Krzysztof Brzęczyszczykiewicz SID 834322002
Venkataratnam Narasimha Rattaiah SID 854377890

CS 111 ASSIGNMENT 127
due Friday, February 13

Problem 1: Consider a sequence defined recursively as $a_0 = 4$, $a_1 = 9$, and $a_n = a_{n-1} + 3a_{n-2}$ for $n \geq 2$. Prove that $a_n = O(2.5^n)$.

Solution 1: We first prove by induction that $a_n \leq 4(2.5)^n$ is true.
Base step: $a_0 = 4$, $4(2.5)^0 = 4$, so we have $a_0 \leq 4(2.5)^0$.
Inductive step: The inductive hypothesis is that $a_n \leq 4(2.5)^n$ is true for $n = 0, 1, ..., k$. To complete the inductive step, we need to show that $a_{k+1} \leq 4(2.5)^{k+1}$.

Since $a_k \leq 4(2.5)^k$ and $a_{k-1} \leq 4(2.5)^{k-1}$ holds, we have

$$\begin{aligned} a_{k+1} &= a_k + 3a_{k-1} \\ &\leq 4(2.5)^k + 3 \cdot 4(2.5)^{k-1} \\ &\leq 4(2.5)^{k-1}(2.5 + 3) \\ &\leq 4(2.5)^{k-1}(2.5)^2 \\ &\leq 4(2.5)^{k+1}. \end{aligned}$$

That is, we have shown that if the hypothesis is true, then $a_{k+1} \leq 4(2.5)^{k+1}$ is also true. This complete the inductive step.

We thus have $a_n \leq C|(2.5)^n|$ for $n \geq 0$ and $C = 4$. Therefore $a_n = O(2.5^n)$.

Problem 2: Suppose we have three sets, A_1, A_2, A_3 with the following properties:

(a) $|A_2| = 2|A_1|$, $|A_3| = 4|A_1|$,

(b) $|A_1 \cap A_2| = 2$, $|A_1 \cap A_3| = 2$, $|A_2 \cap A_3| = 5$,

(c) $|A_1 \cap A_2 \cap A_3| = 1$

(d) $|A_1 \cup A_2 \cup A_3| = 27$.

Use the inclusion-exclusion formula to determine the number of elements in $A_1 \cup A_1 \cup A_3$. Show your work.

Solution 2: We have

$$\begin{aligned} |A_1 \cup A_2 \cup A_3| &= |A_1| + |A_2| + |A_3| - |A_1 \cap A_2| - |A_1 \cap A_3| - |A_2 \cap A_3| + |A_1 \cap A_2 \cap A_3| \\ 27 &= |A_1| + |A_2| + |A_3| - 2 - 2 - 5 + 1 \\ |A_1| + |A_2| + |A_3| &= 35 \\ |A_2| &= 2|A_1| \\ |A_3| &= 4|A_1| \end{aligned}$$

By solving the above equations, we get the cardinalities of A_1, A_2 and A_3:

$$\begin{aligned} |A_1| &= 5 \\ |A_2| &= 10 \\ |A_3| &= 20 \end{aligned}$$

Problem 3: We are given an array $A[0, ..., n-1]$ that contains distinct numbers sorted in increasing order, that is $A[i] < A[i+1]$ for all $i = 0, ..., n-2$. Consider the algorithm below, described in pseudo-code

```
1 i <- 0;
2 j <- 0;
3 count <- 0;
4 while i < n do
5      while 2*A[j] < A[i]
6           do j <- j+1;
7      if 2*A[j] = A[i]
8           then count <- count+1;
9      i <- i+1;
10 print(count);
```

Explain what value is computed by this algorithm and give an asymptotic running time for this algorithm. Justify your answer.

Solution 3: (b) If all the numbers are non-negative, then this program computes the number of those i for which $A[i]/2$ is also in the array. (Equivalently, it's the number of j's for which $2A[j]$ is in the array.) The correctness follows from the fact that the entries in $A[]$ are strictly increasing. A more detailed explanation follows.

The formal proof can be carried out using mathematical induction. We claim that for each i, after the internal while loop stops, j is the smallest index for which $A[i] \leq 2A[j]$.

Indeed, for $i = 0$ we will end up with $j = 0$, which satisfies the claim. For the inductive step, suppose that this is true for some i. Then the smallest j' for which $A[i+1] \leq 2A[j']$ is at least as large as j and cannot be greater than $i+1$. This means that the internal while loop will correctly find this j' because it tries all these values in increasing order and stops when it finds the first one that satisfies this condition. Thus the claim above holds.

The program starts with `count` $= 0$. Fo each i, if $A[i]/2$ is in the array, say at location j, then j is the first index for which $2A[j] \geq A[i]$, in which case the program will increase `count`. This shows that the program computes the number of indices i for which $A[i]/2$ is in the array.

We emphasize that the above argument assumes that all numbers in $A[]$ are non-negative. In fact, if some numbers are negative, the internal loop may increase j beyond the range of $A[]$ (although this is quite easy to correct.)

(c) The running time analysis is very similar to that for the algorithm that computed common elements in two sorted arrays. Here again we assume that the numbers are non-negative.

The total running time for lines 1,2,3 and 10 is $O(1)$, so it's negligible. Lines 7,8,9 run in time $O(1)$ and are executed at most n times, because i is increased at each step. So the contribution of these lines to the overall running time is $O(n)$.

The only non-trivial part is the analysis of the running time of the nested `while` loops. The key observation is that in line 6 we always increase j and the value of j never exceeds i, so it never exceeds n. Thus the total number of increases of j, over the whole computation, is at most n. This shows that the total contribution of the nested `while` loop to the running time is $O(n)$. Putting it all together, we obtain that the running time is $O(n)$.
