Combinatorics

Jacob P., Caitlin N., Dan C.

Introduction to the lab and how the math and computer science interact and assist each other:

A Magician asks you to pick five cards from a 52 card deck and then takes the cards from you and hands one card back. They then display the remaining cards and tell their assistant to come over. The assistant walks in and engages with you for a few moments and then guesses your card correctly, seemingly reading your mind.

What seems like magic to you was actually achieved through careful manipulation of the cards displayed. The Magician looked at the five cards and saw which suits had duplicates. After doing that, he gave one of those cards to you and put the other matching suit card in the first position on display. The remaining three cards would then be permuted in such a way that the ordering of the cards signified how much to add onto the card in the first position. All the assistant has to do is figure out in their head what order the last three cards are in and then add the permutation order to the first card, hence the delaying tactics.

The way this card trick was done was not the most efficient version of this trick. The magician and their assistant can increase the number of cards in the deck but at a certain point it becomes almost impossible to do the calculations in their mind and this is where the computer comes in to optimize this trick. The computer is able to maximize the efficiency of this trick by utilizing computer science and mathematics. It is able to construct and decipher the trick through the use of mathematical algorithms which are enhanced and applied through computer science.

Programs:

Program 1:

In this program, we created two methods, decode and encode. Our encode method takes two parameters, two integers (n and m), which then outputs the mth permutation of the elements {1,2,3,4...n}. To accomplish this, we created an array and set the values from 0 to n. Following this, we used a loop to check if the next index of the array was greater than the current, and if this was true, then we swap these values, and if the permutation value was not zero we would permute until we got the mth permutation. In the decode method we take in an integer n, and a permutation p, with these we output a value m, where p is the mth permutation of n. In this method we use our encode method, passing in the parameter n to be the 'n' value in the encode method, and as the integer m in our encode method we use an integer 'i'. We call the encode method n! times, each time the integer 'i' is incremented. Each iteration, we check to see if the encoded value is equal to 'p' and if this is true, we return the value of 'i' at that iteration.

```
Enter number of digits: 5
Enter number of permutations: (1 to n!) 25
21345
Enter a permutation to find its index: 52314
105
```

Program 2:

This program simulates choosing 5 cards from the 28 available, hiding the lowest one of those cards, and permuting the remaining cards to show what the removed card was. We achieved this by having the user input a set of 5 cards and storing them in an array in ascending order. The lowest card was then removed and stored as the number of times the remaining cards needed to be permuted. The permutation method that was used was the same as the one in Program 1, so the cards were shifted around the array 'removed-card' amount of times to display the correct permutation ordering. The second part of this program takes a set of four numbers from the user and finds out what permutation order they are in to find the removed card. The input from the user was stored in an array and saved and duplicated. This duplicated array would then be reorganized to have the numbers in ascending order and then be permuted until it matches the original array. The number of times that would take would then be the missing card.

```
Enter 5 numbers (1 - 28):
12
15
6
14
25
6
1432
12|25|15|14|
Enter 4 numbers (1 - 28) to find the permutation:
13
28
1
14
2413
11
```

Program 4-5:

