Shri Vile Parle Kelavani Mandal's

INSTITUTE OF TECHNOLOGY

DHULE (M.S.)

DEPARMENT OF COMPUTER ENGINEERING

Subject: Competitive Programming Lab	Remark
Name: Jaykishan Natwar Varma **Roll No.:** 68	
Class: TY. Comp. Engg. **Batch:** T4 **Division:**	Signature
Expt. No.: **Date :**	
Title: Bridge	

ASSIGNMENT/EXPERIMENT: ___________________________		
Date of Performance:	**Date of Submission:**	
Marks Split Up	**Maximum Marks**	**Marks Obtained**
Performance/Conduction	**3**	
Report Writing	**3**	
Attendance	**2**	
Viva/Oral	**2**	
Total Marks	**10**	
Signature of Subject Teacher		

Title: Bridge

Aim: n people wish to cross a bridge at night. A group of at most two people may cross at any time, and each group must have a flashlight. Only one flashlight is available among the n people, so some sort of shuttle arrangement must be arranged in order to return the flashlight so that more people may cross. Each person has a different crossing speed; the speed of a group is determined by the speed of the slower member. Your job is to determine a strategy that gets all n people across the bridge in the minimum time.

Language used: Python

Platform Used: Pycharm, VS code etc.,

Sample Input: The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs. The first line of input contains n, followed by n lines giving the crossing times for each of the people. There are not more than 1000 people and nobody takes more than 100 seconds to cross the bridge.

Sample Output: For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line. The first line of output must contain the total number of seconds required for all n people to cross the bridge. The following lines give a strategy for achieving this time. Each line contains either one or two integers, indicating which person or people form the next group to cross. (Each person is indicated by the crossing time specified in the input. Although many people may have the same crossing time the ambiguity is of no consequence.) Note that the crossings alternate directions, as it is necessary to return the flashlight so that more may cross. If more than one strategy yields the minimal time, any one will do.

Example:

Sample Input:

```
1


4
1
 2
 5
 10
```

Sample Output:

