Theory Problems: Batch 1

Course: Coursera Algorithms Specialization
Name: Bryan Yaggi

Problem 1

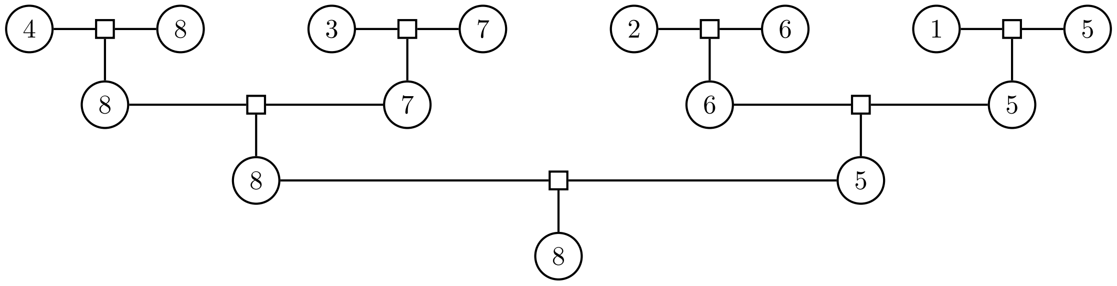
You are given as input an unsorted array of n distinct numbers, where n is a power of 2. Give an algorithm that identifies the second-largest number in the array, and that uses at most $n + \log_2 n - 2$ comparisons.

The algorithm consists of 2 main steps.

1. Compare pairs of consecutive elements until finding the largest. For each element, store a list containing elements with which it was compared.
2. Find the largest element in the list of elements compared with the largest element. The result is the second largest element.

Step 1 requires $n - 1$ comparisons. Step 2 requires $\log_2 n - 1$ comparisons. The total is $n + \log_2 n - 2$ comparisons.

Example: Given $[4, 8, 3, 7, 2, 6, 1, 5]$.



The second largest element is 7, which is the largest of $[4, 7, 5]$.

Problem 2

You are a given a *unimodal* array of n distinct elements, meaning that its entries are in increasing order up until its maximum element, after which its elements are in decreasing order. Give an algorithm to compute the maximum element that runs in $O(\log n)$ time.

Treat the array as a binary tree with the root as the middle element. Recursively, the middle elements of the resulting subarrays are the next nodes in the tree. Use binary search checking the value of the middle node with its neighbors until the peak is found.

Problem 3

You are given a sorted (from smallest to largest) array A of n distinct integers which can be positive, negative, or zero. You want to decide whether or not there is an index i such that $A[i] = i$. Design the fastest algorithm that you can for solving this problem.

Again, treat the array as a binary tree. Use binary search to find the solution. If $A[i] < i$, go right. If $A[i] > i$, go left. The algorithm will run in $O(\log n)$ time.

Problem 4

You are given an n by n grid of distinct numbers. A number is a local minimum if it is smaller than all of its neighbors. (A neighbor of a number is one immediately above, below, to the left, or the right. Most numbers have four neighbors; numbers on the side have three; the four corners have two.) Use the divide-and-conquer algorithm design paradigm to compute a local minimum with only $O(n)$ comparisons between pairs of numbers. (**Note**: since there are n^2 numbers in the input, you cannot afford to look at all of them. **Hint**: Think about what types of recurrences would give you the desired upper bound.)

The algorithm consists of 2 recursive steps. We will find a local minimum, but not necessarily the global minimum.

1. Take the middle row, middle column, and border elements of the 2D array. Find the minimum element of this set.
2. Find the minimum of the element found in Step 1 and its neighbors. If the minimum is not the element found in Step 1, select the quadrant in which the minimum element lies and recurse.

Step 1 requires $6n - 9$ comparisons. Step 2 requires 2 comparisons. The total is $6n - 7$ for each recursion. Using the master method, $T(n) = aT(\frac{n}{b}) + O(n^d) = T(\frac{n}{2}) + O(n) \therefore O(n)$

Example: