Analysis of Algorithms - Midterm (Solutions)

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Problems

1. **Asymptotics:**

   (a) Show that $[(f(n) \in \Omega(g(n))) \wedge (g(n) \in \Omega(h(n)))] \Rightarrow f(n) \in \Omega(h(n))$.

   (b) Does $\log^3 n \in O(n^{0.5})$?

   Solution:

   (a) The hypothesis states that $f(n) \in \Omega(g(n))$, from which it follows that $f(n) \geq c_1 \cdot g(n)$, for some $c_1 > 0$ and all $n \geq N_1$. Likewise, we have $g(n) \in \Omega(h(n))$, which implies that $g(n) \geq c_2 \cdot h(n)$, for some $c_2 > 0$ and all $n \geq N_2$. Observe that for all $n \geq N_3 = \max(N_1, N_2)$, we have $f(n) \geq c_1 \cdot g_n$ *and* $g(n) \geq c_2 \cdot h_n$ and therefore, $f(n) \geq c_1 \cdot c_2 \cdot h(n)$. Putting $c_3 = c_1 \cdot c_2$, we get $f(n) \geq c_3 \cdot h(n)$, for all $n \geq N_3$, where $c_3 > 0$. In other words,

   $$[(f(n) \in \Omega(g(n))) \wedge (g(n) \in \Omega(h(n)))] \Rightarrow f(n) \in \Omega(h(n))$$

   (b) Observe that

   $$\begin{aligned} \lim_{n\to\infty} \frac{\log^3 n}{n^{0.5}} &= \lim_{n\to\infty} \frac{3\log\log n}{0.5\log n} \\ &= c_1 \cdot \lim_{n\to\infty} \frac{\log\log n}{\log n} \\ &= c_1 \cdot \lim_{n\to\infty} \frac{\frac{1}{\log n} \cdot \frac{1}{n}}{\frac{1}{n}} \\ &= c_1 \cdot \lim_{n\to\infty} \frac{1}{\log n} \\ &= 0 \end{aligned}$$

   It follows that $\log^3 n \in O(n^{0.5})$.

   □

2. **Algorithm Design for order:** Given an integer array of n elements, design an algorithm to find both the maximum element and the minimum element, using at most $\frac{3n}{2}$ element to element comparisons. Comparisions for iterators (e.g., **for** loops) do not count.

   Solution: Without loss of generality, we assume that the number of elements is even, i.e., $n = 2k$, for some integer k.

   Break up the array elements into k pairs of 2 elements each. In each pair, perform a comparison and make the larger element, the left element of the pair. Note that k element to element comparisons are performed to achieve this goal.

Let $(a_1, b_1),\ (a_2, b_2)\ \ldots (a_k, b_k)$ denote the elements; observe that as per the above comparison operations, $a_i \geq b_i$, $\forall i = 1, 2, \ldots, k$.

The crucial observation is that to find the maximum element, we can focus purely on the a elements. To see this, let us assume that the maximum element in the array is one of the b elements, say b_j; but $a_j \geq b_j$ and our claim holds. Since there are k elements of type a, the maximum among them can be found using at most $k-1$ element to element comparisons. Using a similar argument, the minimum element in the array can be found by focussing purely on the b elements, and this can be achieved using another $k-1$ element to element comparisons.

Thus, the total number of element to element comparisons is $k + (k-1) + (k-1) = 3k - 3 \leq 3k = \frac{3 \cdot n}{2}$. □

3. **Binary Search Trees:** Enumerate all the binary search trees on the keys 1, 2 and 3.

   Solution: There are five distinct binary search trees:

   (i) 2 at the root, 1 as its left child and 3 as its right child.

   (ii) 3 as the root, 1 as the left child of 3 and 2 as the right child of 1.

   (iii) 3 as the root, 2 as the left child of 3 and 1 as the left child of 2.

   (iv) 1 as the root, 2 as the right child of 1 and 3 as the right of 2.

   (v) 1 as the root, 3 as the right child of 1 and 2 as the left child of 3.

