CS 6363.003 Spring 2021 Official Solutions

Homework 1 Problem 1

(a) Truthfully write the phrase ***"I have read and understand the policies on the course website."***

Solution: ***I have read and understand the policies on the course website.*** ■

Rubric: 1 point total.

Remark The proofs for parts (b) and (c) are far more verbose than I would normally expect. In particular, I often do not explicitly state the induction hypothesis in my own writing or lectures if it is clear from context. However, I wanted to make things especially clear for your first homework. When grading, Greg is free to be a bit strict if he believes *you* don't understand what the induction hypothesis is during your proofs.

(b) Prove that every non-negative integer can be written as the sum of distinct, non-consecutive Fibonacci numbers.

Solution: Let n be an arbitrary non-negative integer. Assume for all integers n' such that $0 \le n' < n$, that n' can be written as the sum of distinct, non-consecutive Fibonacci numbers.

Suppose $n = 0$. In this case, $n = F_0$.

Now suppose otherwise that $n \ge 1$. Let i be the largest integer such that $n \ge F_i$. Let $n' = n - F_i < n$. By assumption, n' can be written as the sum of distinct, non-consecutive Fibonacci numbers. Because $n < F_{i+1}$, we have $n' < F_{i+1} - F_i = F_{i-1}$. Therefore, either $n' = 0$ and its sum is empty or the largest term in its sum is at most F_{i-2} and therefore non-consecutive with F_i. Integer n can be written as the sum of F_i and the sum for n', proving the lemma for this case. ■

Rubric: 4 points total. -1 for missing the base case.

(c) Prove that every positive integer can be written as the sum of distinct Fibonacci numbers *with no consecutive gaps*.

Solution: Let n be an arbitrary positive integer. Per the hint, we will prove the following stronger claim: Let i be the largest integer such that $n \ge F_i$. We can write n as the sum of distinct Fibonacci numbers with no consecutive gaps, where the largest term in the sum is F_{i-1}. We now proceed with the proof.

Assume for all integers n' such that $1 \le n' < n$, the following claim: Let i' be the largest integer such that $n' \ge F_{i'}$. We can write n' as the sum of distinct Fibonacci numbers with no consecutive gaps, where the largest term in the sum is $F_{i'-1}$.

Suppose $n = 1$. We have $n = F_2$ and $n < F_3$, so $i = 2$. As claimed, we can write $n = F_1$.

Now suppose otherwise that $n \geq 2$. Let $n' = n - F_{i-1} < n$. Let i' be the largest integer such that $n' \geq F_{i'}$. By assumption, we can write n' as the sum of distinct Fibonacci numbers with no consecutive gaps, where the largest term in the sum is $F_{i'-1}$. Because $n < F_{i+1}$, we have $n' < F_{i+1} - F_{i-1} = F_i$. We see the largest term in the sum for n' is at most F_{i-2} and therefore distinct from F_{i-1}. Further, $n \geq F_i$, so $n' \geq F_i - F_{i-1} = F_{i-2}$. The largest term in the sum for n' is *at least* F_{i-3}, so there are not two or more consecutive gaps between that term and F_{i-1}. Integer n can be written as the sum of F_{i-1} and the sum for n', proving the lemma for this case. ■

Rubric: 5 points total. -1 for missing the base case.

CS 6363.003 Spring 2021 Official Solutions

Homework 1 Problem 2

Using Θ-notation, provide asymptotically tight bounds in terms of n for the solution to each of the following recurrences.

(a) $A(n) = 8A(n/3) + n^2$

Solution: Node values at level i of the recursion tree sum to $(8/9)^i n^2$, so the sum of level sums is a decreasing geometric series. The largest term is the value at the root, so $\boldsymbol{A(n) = \Theta(n^2)}$. ■

(b) $B(n) = 27B(n/9) + n^{1.5}$

Solution: Node values at each level of the recursion tree sum to $n^{1.5}$. There are $\Theta(\log n)$ levels in the recursion tree, so summing the level sums results in $\boldsymbol{B(n) = \Theta(n^{1.5} \log n)}$. ■

(c) $C(n) = 7C(n/5) + n$

Solution: Node values at level i of the recursion tree sum to $(7/5)^i n$, so the sum of level sums is an increasing geometric series. The largest term is proportional to the number of leaves, so $\boldsymbol{C(n) = \Theta(n^{\log_5 7})}$. ■

(d) $D(n) = 2D(n/4) + D(n/2) + n$

Solution: Node values at each full level of the recursion tree sum to n. Each subproblem is at most half the size of its parent, so there are at most $\log_2 n = O(\log n)$ levels of any size in the recursion tree. We see $D(n) = O(n \log n)$. On the other hand, each subproblem is at least a forth the size of its parent, so there are at least $\log_4 n = \Omega(\log n)$ full levels in the recursion tree. We see $D(n) = \Omega(n \log n)$ as well, implying $\boldsymbol{D(n) = \Theta(n \log n)}$. ■

