**CS 180** Homework 1

**Problem 1**

1. To prove that there is a move that can transform any favorable table to an unfavorable one, we can consider the base case of n = 1 row of matches. Regardless of the number of matches, we will have a favorable table with columns of odd counts in the binary representation; the player can then just remove the whole row to make an unfavorable table (and win). With n + 1 rows, a favorable table means that the binary representations of each row will differ in at least one column. To transform the table into an unfavorable table just requires finding the leftmost significant column with an odd number of ones and flipping that one to a zero. By doing this, we make that column an even column and now we can make any combination of 1’s and 0’s to the right of that column. Since columns can only be odd or even, we just need to place the 1’s in the odd columns and thus, we will have an unfavorable table. With another row, we can use the same algorithm. It works because when we find the leftmost odd column, that means that all the columns to its left are even (unfavorable). Then we flip the 1 to a 0 and make this targeted column even and in doing so, can place 1’s anywhere to the right of the column and balance out the rest of the table.

Given an unfavorable table, any move can only make the table favorable for the opponent. By definition of an unfavorable table, all the columns in the binary representation are even. In the next move, a player would subtract a number of matches from a row and in doing so, they would change that row’s binary representation and therefore make at least one column odd now.

Given a favorable table, XOR all the rows together to achieve one binary representation; this will be the number of matches to remove. XOR this number to each individual row of the original favorable table. The row that results in a binary representation less than its original value is the row to remove matches from.

1. Find the leftmost column where there is an odd number of ones in the binary representation. Pick any row that contains a one in this column and flip that one to a zero. Then change the bits to the right of this row to a series of 1’s and 0’s that would make the rest of the columns have an even amount of numbers. The opponent, given an unfavorable table will only be able to transform it back into a favorable table so you can continue this strategy until you win.

**Problem 2**

1. This algorithm would take 3 arguments, the array of coefficients a0, a1, a2, … , an, the degree of the polynomial and the variable v. This algorithm would loop n degree times and in each iteration, do n2 multiplications and n additions.

compute (int coeffs[], int degree, int v)

count = degree

sum = 0

temp = 1

while count >= 0

loop from 0 until count

temp \*= v

temp \*= coeffs[count]

sum += temp

temp = 1

count--

return sum

1. This algorithm would only require O(n) multiplications and additions. It does so by starting at the highest degree coefficient and multiplying out the variable, staggering it so the highest degree coefficient multiplies v the most, then the second highest degree coefficient, etc.

compute (int coeffs[], int degree, int v)

result = coeffs[degree]

i = degree - 1

while i >= 0

result = result \* v + coeffs[i]

i--

return result

**Problem 3**

div(a, b)

mult = 1

remainder = 0

while (a > 10b)

remainder += mult \* (a % b)

mult \*= 10

a /= 10

div(a, b, rem, tens)

if a < 10b

result = a / b

rem = a - result \* b

return

else

div(a/10, b, rem, tens + 1)

**Problem 4**

Given an array arr[0,1,2, … n] where each index is a player and n is the number of players…

roundRobin (int arr[], int n) {

if (size == 2) {

match the two players

add match to table

return table

}

count1 = 0

while count1 < n / 2

count2 = 0

while count2 < n / 2

index = 0;

if (count1 + count2 >= n / 2)

index = n/2 - count1 + count2

else

index = count1 + count2

match arr[index] with arr[n/2 + index]

put match into main table;

count2++;

count1++;

table1 = roundRobin (arr[0,1,2, ... (n-1)/2]) // split array into half

table2 = roundRobin (arr[n/2, ... n])

for each row in table1 and table 2

combine the rows and add to main table

return main table;

}