CS180 HW1

Siddharth Joshi

TOTAL POINTS

100 / 100

QUESTION 1

1 Problem 1 25 / 25

✓ **- 0 pts Correct**

- 3 pts wrong length of n

- 5 pts wrong iterations

- 10 pts derived n*logn form

- 15 pts wrong equation

- 20 pts wrong answer, but show efforts

- 25 pts wrong answer

QUESTION 2

Problem 2 25 pts

2.1 10 / 10

✓ **- 0 pts Correct**

- 6 pts wrong answer or missing answer for making favorable table unfavorable

- 2 pts wrong answer or missing answer for making unfavorable table favorable

- 5 pts algorithm not correct

- 2 pts algorithm partial correct

- 7 pts wrong answer, but show efforts

- 10 pts wrong answer

2.2 10 / 10

✓ **- 0 pts Correct**

- 3 pts correct answer, partial correct explanation

- 5 pts correct answer, but wrong or missing explanation

- 8 pts wrong answer, but show efforts

- 10 pts wrong answer or no answer

2.3 5 / 5

✓ **- 0 pts Correct**

- 2 pts wrong answer, but show efforts

- 5 pts wrong answer

QUESTION 3

Problem 3 25 pts

3.1 5 / 5

✓ **- 0 pts Correct**

- 1 pts fail to show which edges to remove/connect

- 5 pts no answer found

3.2 10 / 10

✓ **- 0 pts Correct**

- 2 pts fail to show which vertices to connect

- 2 pts fail to show step by step explanation of the algorithm

- 4 pts fail to show why the algorithm works as expected

- 2 pts correct algorithm, but fail to show why the algorithm works as expected

- 10 pts no answer found

- 0 pts Click here to replace this description.

3.3 10 / 10

✓ **- 0 pts Correct**

- 3 pts fail to show the steps to reduce 2^k to 2^(k-1)

- 10 pts no answer found

QUESTION 4

Problem 4 25 pts

4.1 15 / 15

✓ **- 0 pts Correct**

- 3 pts incorrect function header

- 0 pts Click here to replace this description.

- 15 pts no answer found

4.2 10 / 10

✓ **- 0 pts Correct**

- 15 pts no answer found

HW 1

1. number of times the right-most bit flips is n times as in every iteration we add $1 = 2^0$

similarly the digit bit in the 2^{1}st place i.e. the 2nd right most digit flips $n/2$ times as it takes 2 iterations to add $2 = 2^1$

∴ so on and so forth the $k+1^{th}$ digit bit flips $n/2^k$ times as it takes 2^k iterations to add 2^k

- total number of bit operations $= n + \frac{n}{2} + \cdots + \frac{n}{2^x}$

where $x = \log_2 n$

∴ $= \lim_{n \to \infty}$ (total # of bit operations)

= (sum to infinity of the geometric progression with first term 1 and common ratio 1/2) $\times$ n

$= \left(\frac{1}{1-1/2}\right) * n = 2n = O(n)$

Ans $O(n)$

2. a) For every table there exists a row that has the most number of matches - Let this be row R

Assume there exists a favorable table

1 Problem 1 25 / 25

✓ **- 0 pts Correct**

- 3 pts wrong length of n

- 5 pts wrong iterations

- 10 pts derived n*logn form

- 15 pts wrong equation

- 20 pts wrong answer, but show efforts

- 25 pts wrong answer

2. a) → Let the columns of the table be labeled $c_0 \ldots c_j$ where c_j is the most significant column
→ Let the rows of the table be labeled $r_0 \ldots r_i$ where r_i is the bottom most row

→ since the table is favorable $\exists\ c_{i,}\ 0 \le i \le j$ and $i \in \mathbb{Z}$ s.t. c_j has an odd number of 1s in it

→ Let c_x be the most significant column of this form
→ since the column has an odd number of 1s, $\exists\ r_x$ s.t. has a 1 in this column i.e.
element @ $r_x c_x = 1$
(this is necessary as c_x has at least one 1 in it)

→ since c_x is the most significant column of the form of c_i, all other such columns can have the number of 1s in altered by r_x

→ Flip the bit in this case) $r_x c_x$ (i.e. set it 0, move through the row checking the number of 1s in each corresponding column, if the column has an even number of 1s move ahead else flip the bit (i.e. if it is 1, set to 0 else set to 1)