(e) $E(n) = 2E(n/2) + n \lg^2 n$

Solution: Node values at level i of the recursion tree sum to $n \lg^2(n/2^i) \leq n \lg^2 n = O(n \log^2 n)$. There are $O(\log n)$ levels in the recursion tree, so $E(n) = O(n \log^3 n)$. On the other hand, each of the first, say, $(1/2)\lg n = \Omega(\log n)$ levels sum to at least

$$\begin{aligned} n \lg^2(n/2^{(1/2)\lg n}) &= n \lg^2(n/\sqrt{n}) \\ &= n \lg^2(\sqrt{n}) \\ &= n \lg((1/2)\lg n) \\ &= \Omega(n \log^2 n). \end{aligned}$$

We see $E(n) = \Omega(n \log^3 n)$ as well, implying $\boldsymbol{E(n) = \Theta(n \log^3 n)}$. ■

Rubric: 2 points per part. −1/2 points per part missing some kind of justification.

CS 6363.003 Spring 2021 Official Solutions

Homework 1 Problem 3

(a) Prove by induction that CRUEL correctly sorts any input array.

Solution: The proof relies on two useful observations. First, CRUEL has almost the exact same format as that of a mergesort, suggesting UNUSUAL expects an array with its first and last halves already sorted and then merges the two halves into one sorted array. Second, the for loop in UNUNSUAL moves the quarters of the input array around so its smallest and largest elements are primed for the recursive calls to move them to their final sorted positions. Similar to mergesort, we'll now argue CRUEL's correctness by doing two proofs by induction.

UNUSUAL First, we claim that UNUSUAL($A[1 \,..\, n]$) sorts $A[1 \,..\, n]$ assuming n is a power of 2, $n \geq 2$, and both $A[1 \,..\, n/2]$ and $A[n/2+1 \,..\, n]$ are sorted. If $n = 2$, then UNUSUAL swaps the two elements of A if and only if they are out of order, leaving A sorted.

Now suppose $n > 2$. Because the first and last halves of A are both sorted, we may conclude that each of its quarters, $A[1 \,..\, n/4]$, $A[n/4+1 \,..\, n/2]$, $A[n/2+1 \,..\, 3n/4]$, and $A[3n/4+1 \,..\, n]$ are sorted as well. The for loop in UNUSUAL simply swaps the 2nd and 3rd quarters of A without shuffling their members, so the quarters remain sorted after the for loop. Observe, however, that elements ranked 1 through $n/4$ appeared in the first or third quarters of A before the for loop; indeed, there are fewer than $n/4$ lessor elements than any of those ranked 1 through $n/4$. Therefore, *after* the for loop, all of the elements ranked 1 through $n/4$ appear in $A[1 \,..\, n/2]$, and they are the smallest $n/4$ elements of that subarray. Similarly, the elements ranked $3n/4+1$ through n all appear in $A[n/2+1 \,..\, n]$ after the for loop, and they are the largest $n/4$ elements of that subarray.

Now we get to the recursive calls. We established $A[1 \,..\, n/4]$ and $A[n/4+1 \,..\, n/2]$ are sorted, so we may inductively assume the first recursive call sorts the subarray $A[1 \,..\, n/2]$. In particular, the elements of A ranked 1 through $n/4$ have been placed in sorted order into $A[1 \,..\, n/4]$, and whatever elements were placed in $A[n/4+1 \,..\, n/2]$ are sorted as well. Similarly, the second recursive call moves elements of rank $3n/4+1$ through n into $A[3n/4+1 \,..\, n]$ in sorted order and $A[n/2+1 \,..\, 3n/4]$ is sorted. We may now inductively assume the third recursive calls sorts the remaining elements which are in $A[n/4+1 \,..\, 3n/4]$.

CRUEL We now prove CRUEL($A[1 \,..\, n]$) sorts its array. If $n \leq 1$, the array is already sorted, and CRUEL correctly does nothing. Otherwise, we may inductively assume the two recursive calls sort the subarrays $A[1 \,..\, n/2]$ and $A[n/2+1 \,..\, n]$. The input is now in the form expected by UNUSUAL which finishes sorting $A[1 \,..\, n]$ by our previous claim. ■

Rubric: 5 points total.

(b) What is the running time of UNUSUAL? Justify your answer.

Solution: Let $U(n)$ denote the worst case running time of Unusual($A[1 \,..\, n]$). Assuming n is sufficiently large, we spend $\Theta(n)$ time in the for loop and then do three recursive calls on arrays of half the size. Therefore, the running time follows the recurrence $U(n) = 3U(n/2) + \Theta(n)$.

