Algorithms And Data Structures Report **06/02/24**

Group Members

Member	Name	ID	Contribution %
1	Abdullah Ali	001305726	25
2	Austin Obodo (Leader)	001312468	25
3	Jake Cunningham	001211278	25
4	Zaid Kadir	001323108	25

Task 1 Create Unique Solutions

Austin's approach and understanding of the problem

I understood that there were ranges that had to be set between m and n. The numbers that were special are prime and palindromic which I had to find. A list named special had to be set. I would also need a count variable to set to 0.I realised I had to use a for loop or library to check for the prime numbers and use a for loop to check if the prime numbers are palindromic. After this, if a special number was found the count increased. I realised I would need to check if the count was less than six to display all the numbers. If not, then the first 3 and last 3 numbers of the sorted list of special numbers would be displayed.

Code:

```
#importing time
import time


#SOLUTION 1
def specialNums(m,n):
    #Checking execution time
    start=time.time()
    count=0
    numbers=[]
    #looping through numbers
    #adding numbers to list
    for i in range(m,n+1):

        #Chaning number to string to reverse it
        sub=str(i)
        #Comparing reversed number to original number
        if int(sub[::-1])==i:
            #Checking for primes
             if (i == 2 or i % 2 != 0) and i != 1:
                numbers.append(i)
                count=count+1

    if count<6:
        print(count,":",numbers,"\n")

    elif count>=6:
        #Comparing values in the array
        for b in range(0,len(numbers)-1):
            if numbers[b]>numbers[b+1]:
                numbers[b]==numbers[b+1]
```

```
                numbers[b+1]==numbers[b]

                #Displuaing first three smallest and last three largest
        print("First Three smallest",numbers[0],numbers[1],numbers[2],"\n",
              "Last Three largest",numbers[-3],numbers[-2],numbers[-
1],"\n",
              "Total number of special numbers",count,"\n")
        #Checking execution time
    end=time.time()
    final=end-start
    print("Execuction time",final)
```

Jake's understanding and approach

The brief explains that an algorithm needs to be made that can count and display special numbers, these are numbers that are both prime and palindromic. The brief also requested that specific ranges, M and N, are entered and the algorithm will find the special numbers between, then if there was 6 or less it would display them all, if greater than 6, it would display the first and last 3 along with a counter that displays the total of the special numbers in the range.

To achieve this, I would have to create a program that can take the input of numbers "m" and "n", find out if the number is prime, if the code finds the number is prime, it will then check if it is palindromic. If the number is also palindromic it will append the number to the end of the list and then add 1 to the counter. Once the sequence has all been checked it will display the numbers are explained in the brief.

Code:

```
1.    import time
2.
3.    def is_prime(num):
4.        if num < 2:
5.            return False
6.        for i in range(2, int(num ** 0.5) + 1):
7.            if num % i == 0:
8.                return False
9.            return True
10.
11.    def is_palindrome(num):
12.        return str(num) == str(num)[::-1]
13.
14.    def find_special_numbers(m, n):
15.        special_numbers = []
16.        for num in range(m, n+1):
17.            if (num) is_prime and is_palindrome(num):
18.                special_numbers.append(num)
19.        return special_numbers
20.
21.    m = int(input("Enter the smaller number: "))
22.    n = int(input("Enter the larger number: "))
23.
24.    start_time = time.time()
25.    special_numbers = find_special_numbers(m, n)
26.
27.    if len(special_numbers) < 6:
28.        print("All special numbers between", m, "and", n, "are:")
29.        for num in special_numbers:
30.            print(num)
31.    else:
32.        print("The first three smallest special numbers between", m, "and", n,
"are:")
```

```
33.         for i in range(3):
34.             print(special_numbers[i])
35.         print("The last three biggest special numbers between", m, "and", n,
"are:")
36.         for i in range(len(special_numbers)-3, len(special_numbers)):
37.             print(special_numbers[i])
38.
39.     print("Total number of special numbers between", m, "and", n, "is:",
len(special_numbers))
40.
41.     print("--- %s seconds ---" % (time.time() - start_time))
```

Zaids Understanding and Approach

According to the brief, a special number counting and displaying algorithm must be created. These will be numbers that are both prime and palindromic. In addition, the brief asked for an input of two specific ranges, M and N, between which the algorithm is meant to find any special numbers. If the number of special numbers is six or less, it should display them all; if it is greater than six, it should only display the first 3 and last 3 special numbers along with a counter that shows the total number of special numbers in the range. To solve the problem, I would first use the Sieve of Eratosthenes algorithm to efficiently find all prime numbers in the given range. This algorithm will iterate overall numbers up to the square root of the upper bound of the range and for each prime number found, it marks all its multiples as not prime.

