Solutions Manual to Accompany

The Algorithm Design Manual

By

Melvyn Ian Drag

Contents

List of Figures

Chapter 1: Introduction To Algorithm Design

Problem 1

Question: *Show that $a + b$ can be less than* $\min(a, b)$

Solution: Let $a = b = -1$

Problem 2

Question: *Show that a x b can be less than $min(a, b)$*

Solution: Let $a = b = \frac{1}{2}$

Problem 3

Question: *Design/draw a road network with two points a and b such that the fastest route between a and b is not the shortest route*

Solution: Draw a right triangle with speed limits marked on the edges. The legs have speed limit 80 mph, but the hypotenuse has speed limit 10mph.

Problem 4

Question: *Design/draw a road network with two points a and b such that the shortest route between a and b is not the route with the fewest turns.*

Solution: See solution to problem 3.

Problem 5

Question:

Solution: All of these problems can be solved with the set $S = \{1, 2, 3\}$. $T = 4$.

Problem 6

Question:

Solution: $S_0 = \{1, 3, 5\}$, $S_1 = \{1, 2\}$, $S_2 = \{3, 4\}$, $S_3 = \{5, 6\}$. The algorithm grabs all four sets, but it only needs S_1, S_2, S_3

Problem 7

Question: *Prove the correctness of the following recursive algorithm to multiply two natural* *numbers, for all integer constants ¿= 2*

Solution: Base Case: if z == 0, then the product is zero. Assumptions: The recursion produces a valid result for z ¡= n. c ¿= 2, as specified in the problem. y ¿= 0, as specified in the problem. Proof: multiply(y, n + 1)= multiply(cy, floor((n+1)/c)) + y*((n+1) mod c) = cy*floor((n+1)/c) + y((n+1) mod c), since this multiply function is for input z =floor((n + 1)/ c) ¡= n = y(c floor(z/c) + z mod c) z = qc + r floor(z/c) = q z - cfloor(z/c) = r = zmodc Therefore z = cfloor(z/c) + zmodc And the result follows.

Problem 8 Question: Solution: The base case in which n == 1 and the polynomial is a_0 produces the correct output, so we have a basis for induction. Then let us take the original $A = (a_n, a_{n-1}, \ldots, a_0$ and extend it to be $\bar{A} = (a_{n+1}, a_n, a_{n-1}, \ldots, a_0$. We have to show that the horner() function will produce the correct output in this instance. We can peel the first iteration off of the loop to give:

```
horner(A, x)
        p = A_{n+1}x + A_n
        for i = n-1 to 0:
                p+= p*x + A_i
        return p1 + p2
```

And by the linearity of the loop operation we can split the loop into:

```
horner(A, x){
        p = A_{n+1}x + A_n
        tmp = A_{n+1}x
        for i = n-1 to 0:
                tmp *= x
        p = horner(A[n:0], x) //correct by induction hypothesis.
        return p + tmp
}
```

Problem 9: I wish the algorithm would have been given with a step size provided for the for loops. The outer loop only needs to go from n to 2, because the step on the inner loop is plus one, so when i = 1, the inner for loop wil not execute. In any event, we start by considering the trivial base case when n = 1, and there is a single element in the list. bubbleSort will correctly sort this list. Then, we assume that the function works for lists of size up to n, i.e. on the list A = $(a_1, a_2, a_3, ..., a_n)$, and somewhere in the list we throw another element on the list, a_{n+1}. We need to show that in one iteration of the i loop the largest a_k will bubble to the top, and that we have an unsorted list of n elements left, which we know will be correctly handled. We can peel off the first iteration of the i loop giving:

```
function bubbleSort(A: list[1, ..., n+1])
        var int i,
        for j from 1 to n
                if A[j] > A[j+1]
                        swap(A[j], A[j+1]) // We will look at the jth element in the
        // The later iterations of j will never reach n again, so element n+1 will n
        bubbleSort(A: list[1, n])
```

Problem 10 - 12 These are standard exercises in many math courses. The proofs can surely be found in many place online.

Problem 1-28 I would always begin an interview question with a question of my own. I would say - by integer division you mean the quotient of the divion of two numbers, omitting the remainder? They would say yes. Then I would ask if they wanted some special treatment of the negatives, because to my knowledge the result of integer division is implementatoin specific when it comes to negatives. I guess we want the number n such that a / b = bn + r. I would then work from there. For positive numbers we just do:

```
ans = 0
while a > b
        ans += 1
        b = b + b
return ans
```

The challenge of this problem is the fact that either a or b might be negative.

1-29 Sample Dialogue:

Me: What do you mean by fastest? Like fastest at one instant in time? Or generally the fastest? Fastest while tired? Please expand. Them: You Decide. Me: Ok. Do we have a stop watch? Them: No. Me: Okay, then to get the fastest in one race, we have to run six races on one fixed length course. The first five races are run between disjoint sets of five horses each. Then the finalists run one more race to determine the overall winner.

This would be the easiest way to find the winner, but it is a highly controverial winner. We might want to run 6n races, where n is the number of different tracks we run on. Certainly some horses are marathoners and others are sprinters, and we would get different results on at least some of the n tracks. We might need more races for differences in temperature, different periods of rest, turf - I could continue but it suffices to say that this is a very subjective question. Them: So what is your final answer? Me: Do horses attain different speeds on different turf ?Them: Yes. Me: Does temperature affect a horses speed? Them: Yes. Me: Okay, so I will make the simplifying assumption that turf, temperature, and the course length are the only factors that affect the speed and endurance of a horse. Then, with no stopwatch, we need $6(n_turf * n_temp * n_length)$ different races. Then we take the horse with the most wins out of all. There might be a tie. I am also assuming that there will be a clear winner in every race - the horses will never be so close that we can't decide a winner.

1-30 - 1-33 These are similar to the estimation problems from before.

1-34 Me: How many pages are there in the phone book? Them: ¡A number¿ Me: And how many names are there are pages 1 - 10? Them: ¡Another number¿ Me: Okay so ¡N¿ = ¡Another number¿ / 10 names per page in the phone book. - By the way, does the phone book have white pages? Or just yellow? Them: You decide. Me: Oh, and what do you mean by 'find a specific name'? Do you mean open to the page that the name is on, or open the book and put your finger on the indicated name? Them: Let's say, open the book and put your finger on the correct name. Me: Do you have any name in particular in mind? Them: It could be any name. Me: Is the book new, or is it worn? Also, how is it bound? Is there a seam or something in the binding of the book? Them: The book is new, with no seams. Me: Do you know the name you are looking for? Them: I don't think it matters, because you are opening the book randomly. Me: Just what i was thinking. Ok, and by 'flip' the book open, you do mean opening it normally? Or do you mean something else? Them: Normally. Me: I would first do some tests, and see if people have a prediliction towards opening to pages in the middle of the book. I'm sure they do. Then I would work out some weights to correspond to different bins in the book. Then I would devise a set of weights for different sets of pages in the book. So, for example, pages 1- 10 have weight 2, but pages

100 - 250 have weight 100. Then I would compute the sum of the $\sum_i w_i * num_pages_bin_i$, and then renormalize the probabilities for pages. For example:

4 pages, page 1 has weight 1, pages 2, 3 have weight 10, page 4 has weight 1. 1 + 20 + 1 = 22. 1 / 22 + 20 / 22 = 1 / 22*1/20 = 1 / 1.1 + 1 / 22 So you would open the book once or twice and hit page 2 or three, but probably only once in 22 opens would you hit 1 or 4.