Each level i of the recursion tree sums to at most $(3/2)^i cn$ for some constant c. The sum of level sums form an increasing geometric series, whose total is proportional to its largest term, the number of leaves in the tree. Therefore, $U(n) = O(n^{\log_2 3})$. The same argument implies the same asymptotic lower bound, so the worst case running time really is $\boldsymbol{\Theta(n^{\log_2 3})}$. ■

Rubric: 3 points total. −1 points for not including some kind of justification.

(c) What is the running time of Cruel? Justify your answer.

Solution: Let $C(n)$ denote the worst case running time of Cruel($A[1 \,..\, n]$). We do two recursive calls on arrays of half the size and then call Unusual($A[1 \,..\, n]$) which, as argued in part (b), runs in $\Theta(n^{\log_2 3})$ time. Therefore, the running time follows the recurrence $C(n) = 2C(n/2) + \Theta(n^{\log_2 3})$.

Each level of the recursion tree sums to at most $2^{(1-\log_2 3)i} cn$ for some constant c. As $1 - \log_2 3 < 0$, the sum of level sums form a decreasing geometric series, whose total is proportional to its largest term, the value at the tree's root. Therefore, $C(n) = O(n^{\log_2 3})$. The same argument implies the same asymptotic lower bound, so the worst case running time really is $\boldsymbol{\Theta(n^{\log_2 3})}$. ■

Rubric: 2 points total. −1/2 points for not including some kind of justification.

CS 6363.003 Spring 2021 Official Solutions
Homework 1 Problem 4

Describe and analyze an algorithm to find a central vertex in an arbitrary given binary tree.

Solution: Our algorithm is based on the following intuition: If deleting a vertex leaves a subtree of size greater than $n/2$, then we should search for a central vertex within that subtree; after all, the deletion of the central vertex needs to break up the subtree. Surprisingly, it turns out we can end our search at the highest point that no such subtree exists.

The procedure SizeOrCentral(v) described below is based on the above intuition. It takes a node v of T as its one parameter. It either reports a vertex in the subtree rooted at v as central, immediately terminating the overall algorithm, or it returns the size of the subtree rooted at v to help any recursive callers in their own searches and size computations. Note the procedure has access to n, the total size of T, as a global variable.

```
SizeOrCentral($v$):
  if $v$ has a left child $u$
      $\ell \leftarrow$ SizeOrCentral($u$)
  else
      $\ell \leftarrow 0$
  if $v$ has a right child $w$
      $r \leftarrow$ SizeOrCentral($w$)
  else
      $r \leftarrow 0$

  if $\ell \le n/2$ and $r \le n/2$ and $\ell + r + 1 \ge n/2$
      report $v$ as central and terminate algorithm
  else
      return $\ell + r + 1$
```

We first argue that each subproblem SizeOrCentral(v) either returns the size of the subtree rooted at v or some vertex in the subtree rooted at v is correctly reported as central. For each child of v, we may assume inductively that the recursive call on that child either correctly reports a central vertex in the child's subtree or returns the size of the child's subtree. The value $\ell + r + 1$ is the total size of the subtree rooted at v. If $\ell + r + 1 \ge n/2$, then either v has no parent, or there would be at most $n/2$ vertices in the subtree that would contain v's parent if we were to delete v. Therefore, the final if condition is a correct test on whether or not v is central, and we correctly report v as central if no other central vertex has been found first. Finally, if v is not central and we reach the final if statement, then we return the size of the subtree rooted at v as promised.

We have yet to argue SizeOrCentral(r) will always report a central vertex, however. We will describe a walk descending through T where every vertex is rooted at subtrees with strictly more than $n/2$ vertices. The walk begins at r whose subtree contains $n > n/2$ vertices. Suppose the walk reaches a vertex v. At most one child of v is the root of a subtree with strictly more than $n/2$ vertices. If such a child exists, the walk continues at that child. Otherwise, the walk ends, and we claim v is central. Indeed, the subtree rooted at v contains more than $n/2$ vertices, implying the parent of v (if any) would lie in a subtree of fewer than $n/2$ vertices if v were deleted. And by assumption, every child of v is the root of a subtree containing at most $n/2$ vertices.

The algorithm spends constant time per recursive call. It's a basic tree traversal, so there are $O(n)$ recursive calls total. The algorithm takes $\boldsymbol{O(n)}$ time total. ■

Rubric: 10 points total: 5 points for a correct algorithm. 3 point for the proof of correctness. -2 for not arguing that a central vertex exists. 2 point for the running time analysis.

(Here and more generally, our proofs of correctness will be more detailed than what is needed for full credit. You should still take them seriously, though, to convince both yourself and us that your algorithm is correct.)