CSE 101- Winter ‘16
Discussion Section
January 18th 2016
Topics

• Master theorem examples
• Divide and conquer examples
• Introduction to shortest paths and Breadth-first search
Master Theorem

- \( T(n) = aT\left(\frac{n}{b}\right) + O\left(n^d\right) \)
  - \( T(n) = O\left(n^d\right) \) if \( d > \log_b a \)
  - \( T(n) = O\left(n^d \log(n)\right) \) if \( d = \log_b a \)
  - \( T(n) = O\left(n^{\log_b a}\right) \) if \( d < \log_b a \)

- \( T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 14 \)
- \( T(n) = 3T\left(\frac{n}{9}\right) + n^{0.51} \)
- \( T(n) = 4T\left(\frac{n}{2}\right) + n \)
Master Theorem: Examples

• \( T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 14 \)
  
  \(- O(\sqrt{n} \log(n)) \)

• \( T(n) = 3T\left(\frac{n}{9}\right) + n^{0.51} \)
  
  \(- O(n^{0.51}) \)

• \( T(n) = 4T\left(\frac{n}{2}\right) + n \)
  
  \(- O(n^2) \)
Divide and Conquer example

- (DPV textbook, problem 2.17) Given a sorted array of distinct integers $A[1,...,n]$, you want to find out whether there is an index $i$ for which $A[i] = i$. Give a divide-and-conquer algorithm that runs in time $O(\log n)$
Divide and Conquer example

• (DPV textbook, problem 2.17) Given a sorted array of distinct integers $A[1,...,n]$, you want to find out whether there is an index $i$ for which $A[i] = i$. Give a divide-and-conquer algorithm that runs in time $O(\log n)$

• Hint: Think binary search
Naive Solution

- Compare each number to its index
- $O(n)$
Divide and Conquer Solution

• Use the fact that the array is sorted and it contains distinct integers
Divide and Conquer Solution

• Use the fact that the array is sorted and it contains distinct integers
• If $A[j] = j$
Divide and Conquer Solution

• Use the fact that the array is sorted and it contains distinct integers
• If \( A[j] = j \)
• If \( A[j] > j \)
  – As you move to the right, the index increases by 1 but the corresponding array integer must also increase by at least 1 (since they are distinct).
  – Therefore, the index can never “catch up” with the array entry. Hence, we can eliminate right half
  – Look in \( A[1, \ldots, j-1] \)
Divide and Conquer Solution

- Use the fact that the array is sorted and it contains distinct integers
- If $A[j] = j$
- If $A[j] > j$
  - As you move to the right, the index increases by 1 but the corresponding array integer must also increase by at least 1 (since they are distinct).
  - Therefore, the index can never “catch up” with the array entry. Hence, we can eliminate right half
    - Look in $A[1, \ldots, j-1]$
- Similarly, if $A[j] > j$
  - Look in $A[j+1, \ldots, n]$
Pseudocode

Matching_index(A[], first, last)

if (first > last) return not_found

mid = (first + last)/2

if (mid == A[mid]) return mid

if (mid > A[mid])
    return Matching_index(A[], key, first, mid-1)

if (mid < A[mid])
    return Matching_index(A[], key, mid+1, last)
Runtime Analysis

• Break problem into one subproblem of size n/2.
• No recombination (like mergesort), but for every subproblem you need to check a constant number of equalities and inequalities

• \( T(n) = T(n/2) + O(1) \)

• \( T(n) = O(\log n) \)
Proof of algorithm termination

• Intuition:
  – Show that the size of subproblem reduces by at least 1 each recursive call
  – Will eventually reach base case

• By induction:
  – Show that input of size 1 terminates
  – Assume it terminates for size $k \leq n$
  – Show that it terminates for size $n+1$
Proof of correctness

• Looking in correct branch:
  – As you move to the right, the index increases by 1 but the corresponding array integer must also increase by at least 1 (since they are distinct).
  – Therefore, the index can never “catch up” with the array entry if $A[mid] > mid$. Hence, we can eliminate right half of the array
  – Similarly, if $A[mid] < mid$ …

• Formal proof by induction
Example 2 – Problem statement

• For integers \( a < b \) an interval \([a,b]\) is the set of integers \( \{a, a+1, \ldots, b\} \)

• Suppose you are given a set of \( n \) intervals \( \{[x_1, y_1], \ldots, [x_n, y_n]\} \)

• Show how you can find the greatest overlap between any 2 intervals in \( O(n \log n) \) time - (where size of overlap between 2 intervals = number of common integers)
Solution- preprocessing

• How do we divide?

• Sort intervals by left end point!
  – One time cost of $O(n \log n)$
Solution - divide and conquer

• Divide sorted intervals into 2 sets – left half and right half
• Recursively find greatest overlap in these two halves
• We are done!

• Combine / Merge / Cross overlaps?
Solution - combine

• Find Max overlap between an interval in the left half and right half

• Pick an appropriate interval from left half
  – Largest right end point!

• Check with all intervals in second half and find greatest overlap

• Merge takes O(n) time – why is this important?
Pseudocode

(I_1, I_2, ..., I_n) <- sort intervals by lower bound (x_i)

greatest_overlap(I_1, I_2, ..., I_n)

a = greatest_overlap(I_1, I_2, ..., I_n/2) //first half
b = greatest_overlap(I_n/2+1, ..., I_n) //second half
c = greatest_cross_overlap(first half of intervals, second half of intervals) //"merge"
return max (a,b,c)

greatest_cross_overlap(first half of intervals, second half of intervals)

Imax <- find interval in first half with highest right end point
max = 0
For each interval I in second half:
    if (overlap(I,Imax) > max): max = overlap(I,Imax)
return max
Depth-First Search: What it Does

• Identifies all vertices of a graph reachable from a starting point.

• Identifies a path from a starting vertex in the graph to any other vertex in the graph, if a path exists.

• What it does not: find a short(est) path
Shortest Path: Definitions

• Number of edges from $s$ to $t$.
  – Algorithm: Breadth-First Search

• Sum of edge weights of all edges in path from $s$ to $t$.
  – Algorithms: Dijkstra, Bellman-Ford
Breadth-First Search

- Proceeds “layer by layer”.
- First visit vertices at distance \(d\) from starting vertex, then vertices at distance \(d+1\), and so on.
Breadth-First Search: Pseudo-Code

procedure bfs(G, s)

for all u ∈ V:
    dist(u) = ∞

dist(s) = 0
Q = [s] (queue containing just s)
While Q is not empty:
    u = eject(Q)
    for all edges (u, v) ∈ E:
        if(dist(v)) = ∞
            inject(Q,v)
        dist(v) = dist(u) + 1

(Ref: Algorithms, Fig 4.3)