COMP 1828 - Designing, developing, and testing solutions for the London Underground system.

TASK 1

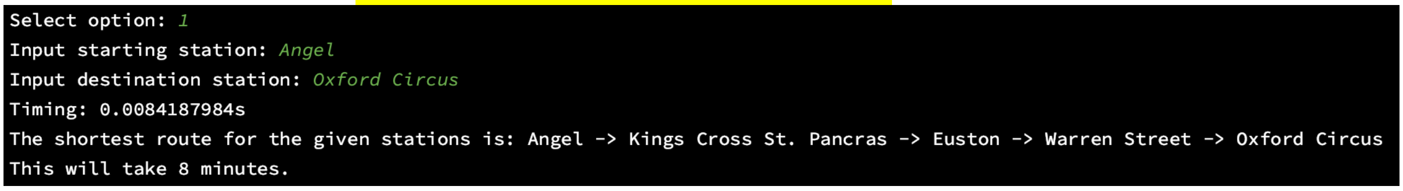
1. Rationale for selecting specific algorithms and data structures, complemented by empirical performance analysis using synthetic data of varying sizes.

The algorithm we employed for task 1 is the Dijkstra Algorithm. The aim of task 1 is to find 'the shortest journey duration in minutes between a specified starting and destination station'. The Dijkstra's Algorithm can find the shortest path between individual nodes of a graph. When we conducted research on this algorithm, it resulted in us finding that it has many real-world applications. Dijkstra's Algorithm is commonly used in GPS devices to find the shortest path between a location and the destination. This drew to the conclusion that this would be a suitable algorithm, when developing a software model for the 'London Underground Tube System's Route Planner', as will be explained in point 4.

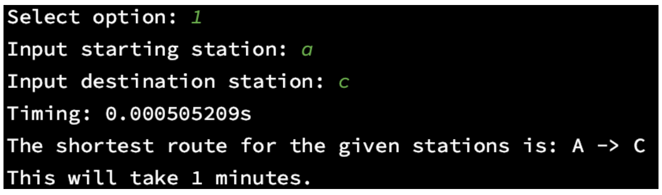
We opted to use the adjacency list graph as our data structure. Within this, each station is a node, and each edge represents the connection between the stations. The London Underground system is a sparse graph, it has less edges(connections) compared to a fully connected graph with the maximum possible edges. It contains many stations across various lines, where stations do not all have direct connections with one another. The adjacency list graph allows for an efficient representation of sparse graphs because it only keeps information about the existing edges.

Additionally, we decided to utilise the adjacency list graph because the Dijkstra algorithm benefits from it. The Dijkstra algorithm works by iterating through close stations of a station and updating the distances. The adjacency list gives the algorithm access to close stations which improves the overall performance of the algorithm.

TASK	ATTEMPT	DATA SET (TUBE MAP)	DATA SET (TEST)
1	1	0.008418798	0.00117898
	2	0.008829117	0.001134157
	3	0.00798893	0.00122714
	AVERAGE	0.008412282	0.001180093



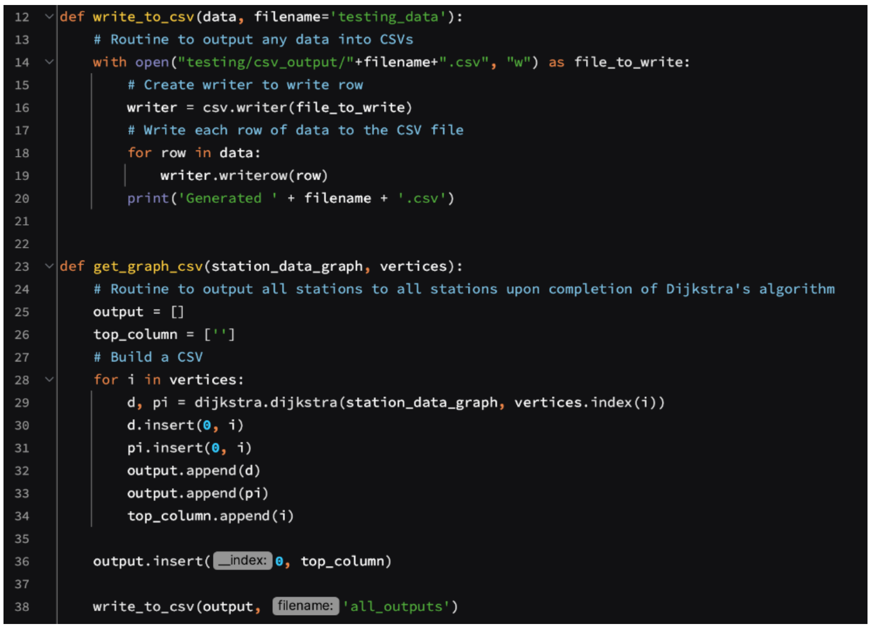
```
Select option: 1
Input starting station: Angel
Input destination station: Oxford Circus
Timing: 0.0084187984s
The shortest route for the given stations is: Angel -> Kings Cross St. Pancras -> Euston -> Warren Street -> Oxford Circus
This will take 8 minutes.
```



```
Select option: 1
Input starting station: a
Input destination station: c
Timing: 0.000505209s
The shortest route for the given stations is: A -> C
This will take 1 minutes.
```

The above screenshots show how we used a synthetic data set, produced to replicate the vertices and edges given in the data set, to test our solution to the tasks without having to import the given data set. This made our testing process a more effective and efficient process.

2. Deliberation on test data selection, accompanied by a table highlighting the tests executed to ascertain correctness and efficiency.

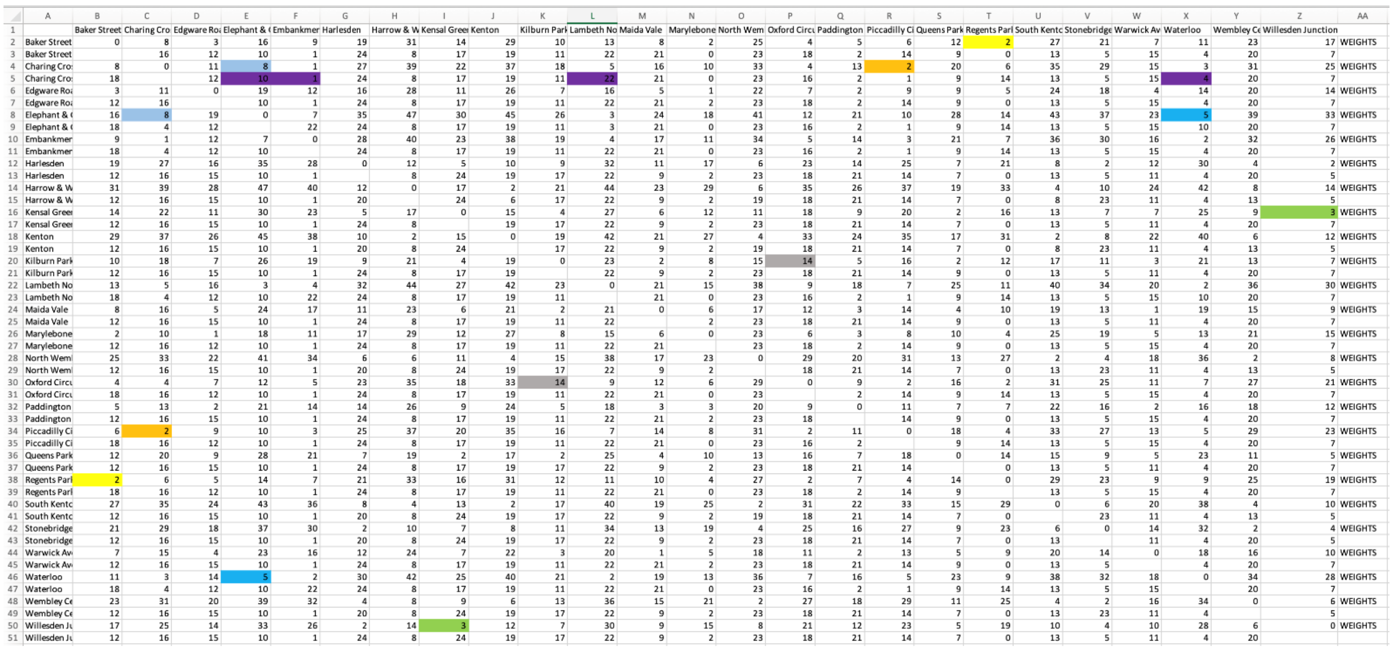


```python
def write_to_csv(data, filename='testing_data'):
    # Routine to output any data into CSVs
    with open("testing/csv_output/"+filename+".csv", "w") as file_to_write:
        # Create writer to write row
        writer = csv.writer(file_to_write)
        # Write each row of data to the CSV file
        for row in data:
            writer.writerow(row)
        print('Generated ' + filename + '.csv')


def get_graph_csv(station_data_graph, vertices):
    # Routine to output all stations to all stations upon completion of Dijkstra's algorithm
    output = []
    top_column = ['']
    # Build a CSV
    for i in vertices:
        d, pi = dijkstra.dijkstra(station_data_graph, vertices.index(i))
        d.insert(0, i)
        pi.insert(0, i)
        output.append(d)
        output.append(pi)
        top_column.append(i)

    output.insert(__index: 0, top_column)

    write_to_csv(output, filename: 'all_outputs')
```



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
1		Baker Street	Charing Cro	Edgware Roa	Elephant & (	Embankmer	Harlesden	Harrow & W	Kensal Greer	Kenton	Kilburn Park	Lambeth No	Maida Vale	Marylebone	North Wem	Oxford Circu	Paddington	Piccadilly Ci	Queens Park	Regents Parl	South Kentc	Stonebridge	Warwick Av	Waterloo	Wembley Ce	Willesden Junction	
2	Baker Street	0	8	3	16	9	19	31	14	29	10	13	8	2	25	4	5	6	12	2	27	21	7	11	23	17	WEIGHTS
3	Baker Street		16	12	10	1	24	8	17	19	11	22	21	0	23	18	2	14	9	0	13	5	15	4	20	7	
4	Charing Cro	8	0	11	8	1	27	39	22	37	18	5	16	10	33	4	13	2	20	6	35	29	15	3	31	25	WEIGHTS
5	Charing Cro	18		12	10	1	24	8	17	19	11	22	21	0	23	16	2	1	9	14	13	5	15	4	20	7	
6	Edgware Roa	3	11	0	19	12	16	28	11	26	7	16	5	1	22	7	2	9	9	5	24	18	4	14	20	14	WEIGHTS
7	Edgware Roa	12	16		10	1	24	8	17	19	11	22	21	2	23	18	2	14	9	0	13	5	15	4	20	7	
8	Elephant & (	16	8	19	0	7	35	47	30	45	26	3	24	18	41	12	21	10	28	14	43	37	23	5	39	33	WEIGHTS
9	Elephant & (	18	4	12		22	24	8	17	19	11	3	21	0	23	16	2	1	9	14	13	5	15	10	20	7	
10	Embankmer	9	1	12	7	0	28	40	23	38	19	4	17	11	34	5	14	3	21	7	36	30	16	2	32	26	WEIGHTS
11	Embankmer	18	4	12	10		24	8	17	19	11	22	21	0	23	16	2	1	9	14	13	5	15	4	20	7	
12	Harlesden	19	27	16	35	28	0	12	5	10	9	32	11	17	6	23	14	25	7	21	8	2	12	30	4	2	WEIGHTS
13	Harlesden	12	16	15	10	1		8	24	19	17	22	9	2	23	18	21	14	7	0	13	5	11	4	20	5	
14	Harrow & W	31	39	28	47	40	12	0	17	2	21	44	23	29	6	35	26	37	19	33	4	10	24	42	8	14	WEIGHTS
15	Harrow & W	12	16	15	10	1	20		24	6	17	22	9	2	19	18	21	14	7	0	8	23	11	4	13	5	
16	Kensal Greer	14	22	11	30	23	5	17	0	15	4	27	6	12	11	18	9	20	2	16	13	7	7	25	9	3	WEIGHTS
17	Kensal Greer	12	16	15	10	1	24	8		19	17	22	9	2	23	18	21	14	7	0	13	5	11	4	20	7	
18	Kenton	29	37	26	45	38	10	2	15	0	19	42	21	27	4	33	24	35	17	31	2	8	22	40	6	12	WEIGHTS
19	Kenton	12	16	15	10	1	20	8	24		17	22	9	2	19	18	21	14	7	0	8	23	11	4	13	5	
20	Kilburn Park	10	18	7	26	19	9	21	4	19	0	23	2	8	15	14	5	16	2	12	17	11	3	21	13	7	WEIGHTS
21	Kilburn Park	12	16	15	10	1	24	8	17	19		22	9	2	23	18	21	14	9	0	13	5	11	4	20	7	
22	Lambeth No	13	5	16	3	4	32	44	27	42	23	0	21	15	38	9	18	7	25	11	40	34	20	2	36	30	WEIGHTS
23	Lambeth No	18	4	12	10	22	24	8	17	19	11		21	0	23	16	2	1	9	14	13	5	15	10	20	7	
24	Maida Vale	8	16	5	24	17	11	23	6	21	2	21	0	6	17	12	3	14	4	10	19	13	1	19	15	9	WEIGHTS
25	Maida Vale	12	16	15	10	1	24	8	17	19	11	22		2	23	18	21	14	9	0	13	5	11	4	20	7	
26	Marylebone	2	10	1	18	11	17	29	12	27	8	15	6	0	23	6	3	8	10	4	25	19	5	13	21	15	WEIGHTS
27	Marylebone	12	16	12	10	1	24	8	17	19	11	22	21		23	18	2	14	9	0	13	5	15	4	20	7	
28	North Weml	25	33	22	41	34	6	6	11	4	15	38	17	23	0	29	20	31	13	27	2	4	18	36	2	8	WEIGHTS
29	North Weml	12	16	15	10	1	20	8	24	19	17	22	9	2		18	21	14	7	0	13	23	11	4	13	5	
30	Oxford Circu	4	4	7	12	5	23	35	18	33	14	9	12	6	29	0	9	2	16	2	31	25	11	7	27	21	WEIGHTS
31	Oxford Circu	18	16	12	10	1	24	8	17	19	11	22	21	0	23		2	14	9	14	13	5	15	4	20	7	
32	Paddington	5	13	2	21	14	14	26	9	24	5	18	3	3	20	9	0	11	7	7	22	16	2	16	18	12	WEIGHTS
33	Paddington	12	16	15	10	1	24	8	17	19	11	22	21	2	23	18		14	9	0	13	5	15	4	20	7	
34	Piccadilly Ci	6	2	9	10	3	25	37	20	35	16	7	14	8	31	2	11	0	18	4	33	27	13	5	29	23	WEIGHTS
35	Piccadilly Ci	18	16	12	10	1	24	8	17	19	11	22	21	0	23	16	2		9	14	13	5	15	4	20	7	
36	Queens Park	12	20	9	28	21	7	19	2	17	2	25	4	10	13	16	7	18	0	14	15	9	5	23	11	5	WEIGHTS
37	Queens Park	12	16	15	10	1	24	8	17	19	17	22	9	2	23	18	21	14		0	13	5	11	4	20	7	
38	Regents Parl	2	6	5	14	7	21	33	16	31	12	11	10	4	27	2	7	4	14	0	29	23	9	9	25	19	WEIGHTS
39	Regents Parl	18	16	12	10	1	24	8	17	19	11	22	21	0	23	18	2	14	9		13	5	15	4	20	7	
40	South Kentc	27	35	24	43	36	8	4	13	2	17	40	19	25	2	31	22	33	15	29	0	6	20	38	4	10	WEIGHTS
41	South Kentc	12	16	15	10	1	20	8	24	19	17	22	9	2	19	18	21	14	7	0		23	11	4	13	5	
42	Stonebridge	21	29	18	37	30	2	10	7	8	11	34	13	19	4	25	16	27	9	23	6	0	14	32	2	4	WEIGHTS
43	Stonebridge	12	16	15	10	1	20	8	24	19	17	22	9	2	23	18	21	14	7	0	13		11	4	20	5	
44	Warwick Av	7	15	4	23	16	12	24	7	22	3	20	1	5	18	11	2	13	5	9	20	14	0	18	16	10	WEIGHTS
45	Warwick Av	12	16	15	10	1	24	8	17	19	11	22	21	2	23	18	21	14	9	0	13	5		4	20	7	
46	Waterloo	11	3	14	5	2	30	42	25	40	21	2	19	13	36	7	16	5	23	9	38	32	18	0	34	28	WEIGHTS
47	Waterloo	18	4	12	10	22	24	8	17	19	11	22	21	0	23	16	2	1	9	14	13	5	15		20	7	
48	Wembley Ce	23	31	20	39	32	4	8	9	6	13	36	15	21	2	27	18	29	11	25	4	2	16	34	0	6	WEIGHTS
49	Wembley Ce	12	16	15	10	1	20	8	24	19	17	22	9	2	23	18	21	14	7	0	13	23	11	4		5	
50	Willesden Ju	17	25	14	33	26	2	14	3	12	7	30	9	15	8	21	12	23	5	19	10	4	10	28	6	0	WEIGHTS
51	Willesden Ju	12	16	15	10	1	24	8	24	19	17	22	9	2	23	18	21	14	7	0	13	5	11	4	20		

This csv file outputs both the d value (time taken) and pi value (previous station) for each station pair. The coloured cells shows that the values are the equal for the same station pairs and therefore validating the data. There are blank cells because there are no previous stations if the journey consists of the same station, this also explains why the journey is 0 minutes.

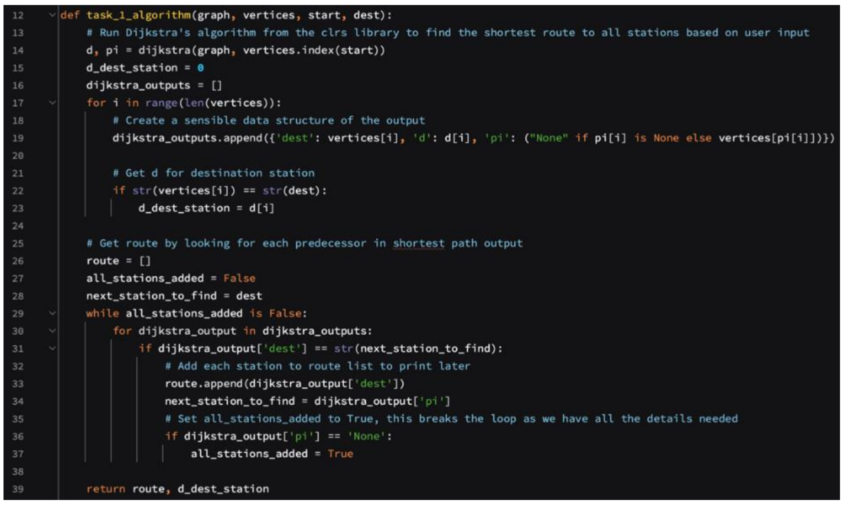
Upon examining the coded section for task 1, we can denote that most lines consist of a complexity of O(1) and O(n) therefore making it linear, until line 51 where the function that runs the Dijkstra algorithm is called. This has a time complexity of O((V+E)*log(V)), where V is the number of vertices and E is the number of edges in the graph. It is the most dominant complexity, therefore the coded solution for task 1 has an overall time complexity of O((V+E)*log(V)).

