CS624: Analysis of Algorithms

Midterm exam 2 – practice

Instructor – Nurit Haspel

1. **Medians and Order Statistics:** (30%) Given two *sorted* arrays, A and B, each of size n. Describe an $O(\log n)$ worst case algorithm to find the median of all the 2n elements in A and B. Provide a brief but accurate runtime analysis (**Hint 1:** It's not unlike binary search... **Hint2:** Use the medians of A and B to guide you)

   First of all - locate the medians of A and B. It can be done in O(1) since the two arrays are sorted. If $median[A] = median[B]$ return one of them as the median. If $median(A) < median(B)$, then the median of both arrays is certainly not in $A[1..n/2-1]$ (the lower half of A), since all the elements there are smaller than more than half of A and more than half of B (see why?). Similarly, the median of both arrays is certainly not in $B[n/2+1..n]$ (the larger half of B), for the same reason. You can then recursively search the median in the larger half of A and the smaller half of B. It's a logarithmic time algorithm.

2. **Binary search trees:**

   (a) (15%). Let x be a leaf node in a binary search tree T. Let y be x's parent. Show that y.key is either the smallest key in T larger than x.key or the largest key in T smaller than x.key.

   This is a special case of the successor-predecessor relationship. Since x is a leaf, its successor and predecessor (if exist) have to be ancestors. Let's assume, without loss of generality, that y, x's parent, is larger than x. Therefore x is y's left child. According to what we saw in class, y's predecessor is the largest node on y's left subtree, in this case it's x since x is a leaf and therefore the only node on y's left subtree. Now, assume by contradiction that this were not the case and there is a node z such that $x.key < z.key < y.key$. z must be on the same subtree as x and y, since x and y are on the same subtree, and z's value is between them. Since y only has x as a leaf, z can only be an ancestor of y, not a descendant. But this is impossible, because that would put both x and y on the same subtree of z, despite the fact that $x.key < z.key < y.key$. Therefore, such z cannot exist. The same is true in reverse if $y < x$.

   (b) (15%) Is the following claim true or false? Explain: in order to determine whether two binary search trees are identical one has to perform an in-order walk on them and compare the results.

   False, since inorder gives the keys in a sorted order. So - any two BSTs with the same set of keys but a different structure gives the same inorder.

3. **Dynamic Programming:** Given an array A of n numbers, the maximum subarray problem is the task of finding the contiguous subarray A[i..j] of numbers which has the largest sum. For example, if $A = \{-2, 1, -3, \mathbf{4}, \mathbf{-1}, \mathbf{2}, \mathbf{1}, -5, 4\}$ then the subarray that gives the maximum sum is $\{4, -1, 2, 1\}$ with sum 6 (emphasized in bold font). Let us define MS(i) as the maximum sum subarray that ends at A[i] (and must include A[i]). For example, in a 1-based index, – MS(1) $=\{-2\}$. MS(2) = $\{1\}$ (since concatenating -2 and 1 gives a smaller sum, so MS(2) includes only A[2]). In other words – for MS(i) we ask ourselves which one is better – for A[i] to extend MS(i-1) or be its own subarray.

(a) Show that the problem has the optimal substructure (Hint: A[i] either extends the maximum sub-array that ends in A[i-1] or alternatively, includes only A[i] itself. Use a cut-and-paste argument for MS(i-1) with respect to MS(i)).

It is a standard cut-paste. If we look at an optimal solution MS(i), then either A[i] extends a maximum sub-array ending in $A[i-1]$ or not. If there were a better MS(i-1), we could have replaced it and get a better MS(i) or at least not a worse one, in case A[i] is in itself MS(i)

(b) Define a recursive algorithm that calculates MS(i), that can be used as a basis for a dynamic programming calculation. Remember to also return the overall maximum sum. It doesn't have to be MS(n) (why?).

The algorithm is: $MS(i) = \begin{cases} -\infty & \text{if } i < 1 \\ \max(MS(i-1) + A[i], A[i]) & \text{otherwise} \end{cases}$ The $-\infty$ is to ensure that MS(1) = A[1] even if it is negative. The end result is $\max_i\{MS(i)\}$, which does not have to be MS(n), as in the example here, since the subsequence can end at some point in the middle of the array. We can then convert the recursive algorithm into DP by maintaining an array MS[] such that MS[i] = MS(i) for each i.

(c) Based on that, calculate MS(i) for every index in the array above.

$MS = \{-2, 1, -2, 4, 3, 5, 6, 1, 5\}$