→ let $r_{x\,new}$ represent the new row we have constructed.
→ since $r_{x\,new}\ c_x = 0$ while $r_x\ c_x = 1$, the number $r_{x\,new}$ represents in binary is definitely smaller than the number represented by $r_x \Rightarrow$ that there is a legal move transforming $r_x \to r_{x\,new}$ and thus there exists a move that makes a favorable table unfavorable as $r_{x\,new}$ guarantees every column has an even number of 1s in it

→ Let r_y be the row the player opponent alters with his/her move

→ Any valid move must pick at least 1 match from r_y which implies that there exists an element $r_y c_y \in r_y$ that is flipped from 1 to 0.

→ This reduces the number of 1's in c_y, and hence the # of 1's in c_y since the table was ~~favorable~~ unfavorable to begin with, was $= 2n$ where $n \in Z^+ \cup \{0\}$, the column at the end of the move has $2n-1$ where $n \in Z^+ \Rightarrow 2n-1$ is odd ~~an~~ $\Rightarrow$ $c_{y\text{new}}$ (the column c_y after the move) has an odd number of 1's and hence the table is necessarily favorable at the end of any move on an unfavorable table

Algorithm

Assume the input is Table with n rows and m columns.
let l be a list of booleans denoting whether a column has an odd number with element i being true if c_i has an odd number of 1's and initialize all elements of l to false
for each column c_i:
{for each element x in the column
{if $x == 1$ then $l[i] \Rightarrow$ not $l[i]$ }}
for each element in l: (called $l[i]$) traversing from the end (n..0)
{ if $l[i]$ then $j = i$ and ~~break~~ break out of the loop }
for each row r_i:
{ if $r_i c_j == 1$ then $r_x = r_i$ } let $r_{x\text{old}} = r_x$
flip $r_x c_j$
for each $r_x c_i$ s.t. $i < j$: {if $l[i]$ then flip $r_x c_i$ }
let x = number represented by $r_{x\text{old}}$ and let y = number represented by r_x
thus return move is to take $x-y$ matches from [illegible]

Time complexity:
→ Traversing each column and each element in the column is $O(n^2)$
→ since all other parts of the algorithm involve traversing lists, which are $O(n)$ ∴ algorithm has complexity = $O(n^2)$

Correctness was proven at the beginning

b) Yes, multiple ways can exist to make a table unfavorable as if more than 1 row has a 1 in the column in (most significant column with odd number of 1s)
then any such row can be chosen for the move to make the table unfavorable

c) Take the current table, if there are only 2 rows and 1 row has just 1 match, then take the entire other row and force a win
else use the algorithm in part a) to play a move that makes the table unfavorable

2.1 10 / 10

✓ **- 0 pts** Correct

- 6 pts wrong answer or missing answer for making favorable table unfavorable

- 2 pts wrong answer or missing answer for making unfavorable table favorable

- 5 pts algorithm not correct

- 2 pts algorithm partial correct

- 7 pts wrong answer, but show efforts

- 10 pts wrong answer

Time complexity:
→ Traversing each column and each element in the column is $O(n^2)$
→ since all other parts of the algorithm involve traversing lists, which are $O(n)$ ∴ algorithm has complexity = $O(n^2)$

Correctness was proven at the beginning

b) Yes, multiple ways can exist to make a table unfavorable as if more than 1 row has a 1 in the column in (most significant column with odd number of 1s)
then any such row can be chosen for the move to make the table unfavorable

c) Take the current table, if there are only 2 rows and 1 row has just 1 match, then take the entire other row and force a win
else use the algorithm in part a) to play a move that makes the table unfavorable

2.2 10 / 10

✓ **- 0 pts** **Correct**

- 3 pts correct answer, partial correct explanation

- 5 pts correct answer, but wrong or missing explanation

- 8 pts wrong answer, but show efforts

- 10 pts wrong answer or no answer

Time complexity:
→ Traversing each column and each element in the column is $O(n^2)$
→ since all other parts of the algorithm involve traversing lists, which are $O(n)$ ∴ algorithm has complexity = $O(n^2)$