1	1	Normal	1	Check if route/time is returned correctly	Piccadilly Circus -> Charing Cross	Piccadilly Circus -> Charing Cross | This will take 2 minutes.	P	[Bakerloo + Central lines only] We manually check against the tube map to see if the routes are correct here - As with all the tests
1	2	Erroneous	1	Check if appropriate error messages appear	Picadilly Circus -> Charing Cross	The start station does not exist or is misspelled. Please check your inputs.	P	[Bakerloo + Central lines only] Spelling error

1	3	Normal	1	Check if route/time is returned correctly	Elephant & Castle -> Charing Cross	Elephant & Castle -> Kennington -> Waterloo -> Embankment -> Charing Cross | This will take 8 minutes.	P	[Bakerloo + Central lines only]
1	4	Erroneous	1	Check if appropriate error messages appear	Charing Cross -> Charing Cross	The start and destination stations cannot be the same. Please check your inputs.	P	[Bakerloo + Central lines only] Repeated starting and destination station
1	5	Normal	1	Check if route/time is returned correctly	Marylebone -> Holborn	Marylebone -> Baker Street -> Regents Park -> Oxford Circus -> Tottenham -> Holborn | This will take 9 minutes.	P	[Bakerloo + Central lines only]
1	6	Erroneous	1	Check if appropriate error messages appear	Marylebone -> Angel	The destination station does not exist or is misspelled. Please check your inputs.	P	[Bakerloo + Central lines only] Angel is on the Northern line

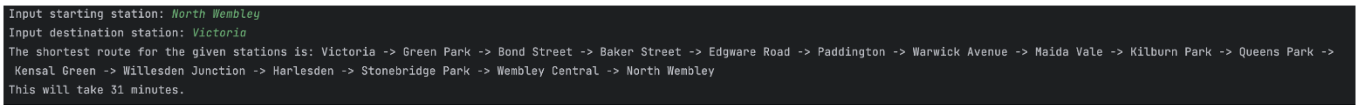
2	1	Normal	1	Check if route/time is returned correctly	Russell Square -> Farringdon	Russell Square -> Kings Cross St. Pancras -> Farringdon | This will take 6 minutes.	P	[All lines]
2	2	Erroneous	1	Check if appropriate error messages appear	Russell Square -> Faringdon	The start and destination stations do not exist or are misspelled. Please check your inputs.	P	[All lines] Spelling Error
2	3	Normal	1	Check if route/time is returned correctly	Angel -> Bayswater	Angel -> Kings Cross St. Pancras -> Euston Square -> Great Portland Street -> Baker Street -> Edgware Road -> Paddington -> Bayswater	P	[All lines] Passes multiple lines
2	4	Erroneous	1	Check if appropriate error messages appear	Angel -> Angel	The start and destination stations cannot be the same. Please check your inputs.	P	[All lines] Repeating starting and destination station
2	5	Normal	1	Check if route/time is returned correctly	Waterloo -> Bank	Waterloo -> Bank | This will take 5 minutes.	P	[All lines]
2	6	Erroneous	1	Check if appropriate error messages appear	Bank -> Waterloo	Bank -> Waterloo | This will take 5 minutes.	P	[All lines]

3. Results derived from the tasks, including screen-captured presentations showcasing the functionality of your application code and compliance with the use of the required library code.

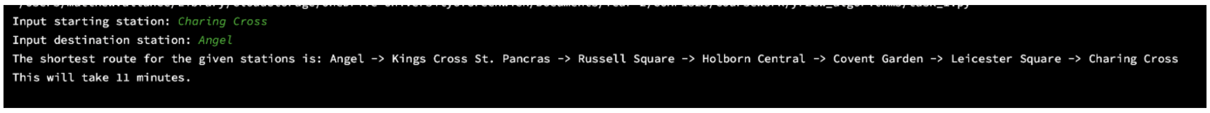


```python
12  def task_1_algorithm(graph, vertices, start, dest):
13      # Run Dijkstra's algorithm from the clrs library to find the shortest route to all stations based on user input
14      d, pi = dijkstra(graph, vertices.index(start))
15      d_dest_station = 0
16      dijkstra_outputs = []
17      for i in range(len(vertices)):
18          # Create a sensible data structure of the output
19          dijkstra_outputs.append({'dest': vertices[i], 'd': d[i], 'pi': ("None" if pi[i] is None else vertices[pi[i]])})
20
21          # Get d for destination station
22          if str(vertices[i]) == str(dest):
23              d_dest_station = d[i]
24
25      # Get route by looking for each predecessor in shortest path output
26      route = []
27      all_stations_added = False
28      next_station_to_find = dest
29      while all_stations_added is False:
30          for dijkstra_output in dijkstra_outputs:
31              if dijkstra_output['dest'] == str(next_station_to_find):
32                  # Add each station to route list to print later
33                  route.append(dijkstra_output['dest'])
34                  next_station_to_find = dijkstra_output['pi']
35                  # Set all_stations_added to True, this breaks the loop as we have all the details needed
36                  if dijkstra_output['pi'] == 'None':
37                      all_stations_added = True
38
39      return route, d_dest_station
```

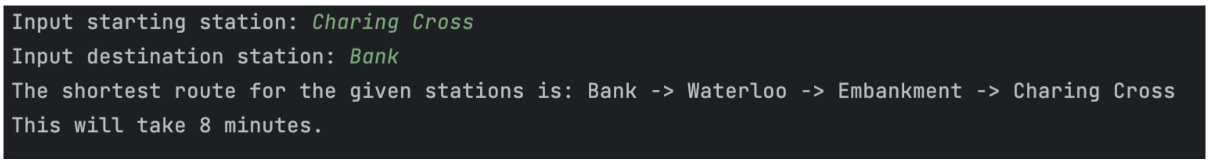
In this task we required the use of Dijkstra's algorithm which is available in the library given to us. It can be seen on line 14 which gathers the values of d and pi.



```
Input starting station: North Wembley
Input destination station: Victoria
The shortest route for the given stations is: Victoria -> Green Park -> Bond Street -> Baker Street -> Edgware Road -> Paddington -> Warwick Avenue -> Maida Vale -> Kilburn Park -> Queens Park ->
 Kensal Green -> Willesden Junction -> Harlesden -> Stonebridge Park -> Wembley Central -> North Wembley
This will take 31 minutes.
```

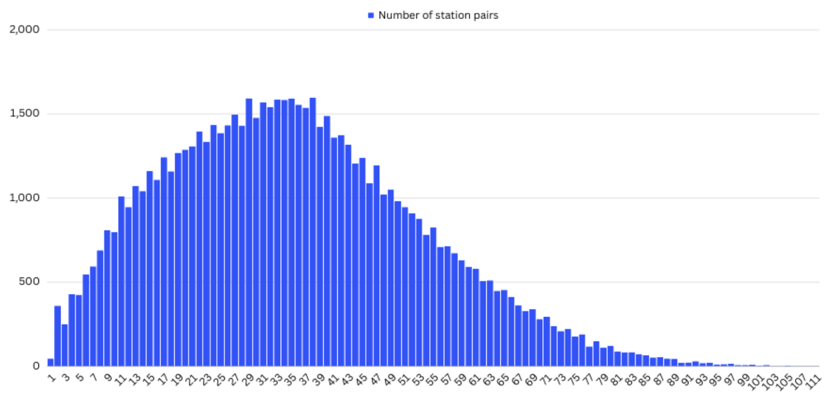


```
Input starting station: Charing Cross
Input destination station: Angel
The shortest route for the given stations is: Angel -> Kings Cross St. Pancras -> Russell Square -> Holborn Central -> Covent Garden -> Leicester Square -> Charing Cross
This will take 11 minutes.
```



```
Input starting station: Charing Cross
Input destination station: Bank
The shortest route for the given stations is: Bank -> Waterloo -> Embankment -> Charing Cross
This will take 8 minutes.
```

In these examples, we have given three different lengths of journeys which includes changing lines. We have included the journey path that makes it possible to check whether the code is running correctly.



Beginning from 1-minute journeys between two stations this histogram shows the time taken between any two station pairs in total minutes. The most common journey between any two stations is 29 minutes at 1593 station pairs. The histogram is positively skewed which means most of the journeys are likely to be smaller than the midpoint. There are only a small portion of station pairs that take over 100 minutes to complete; these are likely on opposite sides of London. The calculation for the total journeys considers station pairs right next to each other that may take 3 minutes for example but also considers stations that are not directly next to each other and may have to change to a different line to reach its destination. This is how there are journeys that take up to 111 minutes despite not having a station pair next to each other with this length of journey.

4. Final remarks, along with an insightful discussion addressing the limitations of the undertaken work.

Overall, whilst using the Dijkstra Algorithm we were able to create a solution that suited our needs and fulfilled any tasks we needed it to for us to be successful. However, when reviewing our code and its limitations we have realised that even though it fits our needs now, that may not always be the case since the Dijkstra's Algorithm only works with non-negative edges. Whereas alternative algorithms like the Bellman-Ford algorithm can work with negative edges. The Dijkstra algorithm has been designed to find the shortest path from a single source to all the other vertices in the graph, resulting in the fact that it can be considered a limitation to use the Dijkstra algorithm to find the shortest path between two specific stations, especially when there are other algorithms such as the A* search algorithm, which is commonly used for pathfinding, that would be more efficient. It might be quite difficult to adapt the algorithm dynamically should new stations need to be added to the London Underground, or should current stations need to be amended. This restricts the future use of the Dijkstra algorithm because there are other algorithms which can adapt to dynamic changes more effectively.

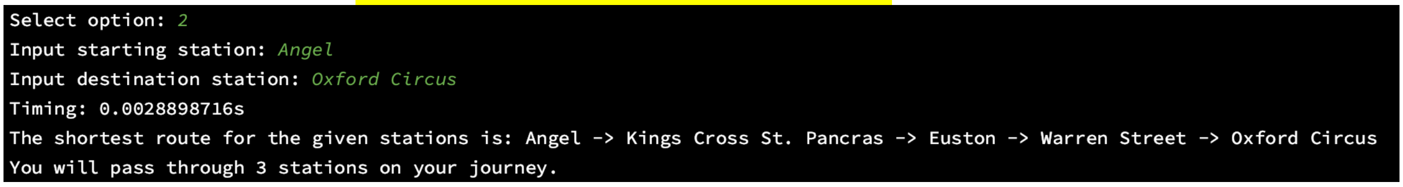
TASK 2

1. Rationale for selecting specific algorithms and data structures, complemented by empirical performance analysis using synthetic data of varying sizes.

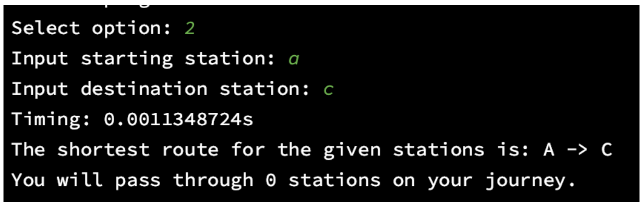
Task 2 requires the use of Dijkstra's algorithm as it performs similar calculations to task 1. As described in task 1 point 1, this algorithm calculates the shortest path through each of the individual nodes however, this task requires us to find the shortest path in terms of the number of stations. We decided to keep the same data structure, of the adjacency list graph, due to it presenting the most efficient structure for our sparse graph. Despite the similarities with the previous task, this task contrasts in terms of its output as it requires the length of the journey in stations as opposed to in minutes. When comparing the two journey times produced, you can sometimes see that although a journey may be shorter in its number of stations it will take a longer duration in minutes. Therefore, creating a difference in the routes found by running the algorithms produced.

