

---

# HEAD-BANGING SESSION 1 SOLUTIONS

## FALL 2008 CS 473: ALGORITHMS

---

### Problem 1. [Recurrences That Defy the Master Method]

1.  $A(n) = A(5n/8) + A(3n/8) + n$

Here are some observations you want to make when you are drawing your recursion tree. Each complete level sums to  $n$ . The tree is lopsided with different levels for different leaves. The shallowest leaf is at depth  $\log_{5/3} n$ . The deepest leaves are at depth  $\log_{8/3} n$ . If you imagine complete trees at those levels, then  $n \log_{5/3} n < A(n) < n \log_{8/3} n$ . Therefore  $A(n) = \Theta(n \log n)$ .

2.  $B(n) = B(5n/8) + B(n/4) + n$

This tree is again lopsided with the sum of the levels terms in a descending geometric series. You can derive an upper bound  $O(n)$  by growing the tree out to an infinite depth. Your lower bound is given by the first node. Therefore,  $B(n) = \Theta(n)$ .

3.  $C(n) = \sqrt{n}C(\sqrt{n}) + n$

The nodes of this tree sum to  $n$  in any level. The depth,  $L$  satisfies  $n^{2^{-L}} = 2$  as shown in the lecture notes. The number of levels multiplied by the work at each level, sums this tree to  $C(n) = \Theta(n \lg \lg n)$ .

4.  $D(n) = 8\sqrt{n}D(\sqrt{n}) + n$

The sum of level  $i$  in this tree is  $8^i n$ . This is an increasing geometric series so  $D(n)$  will be dominated by the sum over the deepest level. Like like previous problem, the depth of this tree is  $n \lg \lg n$ . Therefore,  $D(n) = \Theta(8^{n \lg \lg n}) = \Theta(n \log^3 n)$ .

### Problem 2. [Using Recurrences]

- We will claim that we can establish a bijection between the set of binary string of length  $(i + j)$  with  $i$  1s and  $j$  0s to the set of paths from  $(0, 0)$  to  $(i, j)$ . The number of binary strings of length  $(i + j)$  with  $i$  1s and remaining 0s is  $\binom{i+j}{i}$ . Consider a string  $s$  with  $i$  1s and  $j$  0s. We turn right when we see 1 and go up when we see 0. This gives us a path from  $(0, 0)$  to  $(i, j)$ . It is easy to see that two different strings give two different paths. Now, consider any path from  $(0, 0)$  to  $(i, j)$ , there exists a binary string corresponding to it with  $i$  1s and  $j$  0s. Hence we have a bijection between the two sets.
- Consider any path from  $(0, 0)$  to  $(i, j)$ . The last step is either right or up. So,  $A(i, j) = A(i - 1, j) + A(i, j - 1)$ .
- The expression on left hand side is the total number of paths where you take  $n$  steps (and reach anywhere you want to). There are 2 choices for every step, namely up and right. Hence it is equal to  $2^n$ .
- Consider any path from  $(0, 0)$  of length  $(n - 1)$ . Suppose it ends at  $(i, j)$ . If that path is extended by moving up or by moving right, we get two paths whose  $x$  coordinates are of opposite parity. Consider any path of length  $n$ . Replacing the last right move by up move or vice versa, we obtain paths of length  $n$  which end up at  $x$  coordinates which are of opposite parity. Using these two results, we can obtain the desired result.

- Consider any dominant path  $p$  from  $(0, 0)$  to  $(n, n)$  (we can call it a dominant path of length  $2n$ ). Let  $S_p$  be the set of points of the form  $(i, i)$  in the path  $p$ . Let  $s_p$  be the minimum  $x$  coordinate element of  $S_p \setminus \{(0, 0)\}$ .  $s_p$  is well defined because  $p$  at least touches  $(n, n)$ . Intuitively,  $s_p$  is the first position where  $p$  touches the  $x = y$  line after  $(0, 0)$ .

The first move in path  $p$  has to be a right move. The last move before it reaches  $(s_p, s_p)$  has to be up move. The path from  $(1, 0)$  to  $(s_p, s_p - 1)$  is a dominant path of length  $2(s_p - 1)$ . The remainder of  $p$  after  $(s_p, s_p)$  to  $(n, n)$  is a dominant path of length  $2(n - s_p)$ .

Showing the relation other way around is trivial (this is needed to show bijection). Hence we get that recurrence to be:

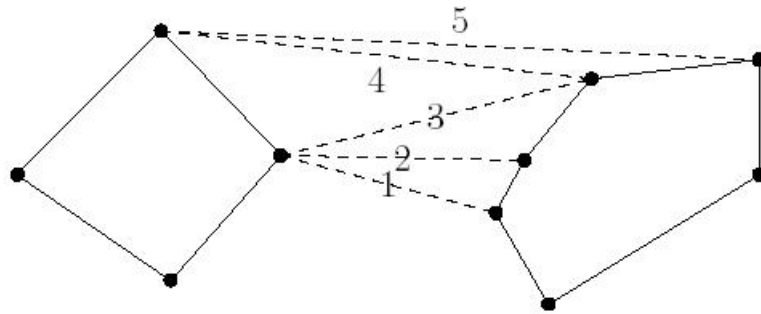
$$B(n) = \sum_{i=0}^{n-1} B(i)B(n - i - 1) \tag{1}$$

**Problem 3. [Convex Hulls]**

1.  $O(n)$  time algorithm that computes the convex hull of  $P1 \cup P2$ .

We start with disjoint hulls  $P1$  and  $P2$ . Define a bridge as any line segment joining a vertex on  $P1$  to a vertex on  $P2$  that does not cross inside of either polygon. What we need to do is turn this bridge into upper and lower bridges that when added to the two hull's outside edges form a new convex hull. The following produces the upper bridge, you can then produce the lower in a similar fashion.

- (a) Start with any bridge.
- (b) Keeping the  $P1$  end of the bridge fixed, see if the  $P2$  end can be raised. That is, look at the next vertex on  $P2$  polygon going clockwise, and see whether that would be a (better) bridge. Otherwise, see if the left end can be raised while the right end remains fixed.
- (c) If made no progress in (b) (cannot raise either side), then stop else repeat (b).



Convince yourself that this algorithm will indeed terminate. Convince yourself that finding the upper bridge takes at most  $O(n)$  time. This implies that then entire merge algorithm takes  $O(n)$  time.

2.  $O(n \log n)$  time divide-and-conquer algorithm to compute the convex hull

With a linear time merger from the first part, the divide and conquer strategy is straightforward. First, sort the points by their  $x$  coordinate. Divide the  $n$  points into two halves. Recursively find convex hulls of each subset. Combine the two subhulls into an overall convex hull by using the algorithm from part 1. Notice that because of the way we divided the points, the two subhulls are clearly disjoint. Also, notice that a perfect bridge between two subhulls is the edge from the rightmost vertex of the left hull to the leftmost vertex of the right hull. The sorting requires  $O(n \log n)$  time and the D&C algorithm runs in  $T(n) = 2T(n/2) + O(n) = \Theta(n \log n)$ .