given points A and B within this graph.

Solution:

We have to reduce a vertex weighted graph G into an edge weighted graph G1. Create a new graph G1 as follows. Split each vertex v in G into two vertices vin and vout, with an edge weight between them equals to the vertex weight in G1. And the original edges from G have weights as 0 in G1. Edges coming to v in G will come to vin in G1. Edges leaving v in G will leave vout in G1. Run Dijkstra's algorithm. In the shortest path tree T1 for G1, an edge between vin and vout means that vertex v is in the shortest path tree T for G. Collapse all such edges (vin, vout) in T1 by merging vin and vout. This will create the shortest path tree T.

Rubric: (10 points)

- (10 points) The algorithm is stated clearly. And the algorithm is correct.

2. For each of the following recurrences, give an expression for the runtime $T(n)$ if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.
   - $T(n) = 3T(n/2) + n^2$
   - $T(n) = 4T(n/2) + n^2$
   - $T(n) = T(n/2) + 2^n$
   - $T(n) = 2^n T(n/2) + n^n$
   - $T(n) = 16T(n/4) + n$
   - $T(n) = 2T(n/2) + nlogn$
   - $T(n) = 2T(n/4) + n^{0.51}$
   - $T(n) = 0.5T(n/2) + 1/n$
   - $T(n) = 16T(n/4) + n!$
   - $T(n) = 10T(n/3) + n^2$

Solution:

- $T(n) = 3T(n/2) + n^2$ — case 3: $T(n) = \Theta(n^2)$

- $T(n) = 4T(n/2) + n^2$ — case 2: $T(n) = \Theta(n^2 log n)$
- $T(n) = T(n/2) + 2^n$ — case 3: $T(n) = \Theta(2^n)$
- $T(n) = 2^n T(n/2) + n^n$ — Does not apply, a is not constant
- $T(n) = 16T(n/4) + n$ — case 1: $T(n) = \Theta(n^2)$
- $T(n) = 2T(n/2) + nlogn$ — case 2: $T(n) = \Theta(nlog^2 n)$
- $T(n) = 2T(n/4) + n^{0.51}$ — case 3: $T(n) = \Theta(n^{0.51})$
- $T(n) = 0.5T(n/2) + 1/n$ — Does not apply ($a < 1$)
- $T(n) = 16T(n/4) + n!$ — case 3: $T(n) = \Theta(n!)$
- $T(n) = 10T(n/3) + n^2$. — case 1: $T(n) = \Theta(n^{log_3 10})$

Rubric: (10 points)

- (10 points) 1 point for each item. One for getting the case right and one for the runtime/justification of case

3. Suppose that we are given a sorted array of distinct integers $A[1, ..., n]$ and we want to decide whether there is an index i for which $A[i] = i$. Describe an efficient divide-and-conquer algorithm that solves this problem and explain the time complexity.

Solution:

If the array has just one integer, then we check whether $A[1] = 1$ with one comparison. Otherwise divide the list into two parts, the first half and the second half, as equally as possible. Consider the largest element $A[m]$ of the left half. We compare $A[m]$ with m. If $A[m] = m$, the answer is yes and we are done. If $A[m] > m$, then we can throw away the right half and continue recursively in the left half. Indeed, then for every integer $k \geq 0$ using the fact that the integers are distinct and sorted, $A[m + k] \geq A[m] + k > m + k$. If $A[m] < m$, then we can throw away the left half and continue recursively in the right half. Indeed, then for every integer $k \geq 0$ using the fact that the integers are distinct and sorted $A[m - k] \leq A[m] - k < m - k$. Thus for the number of comparisons we get the following recursion, $T(n) = T(n/2) + O(1), T(1) = O(1)$. According to Master Theorem, the time complexity is $\Theta(\log n)$.