TASK	ATTEMPT	DATA SET (TUBE MAP)	DATA SET (TEST)
2	1	0.002889872	0.001134872
	2	0.002876759	0.000387192

	3	0.003042221	0.000505209
AVERAGE		0.002936284	0.000675758



```
Select option: 2
Input starting station: Angel
Input destination station: Oxford Circus
Timing: 0.0028898716s
The shortest route for the given stations is: Angel -> Kings Cross St. Pancras -> Euston -> Warren Street -> Oxford Circus
You will pass through 3 stations on your journey.
```



```
Select option: 2
Input starting station: a
Input destination station: c
Timing: 0.0011348724s
The shortest route for the given stations is: A -> C
You will pass through 0 stations on your journey.
```

The above screenshots show how we used a synthetic data set, produced to replicate the vertices and edges given in the data set, to test our solution to the tasks without having to import the given data set. This made our testing process a more effective and efficient process.

2. Deliberation on test data selection, accompanied by a table highlighting the tests executed to ascertain correctness and efficiency.

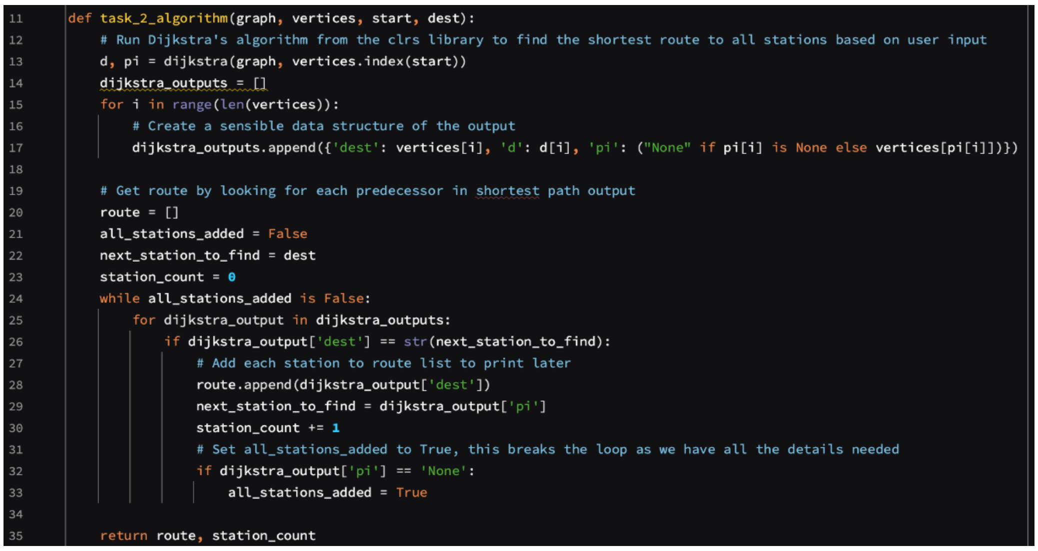
1	7	Normal	2	Check if route/time is returned correctly	Piccadilly Circus -> Charing Cross	Piccadilly Circus -> Charing Cross | You will pass through 0 stations on your journey.	P	[Bakerloo + Central lines only]
1	8	Erroneous	2	Check if appropriate error messages appear	Picadilly Circus -> Charing Cross	The start station does not exist or is misspelled. Please check your inputs.	P	[Bakerloo + Central lines only] Repeated stations for task 2 - 4
1	9	Normal	2	Check if route/time is returned correctly	Elephant & Castle -> Charing Cross	Elephant & Castle -> Kennington -> Waterloo -> Embankment -> Charing Cross | You will pass through 3 stations on your journey.	P	[Bakerloo + Central lines only]
1	10	Erroneous	2	Check if appropriate error messages appear	Charing Cross -> Charing Cross	The start and destination stations cannot be the same. Please check your inputs.	P	[Bakerloo + Central lines only]
1	11	Normal	2	Check if route/time is returned correctly	Marylebone -> Holborn	Marylebone -> Baker Street -> Regents Park -> Oxford Circus -> Tottenham -> Holborn | You will pass through 4 stations on your journey.	P	[Bakerloo + Central lines only]
1	12	Erroneous	2	Check if appropriate error messages appear	Marylebone -> Angel	The start station does not exist or is misspelled. Please check your inputs.	P	[Bakerloo + Central lines only]

2	7	Normal	2	Check if route/time is returned correctly	Russell Square -> Farringdon	Russell Square -> Kings Cross St. Pancras -> Farringdon | You will pass through 1 stations on your journey.	P	[All lines]
2	8	Erroneous	2	Check if appropriate error messages appear	Russell Square -> Faringdon	The start station does not exist or is misspelled. Please check your inputs.	P	[All lines]
2	9	Normal	2	Check if route/time is returned correctly	Angel -> Bayswater	Angel -> Kings Cross St. Pancras -> Euston Square -> Great Portland Street -> Baker Street -> Edgware Road -> Paddington -> Bayswater | You will pass through 6 stations on your journey.	P	[All lines]
2	10	Erroneous	2	Check if appropriate error messages appear	Angel -> Angel	The start and destination stations cannot be the same. Please check your inputs.	P	[All lines]
2	11	Normal	2	Check if route/time is returned correctly	Waterloo -> Bank	Waterloo -> Bank | You will pass through 0 stations on your journey.	P	[All lines]
2	12	Erroneous	2	Check if appropriate error messages appear	Bank -> Waterloo	Bank -> Waterloo | You will pass through 0 stations on your journey.	P	[All lines]

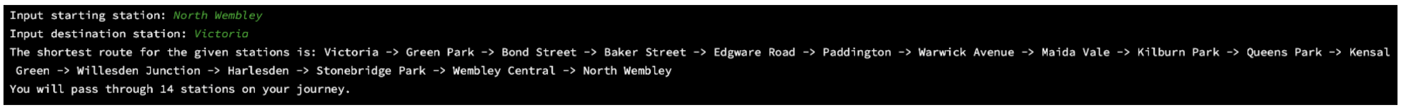
In comparison to task 1, task 2 shows many similarities with regards to the time complexities throughout the code. We can infer that for the duration of the code, the complexity is linear. However, when we reach line 23, the complexity resorts back to the O((V+E)*log(V)), mapping Dijkstra's algorithm, once again being the superior complexity, leaving task 2 with a time complexity of O((V+E)*log(V)).

3. Results derived from the tasks, including screen-captured presentations showcasing the functionality of your application code and compliance with the use of the required library code.

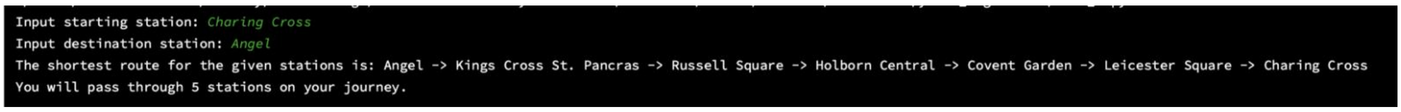
As shown below, on line 13 we have called the Dijkstra's algorithm from the library code to get the shortest path in station counts which is then used further down to complete calculations and output to the user.



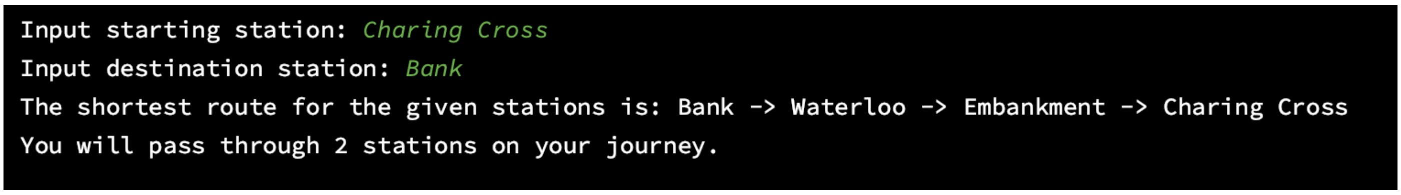
```python
11  def task_2_algorithm(graph, vertices, start, dest):
12      # Run Dijkstra's algorithm from the clrs library to find the shortest route to all stations based on user input
13      d, pi = dijkstra(graph, vertices.index(start))
14      dijkstra_outputs = []
15      for i in range(len(vertices)):
16          # Create a sensible data structure of the output
17          dijkstra_outputs.append({'dest': vertices[i], 'd': d[i], 'pi': ("None" if pi[i] is None else vertices[pi[i]])})
18
19      # Get route by looking for each predecessor in shortest path output
20      route = []
21      all_stations_added = False
22      next_station_to_find = dest
23      station_count = 0
24      while all_stations_added is False:
25          for dijkstra_output in dijkstra_outputs:
26              if dijkstra_output['dest'] == str(next_station_to_find):
27                  # Add each station to route list to print later
28                  route.append(dijkstra_output['dest'])
29                  next_station_to_find = dijkstra_output['pi']
30                  station_count += 1
31                  # Set all_stations_added to True, this breaks the loop as we have all the details needed
32                  if dijkstra_output['pi'] == 'None':
33                      all_stations_added = True
34
35      return route, station_count
```



```
Input starting station: North Wembley
Input destination station: Victoria
The shortest route for the given stations is: Victoria -> Green Park -> Bond Street -> Baker Street -> Edgware Road -> Paddington -> Warwick Avenue -> Maida Vale -> Kilburn Park -> Queens Park -> Kensal
 Green -> Willesden Junction -> Harlesden -> Stonebridge Park -> Wembley Central -> North Wembley
You will pass through 14 stations on your journey.
```

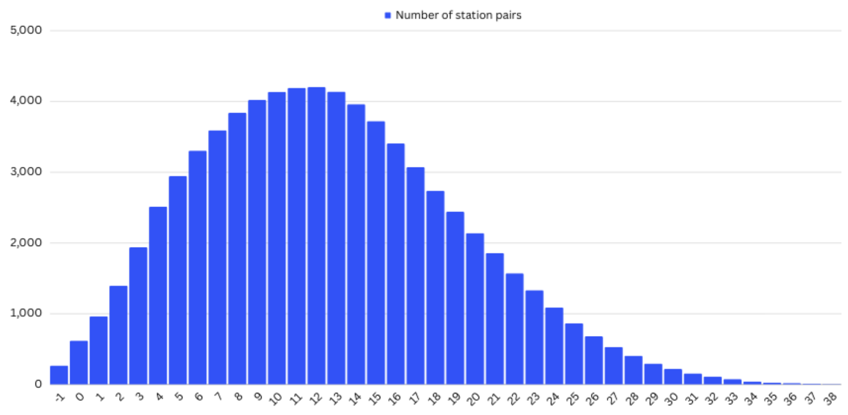


```
Input starting station: Charing Cross
Input destination station: Angel
The shortest route for the given stations is: Angel -> Kings Cross St. Pancras -> Russell Square -> Holborn Central -> Covent Garden -> Leicester Square -> Charing Cross
You will pass through 5 stations on your journey.
```



```
Input starting station: Charing Cross
Input destination station: Bank
The shortest route for the given stations is: Bank -> Waterloo -> Embankment -> Charing Cross
You will pass through 2 stations on your journey.
```

We have used the same examples as task 1 to show that the program outputs the correct number of stations that are within the journey. We have included the stations within the journey again to physically check the output of the program is correct.