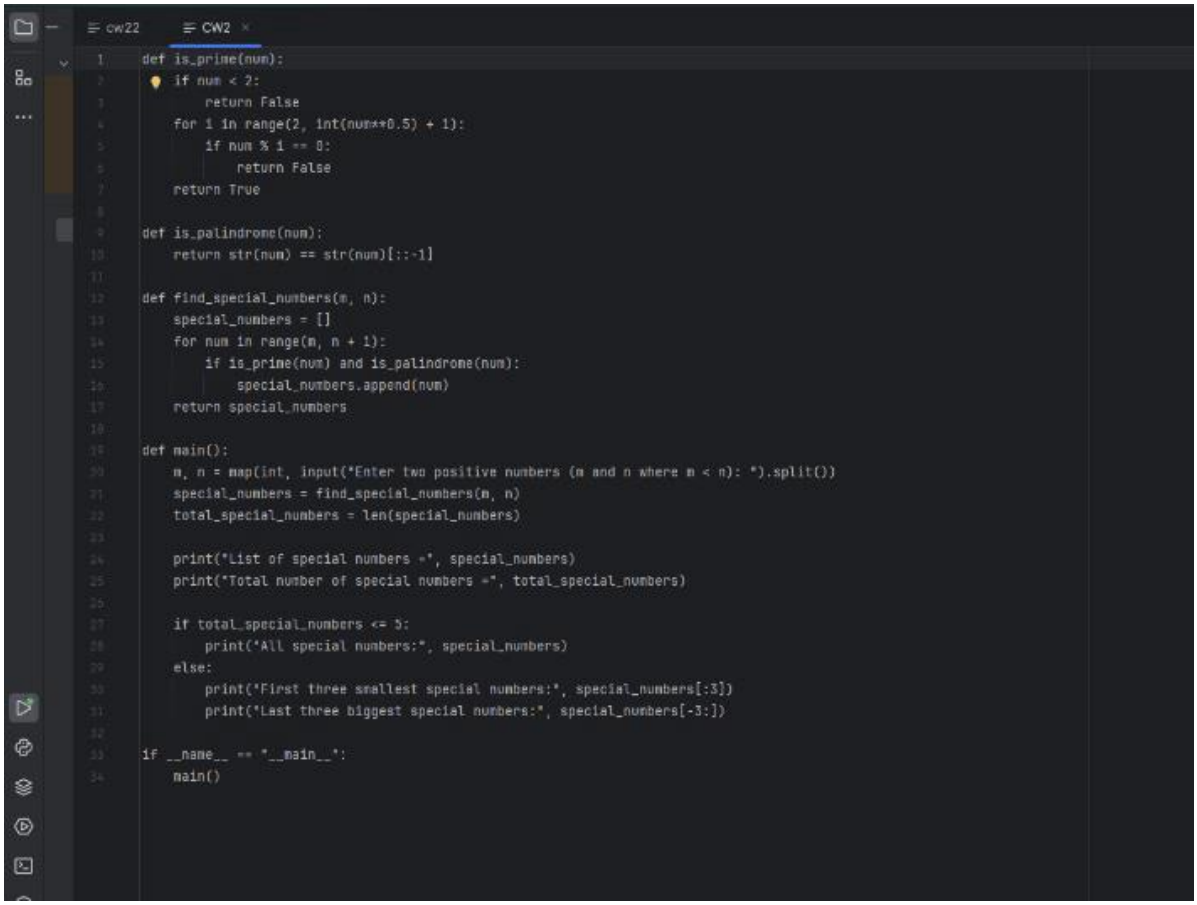
Code:

```
import timeit

def is_palindromic(num):
    str_num = str(num)
    reversed_str_num = str_num[::-1]
    return str_num == reversed_str_num

def sieve_of_eratosthenes(n):
    prime = [True] * (n + 1)
    prime[0], prime[1] = False, False
    for i in range(2, n + 1):
        if prime[i]:
            for j in range(i * i, n + 1, i):
                prime[j] = False
    return prime

def find_special_numbers(m, n):
    prime = sieve_of_eratosthenes(n)
    special_numbers = [num for num in range(m, n + 1) if prime[num] and
is_palindromic(num)]
    return special_numbers

m, n = 1, 2000
time_taken = timeit.timeit(lambda: find_special_numbers(m, n), number=1)
special_numbers = find_special_numbers(m, n)

print(f"Total: {len(special_numbers)} Special Numbers")
if len(special_numbers) < 6:
    print(special_numbers)
else:
    print(special_numbers[:3] + special_numbers[-3:])
print(f"Time taken: {time_taken:.10f} seconds")
```

Abdullah Understanding and approach.

