Data Structures and Algorithms – (COMP SCI 2C03)
Winter 2021
Tutorial-II

January Feb 1, 2021

Question 1. Consider the Weighted Quick Union-Find approach mentioned in Section 1.5 of the textbook. Prove that the depth of any node x is at most $\log_2 N$ in the trees constructed using this approach.

Answer. (Proposition H., p. 229) We prove a stronger fact by (strong) induction: The height of every tree of size k in the forest is at most $\log k$. The base case follows from the fact that the tree height is 0 when k is 1. By the inductive hypothesis, assume that the tree height of a tree of size i is at most $\log i$ for all $i < k$. When we combine a tree of size i with a tree of size j with $i \leq j$ and $i + j = k$, we increase the depth of each node in the smaller set by 1, but they are now in a tree of size $i + j = k$, so the property is preserved because $1 + \log i = \log(i + i) \leq \log(i + j) = \log k$.

Question 2 Prove the basic combinatorial property of binary trees that a tree of height h has no more than 2^h leaves.

Answer. Base Case: $h = 0$. A binary tree of height 0 is just a single node with no children, and therefore has 1 leaf. $1 = 2^0$, so the base case satisfies the induction hypothesis (see below).

Induction Hypothesis: Suppose that for some $k \geq 0$, all binary trees of height $\leq k$ have at most 2^k leaves.

Induction Step: Let T be a binary tree of height $k + 1$. Then T's left and right subtrees are each binary trees of height <= k, and thus by the I.H. have at most 2^k leaves. The number of leaves in T is equal to the sum of the number of leaves in T's subtrees, which must be less than or equal to $2^k + 2^k = 2^{(k+1)}$. Hence the hypothesis holds for $k + 1$, and so the theorem is proved.

Question 3 If a full binary tree has N leaves. Then how many internal nodes does the tree have.

Answer. A Binary Tree is a full binary tree if every node has 0 or 2 children.

Suppose there are I internal nodes, each having 2 children, therefore total children in the tree is $2 \times I$.

There are $I-1$ internal nodes which are children of some other node (root has been excluded hence one less than the total number of internal nodes)

That is, out of these $2 \times I$ children, $I-1$ are internal nodes and therefore the rest $N = ((2 \times I)–(I-1))$ are leaves.

Hence $N = I + 1$ and if we have N leaves, then $I = N - 1$.

Question 4 Prove the recurrence $T(n) = T(n-1) + T(0) + cn$ is in $\Theta(n^2)$, assuming $T(0) = \Theta(1)$

Answer. We have $T(n) = T(n-1) + cn + 1$.

If we continue, $T(n-1) = T(n-2) + c \times (n-1) + 1$.

Also, $T(n-2) = T(n-3) + c \times (n-2) + 1$ and so on.

At the end, we have $T(2) = T(1) + 2c + 1$ and $T(1) = 1 + 1 + c$.

So, we have $T(n) = (1 + 1 + ... + 1) + (1 + 2 + 3 + ... + n)c$.

We know that $1 + 2 + ... + n = \dfrac{n(n+1)}{2}$.

So, we have $T(n) = n + \dfrac{n^2 + n - 2}{2} \times c$, which is $\Theta(n^2)$.

Question 5 Give traces, in the style of the trace given with Algorithm 2.3, showing how the sequence $< 5, 21, 3, 4, 8, 0, 1, 12, 15, 8, 10, 17, 3 >$ is sorted with Shellsort.

Answer.

input: $< 5, 21, 3, 4, 8, 0, 1, 12, 15, 8, 10, 17, 3 >$

4-sort: $< 3, 0, 1, 4, 5, 8, 3, 12, 8, 21, 10, 17, 15 >$

1-sort: $< 0, 1, 3, 3, 4, 5, 8, 8, 10, 12, 15, 17, 21 >$

final: $< 0, 1, 3, 3, 4, 5, 8, 8, 10, 12, 15, 17, 21 >$

Question 6 Give traces, in the style of the trace given with Algorithm 2.3, showing how the sequence $< 5, 21, 3, 4, 8, 0, 1, 12, 15, 8, 10, 17, 3 >$ is sorted with bottom-up approach of Mergesort.

Answer.

input: $< 5, 21, 3, 4, 8, 0, 1, 12, 15, 8, 10, 17, 3 >$

$len = 1$: $< 5, 21, 3, 4, 0, 8, 1, 12, 8, 15, 10, 17, 3 >$

$len = 2$: $< 3, 4, 5, 21, 0, 1, 8, 12, 8, 10, 15, 17, 3 >$

$len = 4$: $< 0, 1, 3, 4, 5, 8, 12, 21, 3, 8, 10, 15, 17 >$

$len = 8$: $< 0, 1, 3, 3, 4, 5, 8, 8, 10, 12, 15, 17, 21 >$
final: $< 0, 1, 3, 3, 4, 5, 8, 8, 10, 12, 15, 17, 21 >$

Question 7 How does Mergesort fare when there are duplicate values in the array?

Answer. If all the items have the same value, the running time is linear (with the extra test to skip the merge when the array is sorted), but if there is more than one duplicate value, this performance gain is not necessarily realized. For example, suppose that the input array consists of N items with one value in odd positions and N items with another value in even positions. The running time is linearithmic for such an array (it satisfies the same recurrence as for items with distinct values), not linear.