   □

4. **Sorting:** Explain briefly how Randomized Quicksort performs $O(n \cdot \log n)$ comparisons, in the expected case, to sort an array of n elements. (You may assume the algorithm discussed in class.)

   Solution: As per the algorithm in class, the pivot is chosen uniformly and at random from the array. The elements of the array are then partitioned about this pivot element and the algorithm either terminates on a partition (if there is at most one element in it) or recurses on it (if there are at least two elements in it).

   Consider the computation tree created by a run of Randomized Quicksort on an array of n elements. The root of the tree has n elements and the leaves have exactly one element each. A node in this tree is called "good", if *both* its children have size at most $\frac{3}{4}^{th}$ of its size.

   Claim 1.1 *A root to leaf path cannot have more than* $\log_{\frac{4}{3}} n$ *good nodes.*

   Proof: Let L be a good node and let L' denote one of its two children. We must have $|L'| \leq \frac{3}{4}|L|$. Accordingly, if there are k good nodes on a root to leaf path, we have $(\frac{3}{4})^k \cdot n \leq 1$, which implies that $k \leq \log_{\frac{4}{3}} n$. □

   Claim 1.2 *The expected height of the computation tree is* $O(\log n)$.

   Proof: The expected height of the computation tree is the expected length of a randomly chosen root to leaf path. Focus on a specific root to leaf path of the computation tree, say p. At any node on p, Randomized Quicksort picks a pivot, uniformly and at random over all the elements in that node. Therefore, probability that a node is good is at least $\frac{1}{2}$. Now the expected number of nodes on p (which is the expected length of p)is equal to the number of recursive invocations made, before $\log_{\frac{4}{3}} n$ good nodes are created. However, for $\log_{\frac{4}{3}} n$ good nodes to be created, at most $2 \cdot \log_{\frac{4}{3}} n$ recursive invocations need to be made, as per the theorem given in class. Hence, the expected height of the computation tree is $O(\log n)$. □

   Observe that the total number of comparisions in each level of the computation tree is $O(n)$ and hence the total number of comparisions performed by Randomized Quicksort is $O(n \cdot \log n)$, in the expected case. □

5. **Properties of Binary Trees:** Let T be a proper binary tree with n nodes and height h. Argue that the number of external nodes in T is at least $h + 1$ and at most 2^h.

   Solution: We use induction on the height h of the tree, for proving both bounds.

MINIMUM: Clearly, if the height h of T is 0, then T has precisely one node, i.e., the root. Further, this node is also an external node and thus the number of external nodes is at least $1 + 0 = 1$. It follows that the base case is true.

Assume that the minimum number of external nodes in a tree of height $h = k$ is $k + 1$.

Now consider a tree T of height $h = k + 1$; let us remove all the nodes at level k; we get a tree T' of height k and by the inductive hypothesis, there are at least $k + 1$ external nodes in T'. An external node of T is created when a node is attached to an external node of T'. To increase the level of T' by one, we need to attach at least two nodes to a node at level k; further both these attached nodes become external nodes; at the same time, the external node of T', to which these nodes were attached becomes an internal node. Thus, the total number of external nodes in T, is at least $k + 1 + 2 - 1 = k + 2$. By applying the second principle of mathematical induction, we can conclude that the total number of external nodes in a tree of height h is at least $h + 1$.

MAXIMUM: Once again, if the height h of T is 0, then T has precisely one node, i.e., the root. Further, this node is also an external node and thus the number of external nodes is at most $2^0 = 1$. It follows that the base case is true.

Assume that the maximum number of external nodes in a tree of height $h = k$ is 2^k.

Now consider a tree T of height $h = k + 1$; let us remove all the nodes at level $k + 1$; we get a tree T' of height k and by the inductive hypothesis, there are at most 2^k external nodes in T'. An external node of T is created when a node is attached to an external node of T'. We can attach at most 2 nodes to each external node of T' and thus the total number of external nodes in T cannot exceed $2 \cdot 2^k = 2^{k+1}$. By applying the second principle of mathematical induction, we can conclude that the total number of external nodes in a tree of height h is at most 2^h. □