

Homework 1

Due February 3

This is the last homework on which the following information will appear, but remember that it applies to every homework:

- Homeworks are due at 11:59 PM on the date specified. Homeworks are to be placed in the CS 16 handin bin, located by the glass doors on the second floor.
- Please staple your homeworks and put your name and login on each page.
- Please ensure that when prompted to provide pseudocode you follow proper pseudocode formatting guidelines. See the relevant links in the Information section of the website. Commenting of your pseudocode is *strongly* encouraged.
- If you use L^AT_EX, use the `newalg` or `algorithmic` packages (<http://www.cs.brown.edu/system/software/latex/packages.html>) to format your pseudocode.
- When writing pseudocode, you may use any algorithms in the lecture slides or course texts for which there is pseudocode provided. If you do so, please cite the specific lecture slide or page from the book where the algorithm is described.
- Credit for problems comes in part from the simplicity of your answers; insanely long answers lose credit.
- Use pictures to illustrate your ideas.
- Write neatly. Hard-to-read homework gets no credit. If you scratch things out, rewrite and hand in a clean copy.

This is a collaborative homework.

Problem 1.1

When a share of common stock of some company is sold, the capital gain (or loss) is the difference between the share's selling price and the price originally paid to buy it. This rule is easy to understand for a single share, but if we sell multiple shares of stock bought over a long period of time, then we must identify the shares actually being sold. A possible accounting principle in such a case is to use a LIFO (last in first out) protocol—the shares sold are the ones that have been held the shortest.

For example, suppose we buy 200 shares at \$20 each on day 1, 20 shares at \$24 on day 2, 50 shares at \$36 on day 3, and then sell 150 shares on day 4 at

\$30 each. To sell 150 shares, LIFO says we sell the most recent 50 shares (i.e. from day 3), the 20 shares from day 2, and 80 of the 200 shares from day 1. The capital gain in this case would be $(50 \times (-6)) + (20 \times (6)) + (80 \times (10))$, or \$620.

Give the pseudocode for a program that takes as input a sequence of transactions of the form “buy x shares at $\$y$ each” or “sell x shares at $\$y$ each,” assuming that the transactions occur on consecutive days and the values x and y are integers. Given this input sequence, the output should be the total capital gain (or loss) for the entire sequence, using the LIFO protocol to identify shares. Your program should use memory proportional to the number of transactions. (The stock of only one company is being bought and sold.)

Problem 1.2

In most programming languages, parenthetical symbols $()$, $[]$, and $\{\}$ must be balanced and properly nested. We define a *parenthetically correct* string of characters as a string that matches one of the following patterns, where S denotes a (possibly empty) string without any parenthetical symbols, and P, P' , and P'' recursively denote parenthetically correct strings:

- S
- $P' (P) P''$
- $P' [P] P''$
- $P' \{P\} P''$

Give the pseudo-code for an algorithm that checks whether a string σ of length n is parenthetically correct in time proportional to n , using a stack of characters as an auxiliary data structure. You may suppose you are given a function $\text{Match}(a, b)$ which returns TRUE if a and b are matching parentheses, and returns FALSE otherwise. σ is given as an array of characters.

Problem 1.3

- (a) Describe in pseudo-code an algorithm for reversing a queue Q containing n elements.
- (b) A palindrome is a word or phrase that reads the same forward and backward; that is, the letters are the same whether you read them from right-to-left or from left-to-right (ignoring spaces, punctuation and capitalization).

For instance, “radar” and “A man, a plan, a canal, Panama!” are palindromes, while “palindrome” and “Computer Science” are not.

Describe in pseudo-code an algorithm that determines whether a string σ of length n is a palindrome, using a stack and a queue. You may assume you have a function $\text{isLetter}(a)$ which returns TRUE if a is a letter, and FALSE otherwise; also, you may use a function $\text{LowerCase}(a)$ which returns a lower case version of a .

Problem 1.4

(a) Consider the k -term series $1 - 2 + 3 - \dots \pm k$; call the sum of this $S(k)$. Evaluate $S(1)$, $S(2)$, $S(3)$ and then inductively make a conjecture about $S(k)$ for any k . You don't need to prove your conjecture.

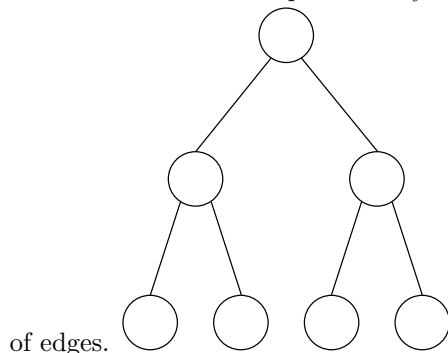
(b) A function F is defined recursively by two rules:

1: $F(0) = 1$.

2: For any integer $k > 0$, $F(k) = 1 + 2F(k - 1)$.

Compute $F(1)$, $F(2)$, $F(3)$, and $F(4)$, and inductively conjecture a pattern that the values of F follow.