This histogram shows the correlations between the number of stations between the start and destination pair and the number of station pairs. The highest number of stations a person would have to travel is 12 stations at 4202 station pairs. Similarly, to the previous histogram, this has a positive skew which means that many of the station pairs have less than the middle of the stations between the pairs on the graph. I have included the -1 value as this means the station pairs are the same which is true for all 270 stations. This helps visualise the total number of stations and the number of comparison pairs this algorithm is undertaking. This histogram also uses a 0 value between two stations which means there are no stations between the station pairs as they are directly next to each other.

4. Final remarks, along with an insightful discussion addressing the limitations of the undertaken work.

Upon the conclusion of this task, we found that the repeated use of the Dijkstra Algorithm was pivotal to the comparison between the two paths generated from this task and task 1. It acted a control variable in the sense of it taking the same steps to find the shortest path but with a different measurement used each time. However, as it uses the Dijkstra Algorithm it not only inherits all the efficiency of the above task but also, it's limitations such as; being unable to process negatively weighted edges, having to find all possible paths before selecting the shortest one, and it being a difficult algorithm to adapt dynamically. These limitations can result in other algorithms, Bellman-Ford, A* search, being an alternative selection when focusing on eliminating these limitations.

TASK 3

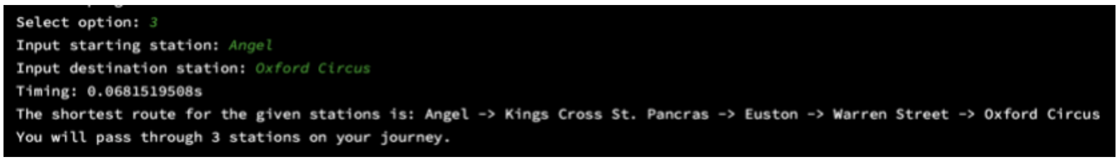
1. Rationale for selecting specific algorithms and data structures, complemented by empirical performance analysis using synthetic data of varying sizes.

The objective of task 3 was to replicate the number of stations or stops between the starting point and the destination, as completed in task 2 but with a different algorithm. We chose to use the adjacency list graph with the Bellman-Ford method. Unlike the Dijkstra method, the Bellman-Ford algorithm is more flexible since it can handle negative edge weights.

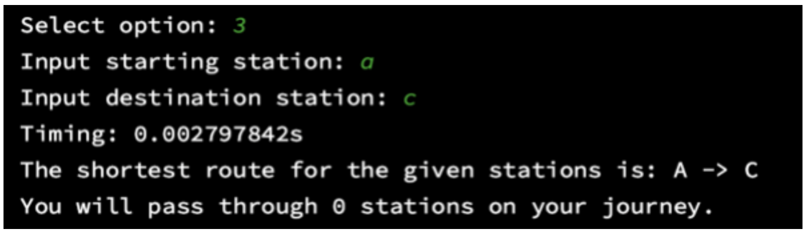
The Bellman-Ford Algorithm is also used within GPS applications in real life which further solidified our confidence when selecting this algorithm as a contrasting method. This algorithm works efficiently with the adjacency list, especially for a sparse graph like the London Underground system, because it lets us store the relevant information surrounding the connections consequentially, allowing the algorithm to run at a better efficiency.

We decided to keep using the adjacency list graph as our data structure because only the algorithm had to change, and this data structure had a direct access to close stations making it more suitable for analysing and navigating the underground system when compared to any alternatives, such as the adjacency matrix.

TASK	ATTEMPT	DATA SET (TUBE MAP)	DATA SET (TEST)
3	1	0.068151951	0.002797842
	2	0.06229806	0.002662897
	3	0.063828945	0.002643108
	AVERAGE	0.064759652	0.002701283



```
Select option: 3
Input starting station: Angel
Input destination station: Oxford Circus
Timing: 0.06815195088s
The shortest route for the given stations is: Angel -> Kings Cross St. Pancras -> Euston -> Warren Street -> Oxford Circus
You will pass through 3 stations on your journey.
```



```
Select option: 3
Input starting station: a
Input destination station: c
Timing: 0.002797842s
The shortest route for the given stations is: A -> C
You will pass through 0 stations on your journey.
```

The above screenshots show how we used a synthetic data set, produced to replicate the vertices and edges given in the data set, to test our solution to the tasks without having to import the given data set. This made our testing process a more effective and efficient process.

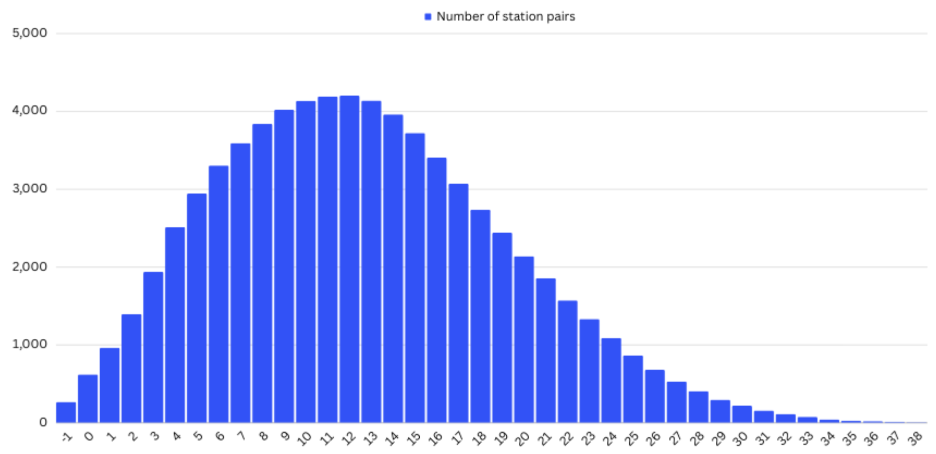
2. Deliberation on test data selection, accompanied by a table highlighting the tests executed to ascertain correctness and efficiency.

1	13	Normal	3	Check if route/time is returned correctly	Piccadilly Circus -> Charing Cross	Piccadilly Circus -> Charing Cross | You will pass through 0 stations	P	[Bakerloo + Central lines only]
1	14	Erroneous	3	Check if appropriate error messages appear	Picadilly Circus -> Charing Cross	The start station does not exist or is misspelled. Please check your inputs.	P	[Bakerloo + Central lines only]
1	15	Normal	3	Check if route/time is returned correctly	Elephant & Castle -> Charing Cross	Elephant & Castle -> Kennington -> Waterloo -> Embankment -> Charing Cross | You will pass through 3 stations	P	[Bakerloo + Central lines only]
1	16	Erroneous	3	Check if appropriate error messages appear	Charing Cross -> Charing Cross	The start and destination stations cannot be the same. Please check your inputs.	P	[Bakerloo + Central lines only]
1	17	Normal	3	Check if route/time is returned correctly	Marylebone -> Holborn	Marylebone -> Baker Street -> Bond Street -> Green Park -> Piccadilly Circus -> Charing Cross | You will pass through 4 stations on your journey.	P	[Bakerloo + Central lines only]
1	18	Erroneous	3	Check if appropriate error messages appear	Marylebone -> Angel	The destination station does not exist or is misspelled. Please check your inputs.	P	[Bakerloo + Central lines only]

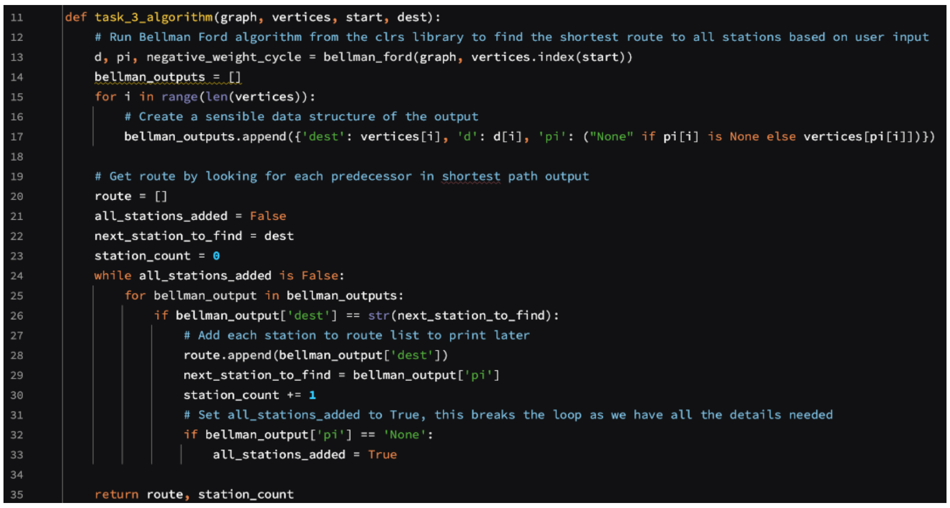
2	13	Normal	3	Check if route/time is returned correctly	Russell Square -> Farringdon	Russell Square -> Kings Cross St. Pancras -> Farringdon | You will pass through 1 stations on your journey.	P	[All lines]
2	14	Erroneous	3	Check if appropriate error messages appear	Russell Square -> Faringdon	The destination station does not exist or is misspelled. Please check your inputs.	P	[All lines]
2	15	Normal	3	Check if route/time is returned correctly	Angel -> Bayswater	Angel -> Kings Cross St. Pancras -> Euston Square -> Great Portland Street -> Baker Street -> Edgware Road -> Paddington -> Bayswater | You will pass through 6 stations on your journey.	P	[All lines]
2	16	Erroneous	3	Check if appropriate error messages appear	Angel -> Angel	The start and destination stations cannot be the same. Please check your inputs.	P	[All lines]
2	17	Normal	3	Check if route/time is returned correctly	Waterloo -> Bank	Waterloo -> Bank | You will pass through 0 stations on your journey.	P	[All lines]
2	18	Erroneous	3	Check if appropriate error messages appear	Bank -> Waterloo	Bank -> Waterloo | You will pass through 0 stations on your journey.	P	[All lines]

According to our reference (Geeks, 2023), the time complexity of the Bellman-Ford algorithm is O(V*E). The point at which we can observe this is line 18 up until which point the code is constant and linear. As a result, it's found to be less efficient than the other algorithms such as the Dijkstra algorithm when comparing the time complexities of both. In addition to this, the Bellman-Ford algorithm may not end when expected and will continue to iterate despite finding the closest station. This is because the algorithm has been designed to perform a fixed number of iterations and will continue to iterate until this number has been reached resulting a less time efficient solution.