Correctness was proven at the beginning

b) Yes, multiple ways can exist to make a table unfavorable as if more than 1 row has a 1 in the column in (most significant column with odd number of 1s)
then any such row can be chosen for the move to make the table unfavorable

c) Take the current table, if there are only 2 rows and 1 row has just 1 match, then take the entire other row and force a win
else use the algorithm in part a) to play a move that makes the table unfavorable

2.3 5 / 5

✓ **- 0 pts** **Correct**

- 2 pts wrong answer, but show efforts

- 5 pts wrong answer

3. a) → ~~→~~

→ Deleting edge $v_a v_b$ from C_1, produces a disjoint path from v_a to v_b that contains all the vertices of C_1 exactly once

→ adding the edge $v_b v_c$, results in a path from

→ deleting edge $v_b v_d$ from C_2, gives a path from v_c to v_d that traverses all the edges of C_2 exactly once

→ adding edge $v_b v_c$ (given as a part of S) gives a path from v_a to v_d which traverses all vertices of C_1 and C_2 exactly once ⇒ it traverses all vertices $\in V$ exactly once as C_1 & C_2 partion V.

→ Finally, adding the edge from v_d to v_a (part of circle S) results in a cycle that traverses all the vertices exactly once

b) →

3.1 5 / 5

✓ **- 0 pts** **Correct**

- 1 pts fail to show which edges to remove/connect

- 5 pts no answer found

b) let the rows be labelled r_1 ... r_{16}, the columns be labelled c_1 ... c_{16}

Algorithm

→ Represent the board as an undirected graph G where each vertex has a label $r_i c_j$ s.t. $i,j \in \{1 \dots 16\}$ indicating which square it corresponds to and let 2 vertices have an edge connecting them if a knight can move from 1 square to the other in 1 move/step.

→ Partition the set of vertices in G into 4 equal sets s.t. $V_1, V_2 \dots V_4$ s.t. $r_i c_j \in V_1$ if $i,j \leq 8$

$r_i c_j \in V_2$ if $i > 8, j \leq 8$

$r_i c_j \in V_3$ if $i \leq 8, j > 8$

$r_i c_j \in V_4$ if $i > 8, j > 8$

→ Use the knight's tour on 8×8 board to find a cycle $C_i \; \forall i \in \{1 \dots 4\}$ corresponding to $V_i \; \forall i \in \{1 \dots 4\}$

→ Combine cycle C_1 & C_2 to form cycle C_4 by:

picking a vertex a s.t. there exists an edge between a and a vertex in V_2 (can be done trivially by picking an element on the border between C_1 and C_2 i.e the row or column in C_1 that is adjacent to a row or column in C_2 on the actual board

Let $a = r_i c_j$ s.t. $\deg(a) = 8$ ⇒ all legal moves from a are within the board $i = 8 \; j = 8$ is a corner square for the board of C_1

let $b = r_{i+2} c_{j-1}$ or $r_{i-1} c_{j+2}$ depending on which of the vertices is within C_1

Let $c = r$

Let $b = r_{i+2} c_j$ or $r_{i-1} c_{j+2}$ depending on which square $\in C_1$

Let $c = r_{i+3} c_{j+1}$

- Pick vertices b, c, d s.t. $b \in C_1$ and $c, d \in C_2$ and ab is an edge in C_1, cd is an edge in C_2 and ad bc are valid edges in G that belong to neither
- Delete ab and cd and add edges ad and bc to form the combined cycle (proved in part a)

Similary use the combine method to combine C_3 and $C_4 \rightarrow C_B$

Finally combine $(C_A, C_B) \rightarrow C_{ANS}$

Return C_{ANS} | Correctness proven through part a) |

c) Algorithm

Assume board is represented as an undirected graph G as mentioned in problem b)

→ Apply the known 8×8 knight's tour to the top right corner of the board and call this cycle C_{prev}

→ For each $i > 3$ and $i < k$:
- Duplicate the cycle in the top corner twice to create a square of side size $2^i \times 2^i$
- Split a sub board of size $2^i \times 2^i$ starts in the top right corner into 4 boards and apply C_{prev} to each of these boards to create 4 cycles that partition the vertices of the sub-board $(C_1, \ldots C_4)$ (these smaller sub boards will be of size $2^{i-1} \times 2^{i-1}$)
- Combine C_1 and C_2 (method explained in b)) → C_A
- Combine C_3 and $C_4 \rightarrow C_B$
- $C_{prev} \rightarrow$ Combine (C_A, C_B)

→ Return C_{prev}

3.2 10 / 10

✓ **- 0 pts** **Correct**

- 2 pts fail to show which vertices to connect

- 2 pts fail to show step by step explanation of the algorithm

- 4 pts fail to show why the algorithm works as expected

- 2 pts correct algorithm, but fail to show why the algorithm works as expected

- 10 pts no answer found

- 0 pts Click here to replace this description.