(c) A “complete binary tree of height k ” is a structure like the ones shown below; for height 1, there's just a single dot; for height 2, there are three dots and two edges. In general, a complete binary tree of height k consists of k rows of dots; every dot except the top one is joined by an edge to exactly one dot above it (its “parent”); every dot except those in the bottom row is joined to *two* dots below it (its “children”). Make a conjecture about the number of dots in a complete binary tree of height k , and about the number



(d) Consider the k -term series $A(k) = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} \dots + \frac{1}{k \cdot (k+1)}$; make an inductive conjecture about the value of $A(k)$.

Problem 1.5

Dilbert is responsible for organizing a marathon. Because the best runners get to start first, he needs a way to order people from front to back. Marathoners wear number-tags so that it's easy to tell who finished first. The simple thing to do would be to assign number 1 to the top seeded runner, number 2 to the next, and so on. But because he's Dilbert, he decides to give everyone a BINARY number instead. (See http://en.wikipedia.org/wiki/Binary_numeral_system). There are 1000 runners expected in the race, so he uses 10-bit binary numbers, and writes them out completely, so that the top seeded runner's number is 00 0000 0001 (spaces included for readability).

When it comes time to race, he wants everyone to line up in order...but most marathoners don't know binary. So he has them queue up randomly on a long street.

With his megaphone, he says "Everyone whose rightmost digit is a 1, take one step to the right; everyone else take one step to the left." Now he has two queues. The people in the left queue are told "step forward until you're close to the person in front of you." Those in the right queue are told "step BACKWARD until you're close to the person behind you." (The foremost and backmost persons are told not to move). When everyone's settled, the tail of the left queue is just a little in front of where the head of the right queue is (assuming each runner occupies the same amount of space).

(a) Explain why the head and tail of the queues match like this.

Now he tells everyone to take a step to the right or left so that they're once again in a single queue. They repeat the process for the second-to-rightmost digit. And then the third-to-rightmost digit, and so on until the very last digit.

You can hand-simulate the process with a deck of cards. Extract the club suit and shuffle it. Associate to each card its binary representation (A = 0001, 2 = 0010, ...9 = 1001, 10=1010, J=1011, Q=1100, K=1101). [You may also want to just get 13 index cards and write these 13 binary numbers on them!] Shuffle the deck. Holding it in your hand face up, look at the first card. If its last binary digit is "1" (i.e., if it's an odd number), place it face-down in a pile on your right; if it's even, place it face-down in a pile on your left. Repeat until you have no more cards in your hand. Then place the face-down "1" pile atop the face-down "0" pile. Pick up the resulting pile and turn it over so that you can see the cards. Now repeat the process for the 2's place, the 4's place, and the 8's place. When your done, the deck should be sorted from lowest to highest (as viewed from the face-up side).

(b) Carry out this exercise until it works for you. On your handin, write "I did part b."

(c) Explain, as best you can, why the runners (or the cards) are now in order from smallest to largest.

(d) Implement this idea in pseudocode: you have a queue full of 12-bit binary numbers. If n is one of these numbers, then $n.bit(i)$ returns either 0 or 1. The bits are ordered from right to left, so if

$n = 0000\ 0000\ 1101$ then

$n.bit(0) = 1$

$n.bit(1) = 0$

$n.bit(2) = 1$

$n.bit(3) = 1$

$n.bit(4) = 0$

$n.bit(k)$ for $k = 5, 6, \dots, 11$ are all zero too.

Write pseudocode for `marathonSort`, where the input is a queue Q of numbers and the output is a queue containing the same numbers in increasing order.

(e) Suppose that we perform this process for a marathon with n runners. What's the big-O running time of your algorithm in terms of n ?

Problem 1.6

In class, we studied seam-carving; the key algorithm took an $n \times k$ array of “importance” values and found an optimal “seam” from the top to the bottom. Recall that a seam is a path from top to bottom that contains one entry in each row, and each entry is either vertically or diagonally adjacent to any entries in the rows above or below it. A seam was “optimal” if it had the lowest possible sum of importance values.

A student proposed a greedy algorithm for finding a seam; in pseudocode, it looked like this:

Input:

- n , a number of rows
- k , a number of columns
- $T[,]$, an $n \times k$ 1-based array of importance values

Output:

- $c[]$, a 1-based array of n column-indices. The seam consists of the pixels at locations:

$$(1, c[1]), (2, c[2]), \dots, (n, c[n])$$

Algorithm:

Let $c[1]$ = the column-index of the smallest entry of row 1 of T .

for $i = 2$ to n **do**

Let $c[i]$ = the column-index of the smallest of $T[i, c[i - 1] - 1]$, $T[i, c[i]]$,
and $T[i, c[i - 1] + 1]$

end for

return c

Sometimes greedy algorithms are reasonably effective – they might find an answer that’s within, say, a factor of two of the optimal answer. In the seam-carving problem, that’s not the case.

Build a 3×3 array in which the greedy algorithm produces a seam that’s 1000 times as costly as the optimal seam. (You get to put any entries you like into your importance array). Briefly explain how you could make the factor of 1000 be any number you liked by slightly altering your example.