3. Results derived from the tasks, including screen-captured presentations showcasing the functionality of your application code and compliance with the use of the required library code.

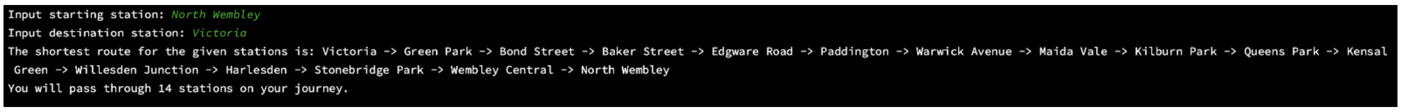


This histogram displays the same data as task 2 due to the results being identical. This allows us to further demonstrate how even though the method we used is different, the outcome remains the same.

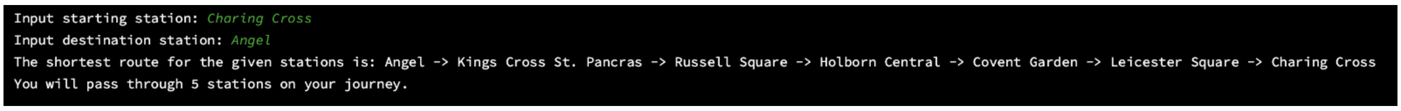


```
11  def task_3_algorithm(graph, vertices, start, dest):
12      # Run Bellman Ford algorithm from the clrs library to find the shortest route to all stations based on user input
13      d, pi, negative_weight_cycle = bellman_ford(graph, vertices.index(start))
14      bellman_outputs = []
15      for i in range(len(vertices)):
16          # Create a sensible data structure of the output
17          bellman_outputs.append({'dest': vertices[i], 'd': d[i], 'pi': ("None" if pi[i] is None else vertices[pi[i]])})
18
19      # Get route by looking for each predecessor in shortest path output
20      route = []
21      all_stations_added = False
22      next_station_to_find = dest
23      station_count = 0
24      while all_stations_added is False:
25          for bellman_output in bellman_outputs:
26              if bellman_output['dest'] == str(next_station_to_find):
27                  # Add each station to route list to print later
28                  route.append(bellman_output['dest'])
29                  next_station_to_find = bellman_output['pi']
30                  station_count += 1
31                  # Set all_stations_added to True, this breaks the loop as we have all the details needed
32                  if bellman_output['pi'] == 'None':
33                      all_stations_added = True
34
35      return route, station_count
```

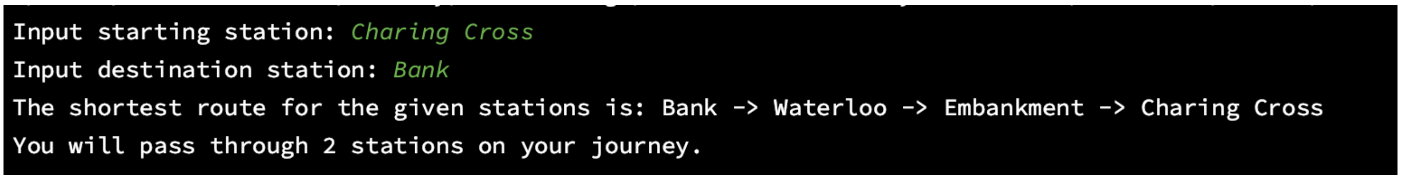
This task requires the Bellman-Ford algorithm which can be found in the library code. We have called this function on line 13 to produce d, pi and the negative weight cycle, needed to run the program. After looking through our data set it was clear that we did not need to produce a negative weight cycle however, it was a specified requirement for us to be able to run the code.



```
Input starting station: North Wembley
Input destination station: Victoria
The shortest route for the given stations is: Victoria -> Green Park -> Bond Street -> Baker Street -> Edgware Road -> Paddington -> Warwick Avenue -> Maida Vale -> Kilburn Park -> Queens Park -> Kensal
 Green -> Willesden Junction -> Harlesden -> Stonebridge Park -> Wembley Central -> North Wembley
You will pass through 14 stations on your journey.
```



```
Input starting station: Charing Cross
Input destination station: Angel
The shortest route for the given stations is: Angel -> Kings Cross St. Pancras -> Russell Square -> Holborn Central -> Covent Garden -> Leicester Square -> Charing Cross
You will pass through 5 stations on your journey.
```



```
Input starting station: Charing Cross
Input destination station: Bank
The shortest route for the given stations is: Bank -> Waterloo -> Embankment -> Charing Cross
You will pass through 2 stations on your journey.
```

These examples are the same as task 2 because even though the algorithm is different for this task, when performed correctly, it should yield the same results.

4. Final remarks, along with an insightful discussion addressing the limitations of the undertaken work.

Throughout our process of using the Bellman-Ford Algorithm we were able to create a solution that was well-suited towards the task leading it to be successful when tested with both the synthetic and given data set. However, when reviewing the code and its limitations we made the realisation that just because it fits our needs now, it may not always be that way, especially when it comes to being time efficient. Moreover, as clarified earlier, the Bellman-Ford Algorithm has the time complexity of O(V*E) which makes it less efficient than the Dijkstra Algorithm used throughout tasks 1 and 2. However, when using this algorithm, it opens the solution to the idea of successfully using negatively weighted edges so depending on what your solution focuses on will alter what algorithm best suits your needs. A personal improvement for our solution would be to find a way I which we don't collect the data for the negative weight cycle, as we don't have any negative weights, and use the space allocated for that data in a more productive way by storing other information.

TASK 4

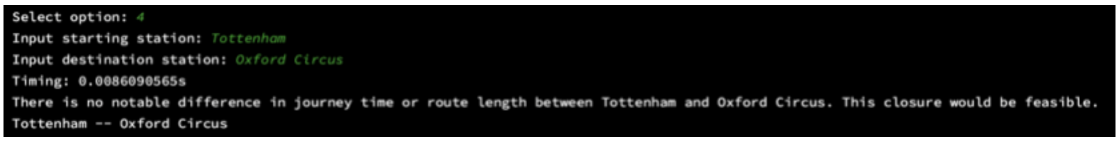
1. Rationale for selecting specific algorithms and data structures, complemented by empirical performance analysis using synthetic data of varying sizes.

For this task we used a combination of the Kruskal and Dijkstra algorithms. The objective of this task was to address the government's plan to shut down tube lines between adjacent stations while ensuring that travel between any two stations remains viable. After conducting research for this task, we concluded that a combination of Kruskal's algorithm for Minimum Spanning Trees (MST) and Dijkstra's algorithm for finding the shortest journey would be the best approach.

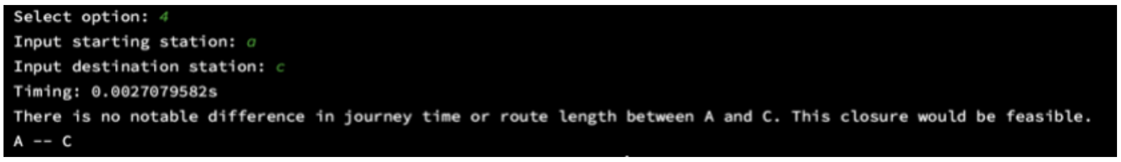
The MST is the tube lines that should be kept operational so that all stations are still connected, it should include all stations with the minimum possible total connections. The Kruskal's algorithm can be used to find the MST of the London Underground network. The Kruskal algorithm works by adding the lightest edges (the most viable stations) to the MST whilst avoiding unnecessary connections. The edges of the MST represent the lines that should be kept operational as to not disrupt the already function London Underground network. After the MST was generated, we decided that the Dijkstra algorithm can be used to find the shortest path between all pairs of stations.

The adjacency list graph remains to be the best data structure for this task and to use with both algorithms in order to yield the most efficient and effective results.

TASK	ATTEMPT	DATA SET (TUBE MAP)	DATA SET (TEST)
4	1	0.008609057	0.002707958
	2	0.015672207	0.003665924
	3	0.008831978	0.001383781
	AVERAGE	0.011037747	0.002585888



```
Select option: 4
Input starting station: Tottenham
Input destination station: Oxford Circus
Timing: 0.0086090565s
There is no notable difference in journey time or route length between Tottenham and Oxford Circus. This closure would be feasible.
Tottenham -- Oxford Circus
```



```
Select option: 4
Input starting station: a
Input destination station: c
Timing: 0.0027079582s
There is no notable difference in journey time or route length between A and C. This closure would be feasible.
A -- C
```

The above screenshots show how we used a synthetic data set, produced to replicate the vertices and edges given in the data set, to test our solution to the tasks without having to import the given data set. This made our testing process a more effective and efficient process.

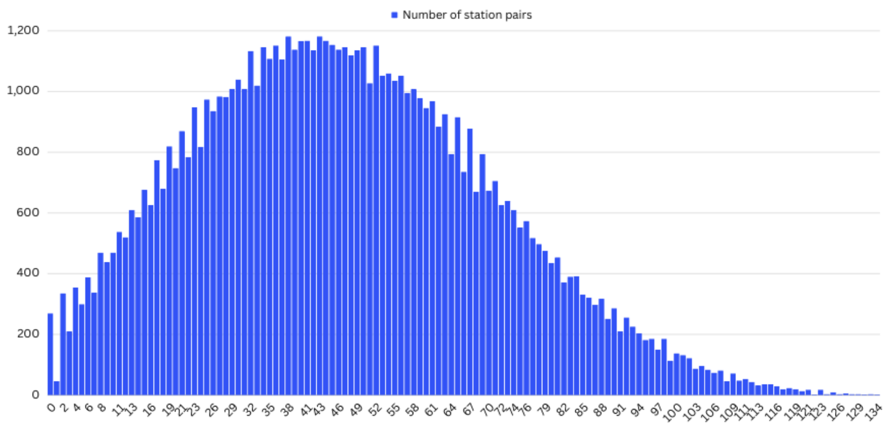
2. Deliberation on test data selection, accompanied by a table highlighting the tests executed to ascertain correctness and efficiency.