- Pick vertices b, c, d s.t. $b \in C_1$ and $c, d \in C_2$ and ab is an edge in C_1, cd is an edge in C_2 and ad bc are valid edges in G that belong to neither
- Delete ab and cd and add edges ad and bc to form the combined cycle (proved in part a)

Similary use the combine method to combine C_3 and $C_4 \rightarrow C_B$

Finally combine $(C_A, C_B) \rightarrow C_{ANS}$

Return C_{ANS} | Correctness proven through part a) |

c) Algorithm

Assume board is represented as an undirected graph G as mentioned in problem b)

→ Apply the known 8×8 knight's tour to the top right corner of the board and call this cycle C_{prev}

→ For each $i > 3$ and $i < k$:
- Duplicate the cycle in the top corner twice to create a square of side size $2^i \times 2^i$
- Split a sub board of size $2^i \times 2^i$ starts in the top right corner into 4 boards and apply C_{prev} to each of these boards to create 4 cycles that partition the vertices of the sub-board $(C_1, \ldots C_4)$ (these smaller sub boards will be of size $2^{i-1} \times 2^{i-1}$)
- Combine C_1 and C_2 (method explained in b)) → C_A
- Combine C_3 and $C_4 \rightarrow C_B$
- $C_{prev} \rightarrow$ Combine (C_A, C_B)

→ Return C_{prev}

Correctness - evident from part b) and part a)

Time complexity : Since all the operations within the loop. like combining 2 cycles, splitting up a board into 4 sub-boards, applying known cycles to vertices, obtaining a $2^i \times 2^i$ sub-board can be seen as $O(1)$ operations, the complexity is clearly $O(k)$

3.3 **10 / 10**

✓ **- 0 pts** **Correct**

- 3 pts fail to show the steps to reduce 2^k to 2^(k-1)

- 10 pts no answer found

where POS = n
SUM = 0

4. a) BinaryToN (Array B, INT POS, INT SUM) in first call

→ if pos = 0 then return sum

→ if B[pos] = 0

→ set pos to position of the first 1 from the right

- if none found return sum
- ~~else~~ sum = sum + (1 << (n - pos))

- Return BinaryToN (B, pos, sum)

→ else

→ set pos to position of the first 0 from the right

- if none found set pos = -1
- sum = sum + ((1 << (n - pos)) - 1)

Return • BinaryToN (B, pos, sum)

- This algorithm uses fewer additions as the worst case is 101010 10 where the algorithm groups together 2 digits at once resulting in n/2 adds instead of n adds.
- Also the algorithm has no multiplications and in the worst case n/2 left shifts which is way better than n multiplications as used in the brute force approach

Time complexity = O(n) as in the worst case the algorithm recurses for every element of B ~~has~~
(Note that while looking for the first 0 from the right is typically an O(n) operation, in this case the total when that the traversal does occur for longer than 1 element it eliminates an equal number of recursive calls, allowing the time complexity to stay at O(n))

4.1 **15 / 15**

✓ **- 0 pts** **Correct**

- 3 pts incorrect function header

- 0 pts Click here to replace this description.

- 15 pts no answer found

b) BinaryToN(Array B)

→ N ~~→ 0~~ Sum → 0

→ pos = n

→ While (pos > 0)

- if B[pos] = 0 then set pos to the ~~right-most~~ first 1 to it's left
  - → if none found then break ~~out of loop~~ out of loop
  - → sum += 1 << (n - pos)
- else
  - → set pos to the first 0 to it's (pos') left
  - → if none found set pos → -1
  - → sum += (1 << (n - pos)) - 1

→ Return Sum

Time complexity and correctness follow from the arguments made for the recursive version.

4.2 10 / 10

✓ **- 0 pts** **Correct**

- 15 pts no answer found