This program essentially works as a cheat sheet the magician can use to find the missing number from an ordered pair of 3 different numbers. The first thing that our program does is create an array of combinations holding 56 slots, these slots are then filled by the combos {123, 124, 125, …, 578, 678}. Each digit in the number combination is assigned a letter value {a, b, c} so that they can be moved around easier. The next step is to order these combos and send them to the fixed-combo array. The combos are ordered by assigning the combos to an ordered pairing (ba, ab, ca, ac, cb, bc), this ordering is known to be successful. Each combo will start at the first pairing of the array and if that fixed-combo hasn't been taken before, it will be added to the fixed-combo array. We then check to see if a fixed-combo hasn't been on the fixed-combo array before by walking through each position of the array and comparing the two fixed-combos to see if they are the same, if so, the programs will move on to the next ordered pairing. If none of the ordered pairings works, the program backtracks to the previous fixed-combo and has it try the next ordered pairing. This process will continue until either the fixed-combo array has been filled out with no duplicate fixed-combos or a maximum number of backtracking has occurred which will cease all operations. The results are then outputted.

```
ba ab ca ac cb bc
123 124 125 126 127 128 134 135 136 137 138 145 146 147 148 156 157 158 167 168 178 234 235 236 237 238 245 246 247 248 256 257 258 267 268 278 345 346 347 348 356 357 358 367 368 378 456 457 458 467 468 478 567 568
21| 12| 51| 61| 71| 81| 31| 13| 16| 17| 18| 41| 14| 74| 84| 15| 75| 85| 76| 86| 87| 32| 23| 62| 72| 82| 42| 24| 27| 28| 52| 25| 58| 26| 68| 78| 43| 34| 37| 48| 63| 35| 83| 67| 38| 73| 65| 45| 54| 64| 46| 47| 57| 56|
```

```
ab ac bc ba ca cb
123 124 125 126 127 128 134 135 136 137 138 145 146 147 148 156 157 158 167 168 178 234 235 236 237 238 245 246 247 248 256 257 258 267 268 278 345 346 347 348 356 357 358 367 368 378 456 457 458 467 468 478 567 568
12| 14| 15| 16| 17| 18| 13| 35| 36| 37| 38| 45| 46| 47| 48| 56| 57| 58| 67| 68| 78| 23| 25| 26| 27| 28| 42| 24| 74| 84| 52| 72| 85| 76| 62| 82| 54| 64| 73| 43| 65| 75| 83| 63| 86| 87| 65| 75| 0| 0| 0| 0| 0| 0| 0| 0|
```

```
cb ba ca ab bc ac
123 124 125 126 127 128 134 135 136 137 138 145 146 147 148 156 157 158 167 168 178 234 235 236 237 238 245 246 247 248 256 257 258 267 268 278 345 346 347 348 356 357 358 367 368 378 456 457 458 467 468 478 567 568
32| 42| 52| 62| 72| 82| 43| 53| 63| 73| 83| 54| 64| 74| 84| 65| 75| 85| 76| 86| 18| 23| 35| 26| 27| 38| 45| 46| 47| 28| 56| 25| 58| 67| 68| 78| 34| 36| 37| 48| 56| 37| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0|
```

Results for Pairings...

(ba, ab, ca, ac, cb, bc): All but the last 8 slots have been filled in leading us to believe that the program would need more time to backtrack to successfully reach the completed table.

(ab, ba, ac, ca, bc, cb) & (cb, ba, ca, ab, bc, ac): Both of these combinations failed to reach a completed table before the exit condition was reached.

Program 6:

This program uses a more practical method to fill out an entry table of 56. First, our program fills out an array of 56 slots with combos {123, 124, 125, …, 578, 678}. Next, we go through the combos, adding them up and then moding them by 3. Whatever the remainder is, it will be the number in that position that will be hidden and the remaining numbers will be added to the finished array. If the remaining numbers are already on the array, this is figured out by going through the entirety of the array comparing the two numbers, the numbers are then switched and are added to the finished array. The finished array is then printed.

```
123 124 125 126 127 128 134 135 136 137 138 145 146 147 148 156 157 158 167 168 178 234 235 236 237 238 245 246 247 248 256 257 258 267 268 278 345 346 347 348 356 357 358 367 368 378 456 457 458 467 468 478 567 568
2–3 1–4 1–2 2–6 1–7 2–1 1–3 3–5 1–6 3–1 3–8 1–5 4–1 4–7 1–8 5–6 7–1 5–1 6–1 6–8 8–1 3–4 2–5 3–2 3–7 2–8 2–4 4–6 2–7 4–2 6–2 5–2 5–8 6–7 8–2 7–2 4–5 3–6 4–3 4–8 5–3 5–7 8–3 7–3 6–3 7–8 6–5 7–4 5–4 6–4 8–6 8–4 7–6 8–5
```

Program 7:

This program takes the intuitive approach with 124 cards, it takes in 5 values from the user and returns the subset of these 5 values. To find the subset of the cards, we look at each card the user inputs, and determine which one to hide, and then iterate through every card in the deck and store the possible cards in an array and find the permutation of that index. To find the permutation of the possible index, we use the encoding and decoding methods that we developed in the first program. When we determine which card to hide, the index of the hidden card in the array it is stored in is the number of permutations that must be done in order to get the correct order for the remaining 4 cards. After this was complete, we modified the program to allow the user to input a set of 4 cards, in the correct permutation, and the program returns the hidden card. To accomplish this, we iterated through the deck and storing the 24 possibile cards in an array. Because the user inputs the cards in order, the program is able to use a decode method to find the permutation. The resulting permutation from the decode method, initially developed in program 1, is used to take this card out of the array.

Exercises:

Exercise 1:

Lower bound

1. Choose 4 cards, show 3, 3 suits, max cards?

$\boxed{A}$ B C D

Hidden Suit Permutaitions: $2! = 2\cdot 1 = 2\cdot 2+1 = 5\times 3 = \boxed{15 \text{ cards}}$

2. Chose 3, show 2, 2 suits, max cards?

$\boxed{A}$ B C

Hidden Suit Permutaitons: $1! = 1\cdot 2+1 = 3\times 2 = \boxed{6 \text{ cards}}$

Chose 2 show 1, 1 suit, max cards?