1	19	Normal	4	Check if route/time is returned correctly	Piccadilly Circus -> Charing Cross	There is no notable difference in journey time or route length between Piccadilly Circus and Charing Cross. This closure would be feasible.	P	[Bakerloo + Central lines only]
1	20	Erroneous	4	Check if appropriate error messages appear	Picadilly Circus -> Charing Cross	The start station does not exist or is misspelled. Please check your inputs.	P	[Bakerloo + Central lines only]
1	21	Normal	4	Check if route/time is returned correctly	Elephant & Castle -> Charing Cross	There is no notable difference in journey time or route length between Elephant & Castle and Charing Cross. This closure would be feasible.	P	[Bakerloo + Central lines only]
1	22	Erroneous	4	Check if appropriate error messages appear	Charing Cross -> Charing Cross	The start and destination stations cannot be the same. Please check your inputs.	P	[Bakerloo + Central lines only]
1	23	Normal	4	Check if route/time is returned correctly	Marylebone -> Holborn	There is no notable difference in journey time or route length between Marylebone and Holborn. This closure would be feasible.	P	[Bakerloo + Central lines only]
1	24	Erroneous	4	Check if appropriate error messages appear	Marylebone -> Angel	The destination station does not exist or is misspelled. Please check your inputs.	P	[Bakerloo + Central lines only]

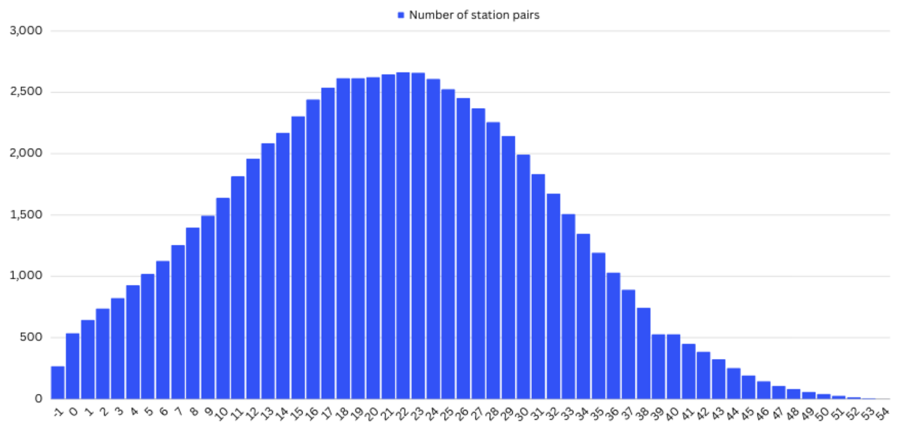
2	19	Normal	4	Check if route/time is returned correctly	Russell Square -> Farringdon	The stations you entered are not adjacent. Try again.	P	[All lines]
2	20	Erroneous	4	Check if appropriate error messages appear	Russell Square -> Faringdon	The start and destination stations do not exist or are misspelled. Please check your inputs.	P	[All lines]
2	21	Normal	4	Check if route/time is returned correctly	Angel -> Bayswater	The stations you entered are not adjacent. Try again.	P	[All lines]
2	22	Erroneous	4	Check if appropriate error messages appear	Angel -> Angel	The start and destination stations cannot be the same. Please check your inputs.	P	[All lines]
2	23	Normal	4	Check if route/time is returned correctly	Waterloo -> Bank	There is no notable difference in journey time or route length between Waterloo and Bank. This closure would be feasible.	P	[All lines]
2	24	Erroneous	4	Check if appropriate error messages appear	Bank -> Waterloo	The journey between Bank and Waterloo took 5 minutes. It now takes 19 minutes. This closure is infeasible as it takes more than double the original journey time.	P	[All lines]

Task 4 combines the usage of Kruskal's Algorithm and Dijkstra's algorithm, providing us with the final outcome. Kruskal's algorithm that runs on line 14 has a complexity of O(ElogV) and Dijkstra's algorithm on line 29, as we have seen before, has a complexity of O((V+E)log(V)). Therefore, when conjoined, they have a runtime complexity of O(ElogV+(V+E)logV).

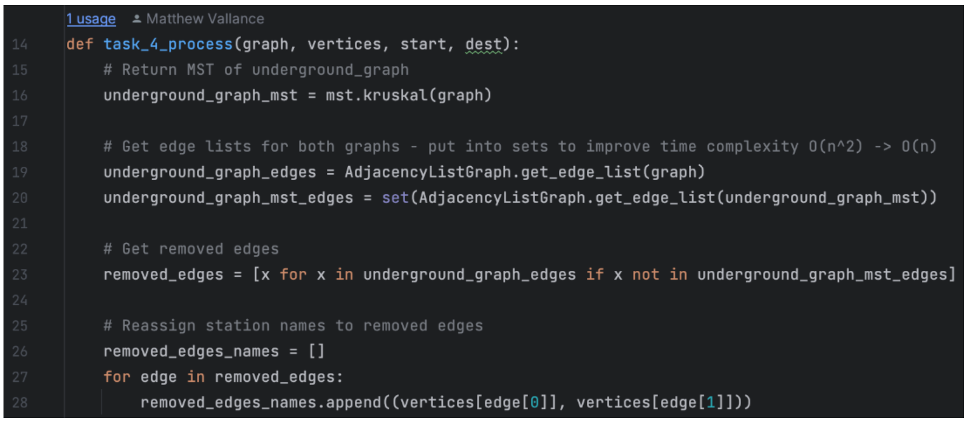
3. Results derived from the tasks, including screen-captured presentations showcasing the functionality of your application code and compliance with the use of the required library code.



This is the histogram for the duration of the journey for each of the station pairs if the government was to use our advice and close all the feasible lines. There are more stations with a longer journey in minutes. In comparison, the longest journey in the original histogram was 111 minutes whereas the new longest journey is 134 with 2 station pairs for each direction of the journey. The mode of this histogram is both 38 and 43 as they both contain the most station pairs of 1182. This graph shows the climb is more staggering than the original data. This may mean that most lines that closed had an odd number of minutes in their journey which lowered all the odd minute journeys. This is why all the odd numbers are much lower than all the even numbers and creates this staggering effect. The skewness has not been affected much although it is slightly less positive than the original histogram due to more of the station pairs being above the mode.

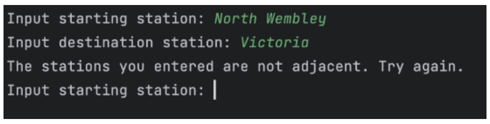


This represents the link between the number of station pairs that contains the same number of stations in its journey. This histogram represents the data for both task 2 and task 3 as they contain the same results and therefore would output the same histogram. The skewness of this histogram is less positive than the original. It is close to having a normal distribution however the initial gradient is less than the final gradient on the higher end of the graph. This is because there are more station pairs with a small number of stations between them. The mode of this histogram is 22 stations with 2666 station pairs. In comparison with the original mode of 12 stations, this is much higher which shows that overall, it would be more likely that the journey would take much longer with the line closures.

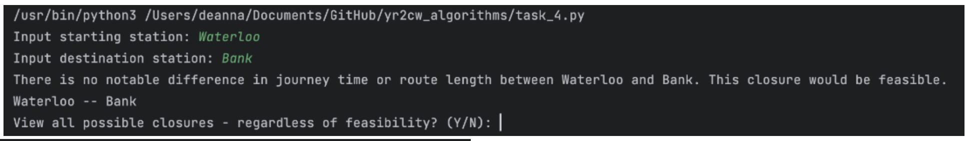


```
    1 usage  Matthew Vallance
14  def task_4_process(graph, vertices, start, dest):
15      # Return MST of underground_graph
16      underground_graph_mst = mst.kruskal(graph)
17
18      # Get edge lists for both graphs - put into sets to improve time complexity O(n^2) -> O(n)
19      underground_graph_edges = AdjacencyListGraph.get_edge_list(graph)
20      underground_graph_mst_edges = set(AdjacencyListGraph.get_edge_list(underground_graph_mst))
21
22      # Get removed edges
23      removed_edges = [x for x in underground_graph_edges if x not in underground_graph_mst_edges]
24
25      # Reassign station names to removed edges
26      removed_edges_names = []
27      for edge in removed_edges:
28          removed_edges_names.append((vertices[edge[0]], vertices[edge[1]]))
```

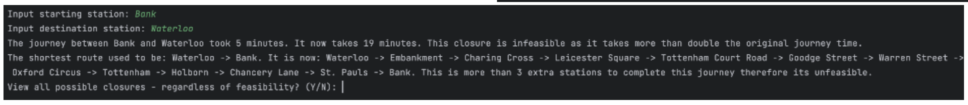
As shown above we have called in the Kruskal's Algorithm on line 16 of the code. The Dijkstra Algorithm was later called indirectly through the usage of task 1's code later within the code, we decided this would be more efficient as it performs the same task just using a different data set.



```
Input starting station: North Wembley
Input destination station: Victoria
The stations you entered are not adjacent. Try again.
Input starting station: |
```



```
/usr/bin/python3 /Users/deanna/Documents/GitHub/yr2cw_algorithms/task_4.py
Input starting station: Waterloo
Input destination station: Bank
There is no notable difference in journey time or route length between Waterloo and Bank. This closure would be feasible.
Waterloo -- Bank
View all possible closures - regardless of feasibility? (Y/N): |
```



```
Input starting station: Bank
Input destination station: Waterloo
The journey between Bank and Waterloo took 5 minutes. It now takes 19 minutes. This closure is infeasible as it takes more than double the original journey time.
The shortest route used to be: Waterloo -> Bank. It is now: Waterloo -> Embankment -> Charing Cross -> Leicester Square -> Tottenham Court Road -> Goodge Street -> Warren Street ->
 Oxford Circus -> Tottenham -> Holborn -> Chancery Lane -> St. Pauls -> Bank. This is more than 3 extra stations to complete this journey therefore its unfeasible.
View all possible closures - regardless of feasibility? (Y/N): |
```

These examples show that the stations must be directly next to each other for this code to output the if it is feasible. We have included two examples of the same station pairs which shows that it is possible to close the line in one direction and not for the other.

4. Final remarks, along with an insightful discussion addressing the limitations of the undertaken work.

When reviewing our solution and code we found that, the Kruskal Algorithm does not always return the same result of Bank -> Waterloo and will instead output different, yet still correct results. Additionally, the Kruskal Algorithm completely relies on the edge weights to construct the Minimum Spanning Tree (MST), which could present a problem if we had to factor in other aspects of the scenario. If we had to involve other aspects such as cost or passenger preferences the algorithm would not be able to capture the complex nature of its given task. On the other hand, the use of the Dijkstra Algorithm when completing this task is the best way, we could see to do it as we should not have any negatively weighted edges and, out of the two algorithms compared across this report, it’s the more time efficient option when approaching the task.