Reviewing the problem statement, it became clear that the task involved creating a Python program to find special numbers within a given range. Special numbers are defined as prime numbers that are also palindromic. The program should prompt the user to input two positive integers, M and N, where M is smaller than n, and then identify and display the special numbers within that range. Additionally, the program should handle cases where there are fewer than 6 special numbers by displaying all of them and cases where there are more than 6 special numbers by displaying the first three smallest and the last three biggest special numbers. This case we are tasked with creating python program that identifies special numbers between 2 positive numbers.

Code:



Task 2: Test cases for each of our code

Austins solution

Number	Input (m, n)	Output	Running time
1	1,2000	First Three smallest 2 3 5 Last Three largest 797 919 929 Total number of special numbers 20	0.0009980201721191406
2	100,10000	First Three smallest 101,131,151 Last Three largest 797 919 929 Total number of special numbers 15	0.003045320510864258
3	20000,80000	First Three smallest 30103,30203, 30403 Last Three largest 79397 79697 79997 Total number of special numbers 48	0.01827859878540039
4	100000,2000000	First Three smallest 1003001 1008001 1022201 Last Three largest 1993991 1995991 1998991 Total number of special numbers 190	0.5661211013793945
5	2000000,9000000	First Three smallest 3001003 3002003 3007003 Last Three largest 7985897 7987897 7996997	3.500065565109253
6	10000000,100000000	First Three smallest 10000001 10011001 10022001 Last Three largest 99977999 99988999 99999999 Total number of special numbers 5000	108.94590711593628
7	100000000,400000000	First Three smallest 100030001 100050001 100060001 Last Three largest 399737993 399767993 399878993 Total number of special numbers 2704	151.01931309700012
8	1100000000,1500000000		
9	15000000000,100000000000		
10	1,1000000000000		

Jake's Solution

#	Input (m n)	Output (Total: Special Numbers)	Comments
1	1 - 2_000	2, 3, 5 - 797, 919, 929	20 numbers 0.002 Seconds
2	100 - 10_000	101, 131, 151 - 797, 919, 929	15 numbers 0.006 Seconds
3	20_000 - 80_000	30103, 30203, 30403 - 79397, 79697, 79997	48 Numbers 0.055 Seconds
4	100_000 - 2_000_000	1003001, 1008001, 1022201 – 1993991, 1995991, 1998991	190 Numbers 5.012 Seconds
5	2_000_000 - 9_000_000	3001003, 3002003, 3007003 – 7985897, 7987897, 7996997	327 Numbers 37.287 Seconds
6	10_000_000 - 100_000_000		0 Numbers 1284.629 Seconds (21.41 Minutes)
7	100_000_000 - 400_000_000		Inconclusive

Zaids Solution

#	Input (m,n)	Output (special numbers)	Comments
1	1 – 2_000	[2, 3, 5] – [797, 919, 929]	20 Numbers 0.0004532 s
2	100 – 10_000	[101, 131, 151] – [797, 919, 929]	15 Numbers 0.0015415 s
3	20_000 – 80_000	[30103, 30203, 30403] – [79397, 79697, 79997]	48 Numbers 0.012988 s
4	100_000 – 2_000_000	[1003001, 1008001, 1022201] – [1993991, 1995991, 1998991]	190 Numbers 0.3473778 s
5	2_000_000 – 9_000_000	[3001003, 3002003, 3007003] – [7985897, 7987897, 7996997]	327 Numbers 1.44095 s
6	10_000_000 – 100_000_000	None	0 Numbers 16.1672738 s
7	100_000_000 – 400_000_000	[100030001, 100050001, 100060001] – [399737993, 399767993, 399878993]	2704 Numbers 72.3650361 s
8	1_100_000_000 – 15_000_000_000		

