Fall 2018, CMPSC 465: Exam 2.
Closed book and closed notes, no 'cheat sheet', no calculators allowed.
Please don't use cell phones during the exam.
Answer questions in the space provided.
The exam is for 40 points (4 problems, 10 points each).

NAME:

---

1. A binary tree is full if all of its vertices have either zero or two children. Let B_n denote the number of full binary trees with n vertices. For general n, give a recurrence relation for B_n. Briefly explain how you arrived at this recurrence. Using the recurrence, give the values for B_8 and B_9.

Solution:

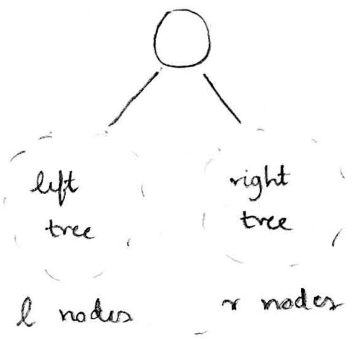
$B_K = 0$ for even k, since a full binary tree must have an odd number of nodes.

+2 (root + an even number at each succeeding level)

+2 $B_1 = 1$ by inspection, $B_3 = 1$, and $B_5 = 2$.

Consider an odd $n > 1$. We can recursively construct full binary trees starting from the root. Fixing the root, we have $n-1$ nodes remaining.

+3



We have $l + r = n-1$

$l = 0, r = n-1 \Rightarrow B_0$ left subtree possibilities, B_{n-1} right subtree possibilities

$l = 1, r = n-2 \Rightarrow B_1$ " , B_{n-2} "

$l = n-1, r = 0 \Rightarrow B_{n-1}$ " , B_0 "

+1 The total number of n-node full binary trees is $\sum_{l=0}^{n-1} B_l B_{(n-1)-l}$.

+2

$B_8 = 0$ $\quad B_9 = 2(B_1 B_7 + B_3 B_5) = 2(5+2) = 14$

$B_7 = 2(B_1 B_5) + B_3^2 = 2(2) + 1 = 5$

2. An array $A[1 \ldots n]$ is said to have a *majority element* if more than half of its entries are the same. Given an array, design an $O(n \log n)$ divide-and-conquer algorithm to determine if the array has a majority element, and, if so, to find that element.

Solution:

Approach 1

Split the array into two halves, A_1 and A_2.

(+1) The majority element satisfies the following property:

(+3) If A has a majority element v, v must also be a majority element of A_1 or A_2 or both.

Pseudocode

MAJ-ELEMENT (A)

(+4)

1. Split A into two halves A_1 and A_2. If $|A_1| = |A_2| = 1$, return the element as majority element.
2. $T(n/2) \leftarrow$ $u \leftarrow$ MAJ-ELEMENT (A_1)
3. $T(n/2) \leftarrow$ $v \leftarrow$ MAJ-ELEMENT (A_2)
4. Compare u to elements in A_2 and get total count of u in A.
5. Compare v to elements in A_1 and get total count of v in A.

$O(n)$ (steps 4–5)

6. Check if either count of u in $A > n/2$ or count of v in $A > n/2$.
7. A either has a majority element (either u or v), or does not.

(+2) $\longrightarrow$ $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$.

Approach 2: Use a comparison-based sort, assuming comparisons are possible. $(<, >, =)$

Use Mergesort (divide-and-conquer), taking $O(n \log n)$ time. (5)

(8)

Scan sorted array to see if an element occurs consecutively more than $n/2$ times. $\rightarrow O(n)$ time. (3)

3. Consider the task of searching a sorted array $A[1 \ldots n]$ for a given element x: a task we usually perform by binary search in time $O(\log n)$. Show that any algorithm that accesses the array only via comparisons must take $\Omega(\log n)$ steps.

Solution:

Any comparison-based algorithm for search in a sorted array can

+1 be represented as a binary tree.

A path from the root to a leaf represents an execution of the algorithm: at every node, a comparison takes place, and according +2 to its result, a new comparison is performed.

+1 A leaf represents an output of the algorithm.

All possible indices must appear as leaves, or the algorithm will fail +3 when one of the missing indices is an output.

Hence, the tree must have at least n leaves, implying that its +3 depth must be $\Omega(\log n)$, since a binary tree of height h has at most 2^h leaf nodes.

4. The square of a matrix A is its product with itself, AA.

   (a) Show that five multiplications are sufficient to compute the square of a 2×2 matrix.

   (b) Is the following a valid algorithm for computing the square of an $n \times n$ matrix? If yes, briefly illustrate its working. If not, explain why.
   "Use a divide-and-conquer approach as in Strassen's algorithm, except that instead of getting 7 subproblems of size $n = 2$, we now get 5 subproblems of size $n = 2$ thanks to part (a). Using the same analysis as in Strassen's algorithm, we can conclude that the algorithm runs in time $O(n^{\log_2 5})$."

Solution:

(a) Consider $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$.

$a_{12}(a_{11} + a_{22})$

$$A^2 = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} a_{11}^2 + a_{12}a_{21} & a_{11}a_{12} + a_{12}a_{22} \\ a_{21}a_{11} + a_{22}a_{21} & a_{21}a_{12} + a_{22}^2 \end{pmatrix}$$

$= a_{21}(a_{11} + a_{22})$

6

The five multiplications required are a_{11}^2, a_{22}^2, $a_{12}a_{21}$, $a_{12}(a_{11}+a_{22})$, and $a_{21}(a_{11}+a_{22})$.

(b) This does not hold for submatrices.

$A_{12}A_{21} \neq A_{21}A_{12}$ (not commutative)

4

$A_{21}A_{11} + A_{22}A_{21} \neq A_{21}(A_{11} + A_{22})$