A sequentially organized progress journal with weekly entries, marked by dates, detailing communication logs (along with accumulated credit for each member), an outline of each member's contributions, compliance with the given format, language clarity, etc.

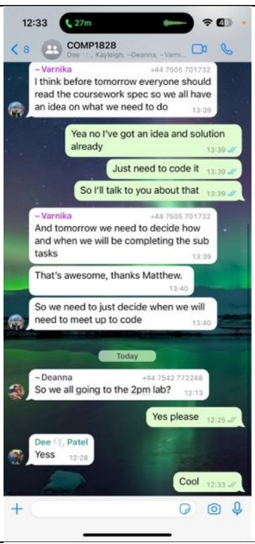
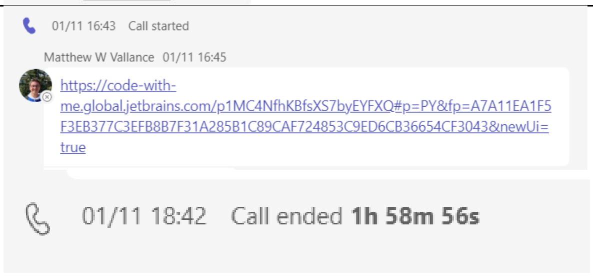
Week	Progress, including descriptions of contributions from each member	
23/Oct -	**Name**	**Description**
	Matthew	Started creating the data set needed for all the tasks and drafted a solution to task 1.
	Deanna	Started to research into how to complete the histograms needed for the report.
	Varnika	Prepared the document layout.
	Kayleigh	Pair programmed task 1 with Matt.
	Deeya	Helped debug the data set code.
30/Oct -	**Name**	**Description**
	Matthew	Completed tasks 1 and 2 then left them for debugging, started on task 3.
	Deanna	Focused on pair programming the code, helped with debugging the code.
	Varnika	Focused on pair programming the code.
	Kayleigh	Focused on pair programming, also gathered communication screenshots for the week.
	Deeya	Helped debug the code left by Matthew.
6/Nov -	**Name**	**Description**
	Matthew	Completed task 3 and left for debugging, started on task 4.
	Deanna	Completed the Histogram for task 1.
	Varnika	Started the write up for task 1 and 2.
	Kayleigh	Focused on pair programming, also gathered communication screenshots for the week.
	Deeya	Helped debug the code left by Matthew.
13/Nov -	**Name**	**Description**
	Matthew	Completed the code needed for the solutions.
	Deanna	Completed the Histogram for task 2.
	Varnika	Started the write up for task 3.
	Kayleigh	Focused on pair programming, also gathered communication screenshots for the week.
	Deeya	Worked on complexities.

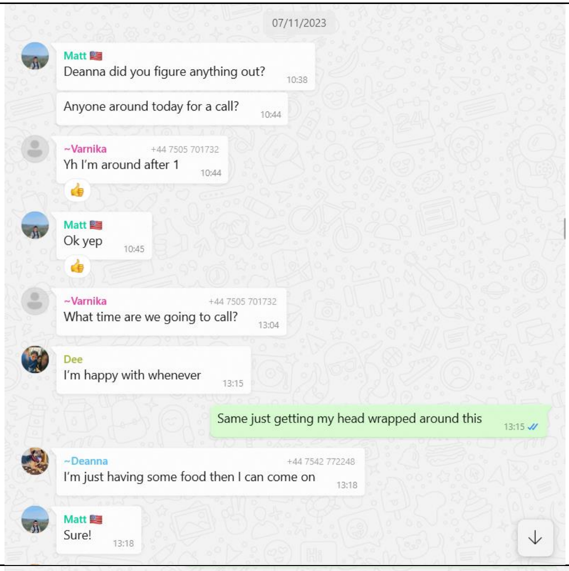
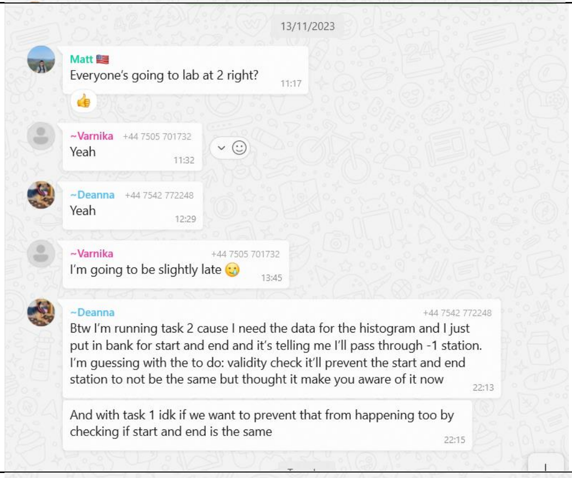
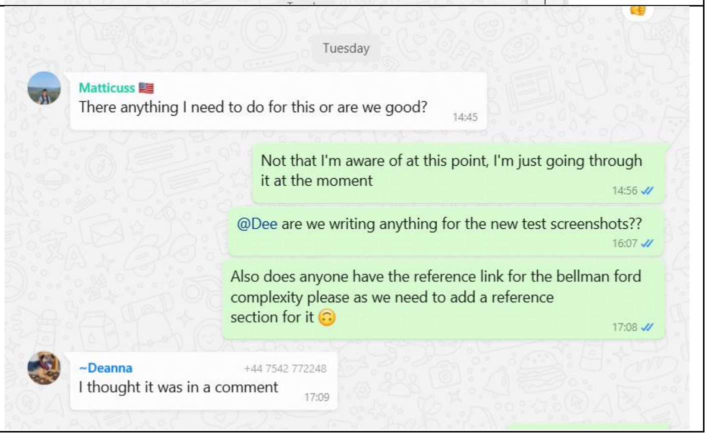
	Name	Description
20/Nov -	Matthew	Placed accurate screenshots of the code and the testing process into the document.
	Deanna	Completed the rest of the histograms, worked on complexities.
	Varnika	Focused on completing the write up for task 4.
	Kayleigh	Wrote up communication evidence and proofread the document.
	Deeya	Worked on complexities

Final credit for each member

#	Name	ID in 9 digits	Email in 7 letters (e.g. jk7492y)	Accumulated credit per member (0 -100%)
1	Vallance, Matthew	001225832	mv5742c	100%
2	White, Deanna	001208356	kw5189t	100%
3	Mogali, Varnika	001279757	vm5770h	100%
4	Harmsworth, Kayleigh	001218868	kh7920k	100%
5	Patel, Deeya	001230057	dp4381f	100%

Communication evidence:

Description	Evidence
23rd October 2023: • Group decided to host a coding session. • Group decided to all attend the same lab to discuss completion and tactics of the project	 12:33 27m COMP1828 Dee, Kayleigh, ~Deanna, ~Varni... ~Varnika +44 7505 701732 I think before tomorrow everyone should read the coursework spec so we all have an idea on what we need to do 13:39 Yea no I've got an idea and solution already 13:39 Just need to code it 13:39 So I'll talk to you about that 13:39 ~Varnika +44 7505 701732 And tomorrow we need to decide how and when we will be completing the sub tasks 13:39 That's awesome, thanks Matthew. 13:40 So we need to just decide when we will need to meet up to code 13:40 Today ~Deanna +44 7542 772248 So we all going to the 2pm lab? 12:13 Yes please 12:25 Dee Patel Yess 12:28 Cool 12:33
1st November 2023: • Organised a group meeting to go through the errors that were being presented in the code. • Concluded in us deciding to pull data from the main data set instead of creating a dictionary/array from the data.	 01/11 16:43 Call started Matthew W Vallance 01/11 16:45 https://code-with-me.global.jetbrains.com/p1MC4NfhKBfsXS7byEYFXQ#p=PY&fp=A7A11EA1F5F3EB377C3EFB8B7F31A285B1C89CAF724853C9ED6CB36654CF3043&newUi=true 01/11 18:42 Call ended **1h 58m 56s**

<u>7th November 2023</u> • The group decided on having a teams call to look over the changes made in the code. • Within the call we saw what progress had been made with the code and started to denote what was needed for answering each of the required questions in the report.	 07/11/2023 Matt 🇺🇸 Deanna did you figure anything out? 10:38 Anyone around today for a call? 10:44 ~Varnika +44 7505 701732 Yh I'm around after 1 10:44 👍 Matt 🇺🇸 Ok yep 10:45 👍 ~Varnika +44 7505 701732 What time are we going to call? 13:04 Dee I'm happy with whenever 13:15 Same just getting my head wrapped around this 13:15 ~Deanna +44 7542 772248 I'm just having some food then I can come on 13:18 Matt 🇺🇸 Sure! 13:18
<u>13th November 2023:</u> • Shows how we kept finding little bugs within the solution and were making continuous improvements to our code. • Also shows how we consistently went to the 2pm lab as a group so that we could discuss any important queries in person as to not prolong any confusion.	 13/11/2023 Matt 🇺🇸 Everyone's going to lab at 2 right? 11:17 👍 ~Varnika +44 7505 701732 Yeah 11:32 ~Deanna +44 7542 772248 Yeah 12:29 ~Varnika +44 7505 701732 I'm going to be slightly late 😢 13:45 ~Deanna +44 7542 772248 Btw I'm running task 2 cause I need the data for the histogram and I just put in bank for start and end and it's telling me I'll pass through -1 station. I'm guessing with the to do: validity check it'll prevent the start and end station to not be the same but thought it make you aware of it now 22:13 And with task 1 idk if we want to prevent that from happening too by checking if start and end is the same 22:15
<u>21st November 2023:</u> • Shows how we are in the final stages of the documentation stage. • Finding the references needed and proofreading the report was our top priority at this point.	 Tuesday Matticuss 🇺🇸 There anything I need to do for this or are we good? 14:45 Not that I'm aware of at this point, I'm just going through it at the moment 14:56 @Dee are we writing anything for the new test screenshots?? 16:07 Also does anyone have the reference link for the bellman ford complexity please as we need to add a reference section for it 🙃 17:08 ~Deanna +44 7542 772248 I thought it was in a comment 17:09

References

Geeks, G. f., 2023. *Geeks for Geeks.* [Online]
Available at: https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/
[Accessed 18 November 2023].