$\boxed{A}$ B

Hidden Suit Permutaitons: $0! = 0\cdot 2+1 = 1\times 1 = \boxed{1 \text{ card}}$

3. Chose 6, show 5, 5 suits

$\boxed{A}$ B C D E F

Hidden Suit Permutation: $4! = \underset{48+1}{24\cdot 2+1} = 49\times 5 = \boxed{245 \text{ cards}}$

4. Generalize the results above with subset of n-1:

$\boxed{C = (2(n-2)!+1)\underbrace{(n-1)}_{\text{Suits}}}$

> upper bounds Arguments on # of cards

(5) upper bound for $n=5$ is 124 cards
what about $n = 2, 3, 4 \rightarrow n! + (n-1)$
> $2 = 2! + (2-1) = 2+1 = 3$
> $3 = 3! + (3-1) = 3\cdot2\cdot1 + (2) = 8$
> $4 = 4! + (4-1) = 4\cdot3\cdot2\cdot1 + (3) = 27$

(6) $u = n! + (n-1)$ $\quad 5\cdot4\cdot3\cdot2\cdot1 + 4$
$20 \times 3 \quad 120+4 = 124$ ✓

(7) ways to choose n cards from u
> $f(n) = \boxed{\binom{n!+(n-1)}{n}}$

> number of ways to order n-1 cards from u.

$$\boxed{\binom{n!+(n-1)}{n-1}} = \frac{n!+(n-1)\cdot(n-1)!+((n-1)-1)}{n(n-1)(n-2)} = \frac{[n!+(n-1)](n-1)!\,\cancel{(n-2)}}{n(n+1)(n-2)} = \binom{n!+(n-1)}{n}$$

(8) using largest deck of cards possible. let n be the # of cards chosen, where the accomplice chooses an ordered subset of n-1. Calculate formulas in terms of n for

a) # of ways the contestant can choose his card

$T(n) = \frac{k!}{(k-n)!}$ (where k is deck size) contestant has n cards to chose from, then n-1 choices, then n-2 etc.

b) $n(n-1)! = P(n)$ → there are n options, one is chosen to hide, then there are n-1 possible permutations

number tables? list of permutations (n-1)

c) n! permutations, how many lists $\boxed{n(n-5)!}$ } multiply by pos

* 5 possible cards, 5, 4, 3, 2, or 1, count down

Exercise 3:

For 6 Cards

123	124	125	126	134	135	136
2-1	1-2	5-1	6-1	3-1	1-3	1-6
234	235	236	345	145	146	156
3-2	2-3	6-2	4-3	4-1	1-4	1-5
356	456	245	246	256	346	
5-3	5-4	4-2	2-4	5-2	3-4	

For 7 Cards

123	124	125	126	127	134	135
2-1	1-2	5-1	6-1	7-1	3-1	1-3
136	137	145	146	147	156	157
1-6	1-7	4-1	1-4	7-4	1-5	7-5
167	234	235	236	237	245	246
7-6	3-2	2-3	6-2	7-2	4-2	2-4
247	256	257	267	345	346	347
2-7	5-2	2-5	2-6	4-3	3-4	7-3
356	357	367	456	457	467	567
5-3	3-5	6-3	5-4	4-5	6-4	6-5

Impressions of the lab and/or enrichment: Discuss what you liked best, what you liked least, and what you learned:

Two magicians perform a trick where one chooses 5 cards at random and asks the second magician to find out what the missing random card is, just by looking at the other 4 cards presented to them. After watching the magic trick a few times, we soon discovered it had a dependency on mathematics, and we were able to turn the magic trick into a computer program.

While programming in this lab, one aspect we liked was being able to see the magic trick and then apply our understanding of this trick and turn it into a science and math-based project. Being able to watch the connection between a real-world application and computer science is rewarding. This was especially satisfying when we had the epiphany of what needed to be done and watched as the numbers lined up with a potential scenario in real life.

Although the programming was not too complicated, something we did not enjoy as much in this lab was understanding how to approach each program. We would be unsure of how to start off the problem or what was directly being asked, but once we got the ball rolling and it clicked for us, it was not as difficult as we previously assumed. The programs were intuitive to understand, especially after attempting the trick on our own a few times to ensure our understanding, but we had trouble grasping the different scenarios, namely the use of 124 cards and the counter-intuitive methods we had to develop when we changed rolls from the guesser to the one who permutes the deck. Despite the minor short-comings, we thoroughly enjoyed this project, as it allowed us to have results that were not only believable but tangible, as we could do this trick on our own.