Josh Bowden

Professor Sasaki

CS 330 – Discrete Structures

20 March 2016

Homework 6

1. 4 points:
   1. Find a recurrence relation for the number of bit strings of length n that contain three consecutive 0s.
   2. What are the initial conditions?
   3. How many bit strings of length seven contain three consecutive 0s?
2. 4 points: A bus driver pays all tolls using only nickels and dimes, by throwing one coin at a time into the mechanical toll collector.
   1. Find a recurrence relation for the number of different ways the bus driver can pay a toll of n cents (where the order in which the coins are used matters).
   2. In how many different ways can the driver pay a toll of 45 cents?
3. 4 points: Suppose that there are teams in an elimination tournament, where there are games in the first round, with the winners playing in the second round, and so on. Develop a recurrence relation for the number of rounds in the tournament.
4. 4 points: Solve the recurrence relation in number 3.

*(continued on next page)*

1. 4 points: Suppose that each person in a group of *n* people votes for exactly two people from a slate of candidates to fill two positions on a committee. The top two finishers both win positions as long as each receives more than *n*∕2 votes.

Hint: Assume that n is even and split the sequence of votes into two sequences, each with n∕2 elements. A candidate cannot receive a majority of votes without receiving a majority of votes in at least one of the two halves.

* 1. Write out (in pseudocode or words) a divide-and-conquer algorithm that determines whether the two candidates who received the most votes each received at least *n*∕2 votes and, if so, determine who these two candidates are.

def tally(votes: List[Int], canidate: Int) {

n := 0

foreach vote in votes {

if vote == canidate {

n := n + 1

}

}

return n

}

def findMajority(a: list of votes) {

n := len(votes) // assuming n is even

if n == 1 { return a[1] }

split = floor(n/2)

majorityLeft := f(votes[1..split])

majotityRight := f(votes[split+1..n])

if left == right {

return majorityLeft

}

else {

votesLeft = tally(votes, majorityLeft)

votesRight = tally(votes, majorityRight)

if (votesLeft > split + 1) {

return majorityLeft

}

else if (votesRight > split + 1) {

return majorityRight

}

else {

return None // no majority

}

}

}

*(continued on next page)*

def findTwoMajorities(votes: list of votes) {

first = findMajority(votes)

if (first == None) { return None }

for i in range(1..len(votes)) {

if votes[i] == first:

delete votes[i]

}

second = findMajority(votes)

if (second == None) { return None }

return (first, second)

}

* 1. Use the master theorem to give an *O*(…) estimate for the number of **comparisons** needed by the algorithm you devised in part (a).

1. 8 points: Find the solution to each of these recurrence relations with the given initial conditions. Use an iterative approach.
2. 9 points: Solve these recurrence relations together with the initial conditions given.

*(continued on next page)*

1. 3 points: A model for the number of lobsters caught per year is based on the assumption that the number of lobsters caught in a year is the average of the number caught in the two previous years.
   1. Find a recurrence relation for , where is the number of lobsters caught in year n, under the assumption for this model.
   2. Find if 100,000 lobsters were caught in year 1 and 300,000 were caught in year 2.