9	15_000_000_000 – 100_000_000_000		
10	1 – 1_000_000_000_000		

Task 3 Optimised solutions

Solution 1

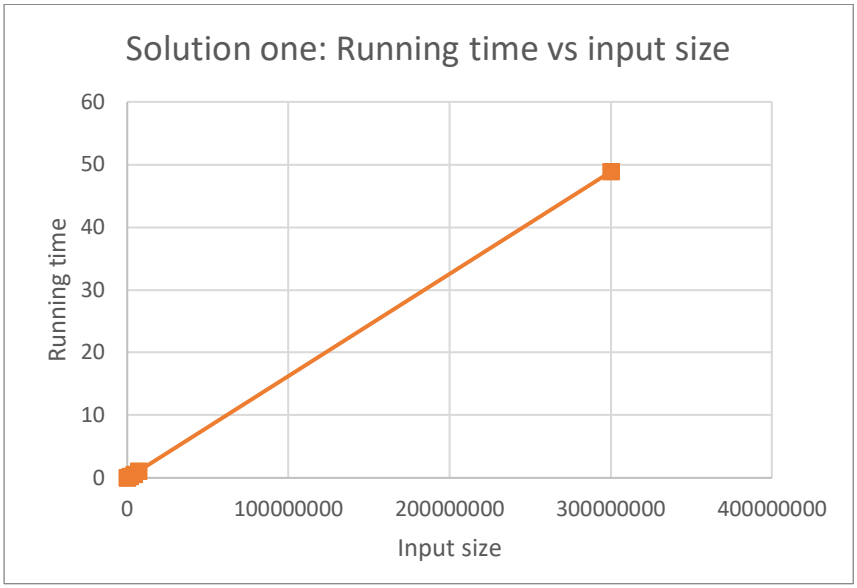
For solution 1, it was determined that the bottle neck of the code was that each number between m and n was being checked to see if it was both a palindrome number and a prime number separately, even if the code had discovered that one was already false. This method isn't as noticeable when the numbers are relatively small like in the initial test cases, however it is quickly proven inefficient when the volume of numbers becomes larger. It was determined to make one of the checks the initial check, if this determined that a number was not prime the code would not check for palindrome numbers. This improved the speed but not significantly, instead the code was changed to check for a prime number first, then if the number was not prime the code would move onto the next number. However, if the number was prime, an if statement would run and then the code would check if the number was a palindrome. This greatly reduced the time that the tests would take, for example, test 6 went from taking 108.9 seconds to only 13.5 seconds an 87.6% improvement.

```
import time
#Chek if number is prime
def is_prime(num):
    #before this is_palindrome will have run so this if statement can
run
    if is_palindrome == False:
        return False
    else:
        #if number is less than 2 just move on
        if num < 2:
            return False
        #finds the square root, add 1, create a sequence from 2 to
'num' +1.
        for i in range(2, int(num ** 0.5) + 1):
        #then check if any are left over when divided by whatever i is.
            if num % i == 0:
                return False
        return True

def is_palindrome(num):
    #takes the number and compares it to the same number backwards
    return str(num) == str(num)[::-1]

#this runs each class above when called
def find_special_numbers(m, n):
    special_numbers = []
    for num in range(m, n+1):
        if is_palindrome(num) and is_prime(num):
            special_numbers.append(num)
    return special_numbers

m = int(input("Enter the smaller number: "))
n = int(input("Enter the larger number: "))

start_time = time.time()
special_numbers = find_special_numbers(m, n)
```

```
#displays all 6 numbers if there are ,6
if len(special_numbers) < 6:
    print("All special numbers between", m, "and", n, "are:")
    for num in special_numbers:
        print(num)
#other wise show only the first 3 and last 3
else:
    print("The first three smallest special numbers between", m, "and",
n, "are:")
    for i in range(3):
        print(special_numbers[i])
    print("The last three biggest special numbers between", m, "and",
n, "are:")
    for i in range(len(special_numbers)-3, len(special_numbers)):
        print(special_numbers[i])

print("Total number of special numbers between", m, "and", n, "is:",
len(special_numbers))
#logs how long the code took to run
print("--- %s seconds ---" % (time.time() - start_time))
```

