Algorithms Homework 4
H. DiMarchi

1 Chapter 3

1.1 Section 2

1.1.1 Problem 8

a. Start by initializing a count of 0. Iterate through the string. Check each character. If it is an A, iterate through the rest of the string, checking each character: If the character is a B, increment your count of substrings that start with A and end with B. Continue until you have finished the top level iteration through the string.

b. Start by initializing a count of 0, and an increment_amount of 1. Iterate through the string, checking each character. If the character is a B, increment your count of substrings that begin with A and end with begin by your increment amount. If the character is an A, increment your increment count by 1: By doing this you will account for each substring that could be constructed from a particular B after that A. Continue this until you have finished iterating through the string.

This is a more efficient algorithm because it requires you to iterate through the string only once.

2 Chapter 4

2.1 Section 5

2.1.1 Problem 13

Iterate through matrix by length of rows such that each item you touch on begins a row. For each item, check if that item is greater than or equal to the value you are searching for. If that item is equal to the value, return that index. If that item is greater than the value look through preceding row from the second element of that row to the end of that row for the value you are searching for until you find it, then return the index of the value searched for. If that item is neither go to the first element of the next row and make the same check. If you arrive at the last row, and its first value is less than the value searched for , iterate through that row for the value.

Value searched for: 7

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Items Visited: 1 -> 5 -> 9 -> 6 -> 7 -> done

3 Chapter 5

3.1 Section 1

3.1.1 Problem 5

a. $T(n) = 4T(n/2) + n, T(1) = 1$

Using the Master Theorem:

$a = 4, b = 2, c = 1, k = 1$

Note, $4 > 2^1$ thus $a > b^k$ and $O(T(n)) = O(n^{\log_2 4}) = O(n^2)$

b. $T(n) = 4T(n/2) + n^2, T(1) = 1$

Using the Master Theorem:

$a = 4, b = 2, c = 1, k = 2$

Note, $4 = 2^2$ thus $a = b^k$ and $O(T(n)) = O(n^2 \log n)$

c. $T(n) = 4T(n/2) + n^3, T(1) = 1$
Using the Master Theorem:
$a = 4, b = 2, c = 1, k = 3$
Note, $4 < 2^3$ thus $a < b^k$ and $O(T(n)) = O(n^3)$

3.2 Section 2

3.2.1 Problem 11

Start by selecting a random bolt, and then iterate through the nuts, checking if each nut fits. If it is too small, place it in one pile. If it is too large, put it in another. Do this until all nuts but one are in one pile or the other and one nut is with the bolt. Select your next bolt and check to see if the nut on the bolt just picked is too large or too small for it. If it is too small, check each nut in the pile of nuts that were too large for the bolt. If it is too large, then the other pile. When checking these nuts, again split the nuts into two seperate piles, creating a total of 3 piles. To extend this, every time a new bolt is selected check it first against the nuts that have found their bolt, checking the pile of nuts that is between the last bolt whose nut the new bolt is too large for and the first bolt whose nut the new bolt is too small for.

3.3 Section 3

3.3.1 Problem 5

a. Preorder
abdecf

b. Inorder
dbeacf

c. Postorder
debfca

4 Chapter 6

4.1 Section 1

4.1.1 Problem 8

Split pairs into a single list of numbers, marking which is a 1st element of a pair, and which is a 2nd. Sort the list, then traverse it. Keep track of a current count and a max count, both starting at 0. Add 1 to the current count each time a starting element is arrived at. Each time an ending element is arrived at, first check to see if the current count is greater than the current max count. If it is, replace the max count with he current count. Then subtract 1 from the current count.

4.2 Section 6

4.2.1 Problem 6

Algorithm:
first, second = 0
max_product = 0
if n is even:

first = n/2
second = first
max_product = first * second
if n is odd:
first = floor(n/2)
second = ceiling(n/2)
max_product = first * second

Efficiency class = O(1)
Works because of all pairs that sum to n, the pair that are equal or closest to it will always multiply to produce the greatest product.
Consider:
$first + second = n, second = n - first$
$first * second = product = first * n - first = first(n) - first^2$
This function produces a parabola whose maximum is approximately $n/2$

5 Chapter 7

5.1 Section 1

5.1.1 Problem 6

If the vertex u is an ancestor of the vertex v then it is visited first in a preorder traversel and second in a postorder traversel, which can be stated as $preorder(u) <= preorder(v)$ and $postorder(u) >= postorder(v)$. Thus we can traverse this tree in both ways, which is an O(n) operation, and record when u and v are visited. We can then compare these values in constant time.