#	Input (m n)	Output (Total: Special Numbers)	Comments
1	1 - 2_000	2, 3, 5 - 797, 919, 929	20 numbers 0.00096 seconds
2	100 - 10_000	101, 131, 151 - 797, 919, 929	15 numbers 0.002 Seconds
3	20_000 - 80_000	30103, 30203, 30403 - 79397, 79697, 79997	48 Numbers 0.010 Seconds
4	100_000 - 2_000_000	1003001, 1008001, 1022201 – 1993991, 1995991, 1998991	190 Numbers 0.296 Seconds
5	2_000_000 - 9_000_000	3001003, 3002003, 3007003 – 7985897, 7987897, 7996997	327 Numbers 1.067 Seconds
6	10_000_000 - 100_000_000		0 Numbers 13.539 Seconds
7	100_000_000 - 400_000_000	100030001, 100050001, 100060001 – 399737993, 399767993, 399878993	2704 Numbers 48.839 Seconds
8	1_100_000_000 - 15_000_000_000	10000500001, 10000900001, 10001610001 – 14998289941, 14998589941, 14998689941	5474 Numbers 5151.305 Seconds (1.43Hours)

9	15_000_000_000 - 100_000_000_000		Inconclusive



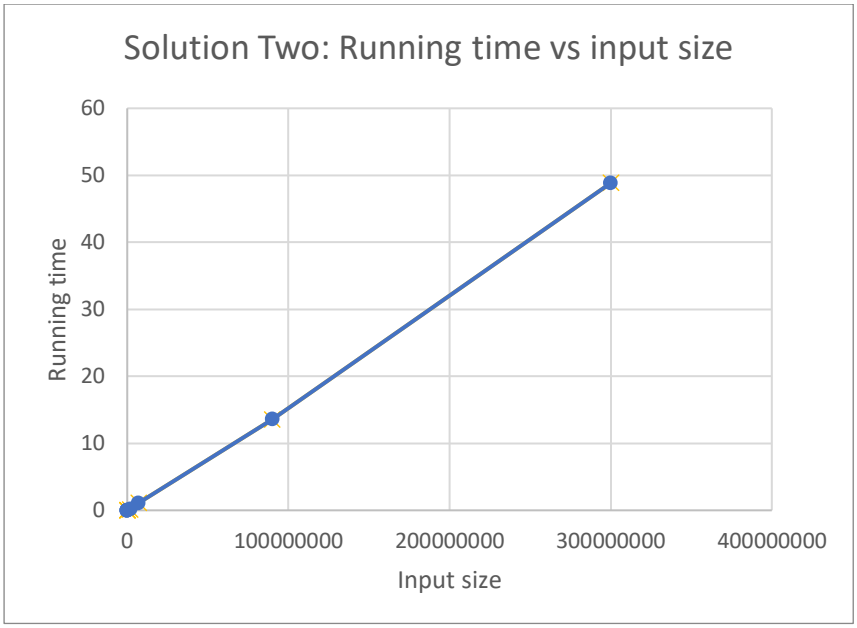
Solution 2

The updated code in the question simplifies the 'is_palindromic' function by directly reversing the number using integer arithmetic instead of converting it to a string and reversing it. This optimisation reduces the number of operations required to check if a number is a palindrome. The 'sieve_of_eratosthenes' function has also been optimised by starting the loop from 2 instead of 0 and only marking multiples of prime numbers as non-prime. Both these optimizations I have implemented have managed to reduce the time complexity of the algorithm. For test case 7 the optimisations have managed to reduce the run time from 72.3650361 seconds to 47.8805808 seconds which is a 33.85% improvement from my initial code.

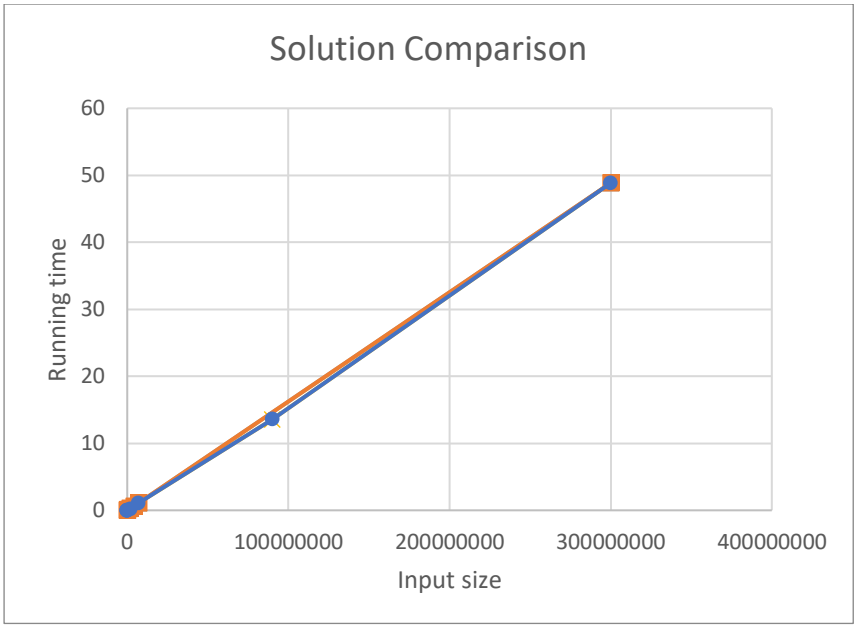
```
import timeit

def is_palindromic(num):
    original_num, reversed_num = num, 0
    while num > 0:
        reversed_num = reversed_num * 10 + num % 10
        num //= 10
    return original_num == reversed_num

def sieve_of_eratosthenes(n):
    prime = [True] * (n + 1)
    prime[0], prime[1] = False, False
    p = 2
    while p * p <= n:
        if prime[p]:
            for i in range(p * p, n + 1, p):
                prime[i] = False
```

```
        p += 1
    return prime

def find_special_numbers(m, n):
    prime = sieve_of_eratosthenes(n)
    special_numbers = [num for num in range(m, n + 1) if prime[num] and
is_palindromic(num)]
    return special_numbers

m, n = 100000000, 400000000
time_taken = timeit.timeit(lambda: find_special_numbers(m, n), number=1)
special_numbers = find_special_numbers(m, n)

print(f"Total: {len(special_numbers)} Special Numbers")
if len(special_numbers) < 6:
    print(special_numbers)
else:
    print(special_numbers[:3] + special_numbers[-3:])
print(f"Time taken: {time_taken:.10f} seconds")
```

#	Input (m,n)	Output (special numbers)	Comments
1	1 – 2_000	[2, 3, 5] – [797, 919, 929]	20 Numbers 0.0001743 s
2	100 – 10_000	[101, 131, 151] – [797, 919, 929]	15 Numbers 0.0008258 s
3	20_000 – 80_000	[30103, 30203, 30403] – [79397, 79697, 79997]	48 Numbers 0.00534 s
4	100_000 – 2_000_000	[1003001, 1008001, 1022201] – [1993991, 1995991, 1998991]	190 Numbers 0.1678358 s
5	2_000_000 – 9_000_000	[3001003, 3002003, 3007003] – [7985897, 7987897, 7996997]	327 Numbers 0.7944658 s
6	10_000_000 – 100_000_000	None	0 Numbers 10.6204737 s
7	100_000_000 – 400_000_000	[100030001, 100050001, 100060001] – [399737993, 399767993, 399878993]	2704 Numbers 47.8805808 s
8	1_100_000_000 – 15_000_000_000		
9	15_000_000_000 – 100_000_000_000		
10	1 – 1_000_000_000_000		


Solution Two: Running time vs input size
Running time
60
50
40
30
20
10
0
0
100000000
200000000
300000000
400000000
Input size

Task 4 Comparison of code



When comparing the two optimised solutions together and plotting their run time into a graph. The graph shows how the two codes run quite similar as the numbers grow. Solution 2 runs faster with small numbers such as in test case 1-4, however by test 7, the run times are similar relatively speaking. Overall Solution 2 seems to be the more efficient code as utilises less code and runs just slightly faster overall.

Task 5 Limitations faced.

Limitations faced were communication issues. It was difficult to communicate with each other, because there was no group chat set up around the end of January. I emailed my members in Group 6 about starting a group chat in Microsoft Teams on 05/02/2024.

When the chat was made it took time for people to notice. But as the days went by Zaid, Abdullah and Jake noticed the chat and we were able to communicate about the group work.

Group management was difficult as some people in the group took a while to respond to messages concerning their progress. This made it difficult to check how we were doing on the coursework and caused slight delay.

Weekly Journal

Week and date	Task note	Status
29th of January – 4th February	**Austin set up a Teams chat.**	**Complete**
5th of February- 11th February	**Discuss the coursework and what we need to do.**	**Complete**
12th February – 18th February	**Met up with each other (Rupok did not respond to the Teams group chat). Sent**	**Complete**

	everyone in the group an email.	
19th February – 25th February	**Met up in class and discussed how each of our code is going**	**Complete**
26th February- 3rd March	**Each of us have made our code for task 1.**	**Complete**
4th-10th March	**Group completed task 1 and task 2.**	**Complete**
11th March	**Our group decided to use Jakes and Zaids code to optimise.**	**Complete**
12th March-16th March	**Completed Task 4 and Task 5**	**Complete**