17

1 2

1

5 10

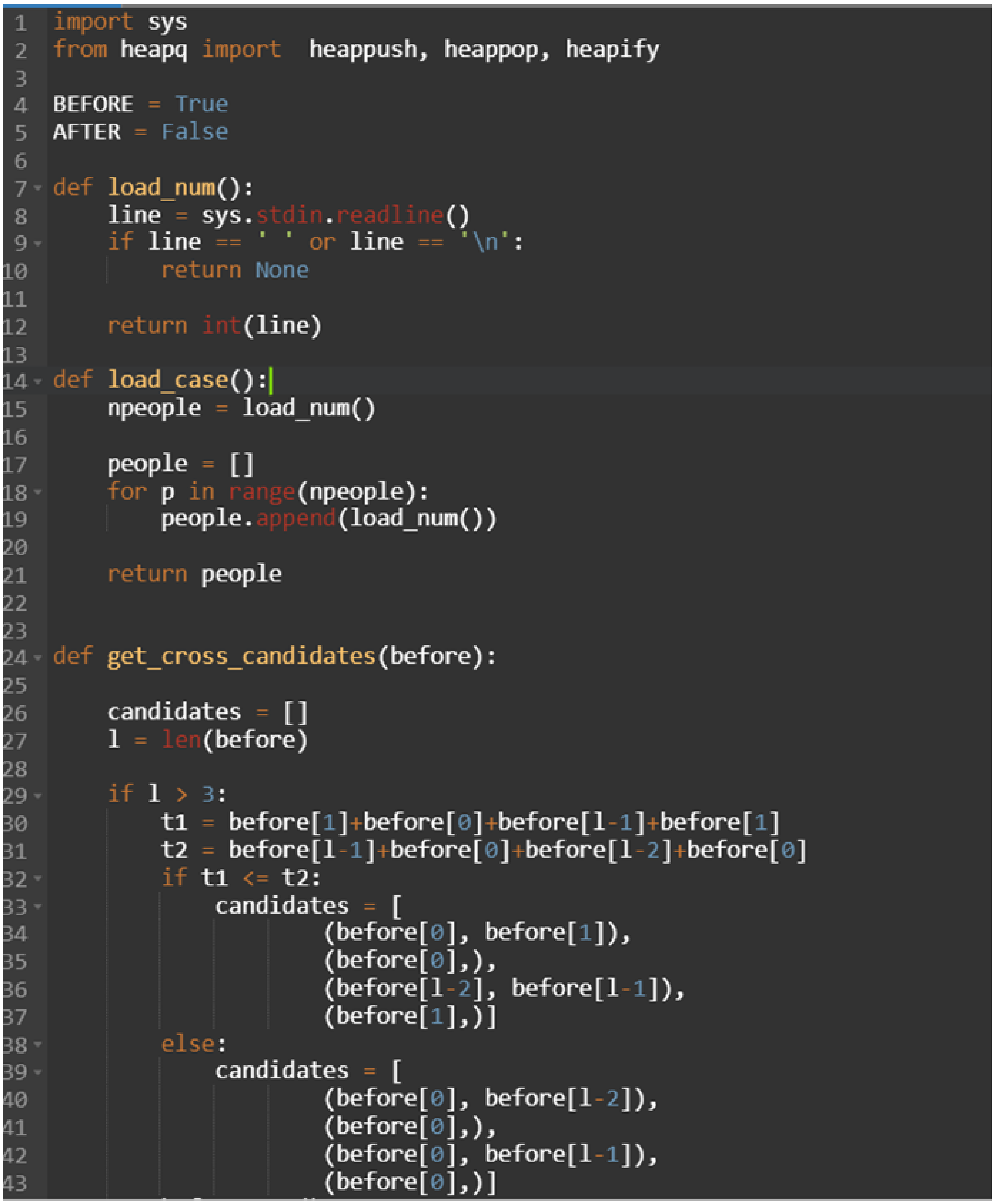
2

1 2

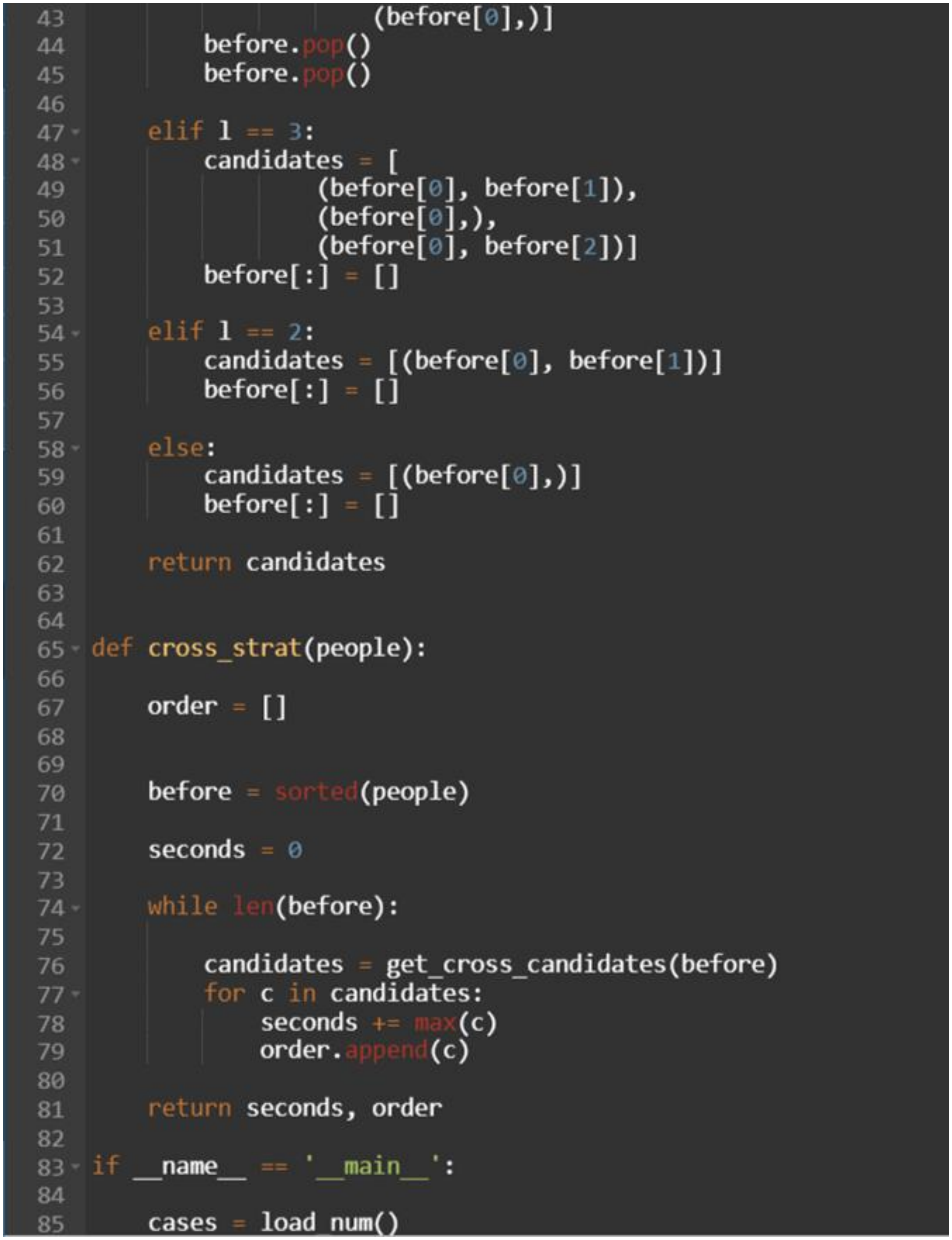
Algorithm/Flowchart:

```
function min_crossing_time(people):
    // Sort the list of people by their crossing time (in ascending order)
    sort(people)

    // Use a priority queue (min-heap) to simulate the crossing process
    priority_queue = []  // Priority queue to track the time taken for each group to cross
    total_time = 0

    // Iterate over each person in sorted order
    for person in people:
        // If the priority queue is empty, simply add the person's time to the queue
        if empty(priority_queue):
            push(priority_queue, person.time)
        else:
            // Otherwise, combine the current person's time with the smallest time in the queue
            current_time = pop(priority_queue)  // Get the smallest time from the queue
            new_time = max(current_time, person.time)  // Calculate the new crossing time
            total_time += new_time  // Update the total crossing time
            push(priority_queue, new_time)  // Push the new crossing time back into the queue

    // Process the remaining times in the priority queue
    while not empty(priority_queue):
        total_time += pop(priority_queue)  // Add the remaining times to the total crossing time

    return total_time

// Example usage:
people = [(1, 2), (2, 5), (5, 10), (10, 15)]
result = min_crossing_time(people)
print("Minimum crossing time:", result)
```

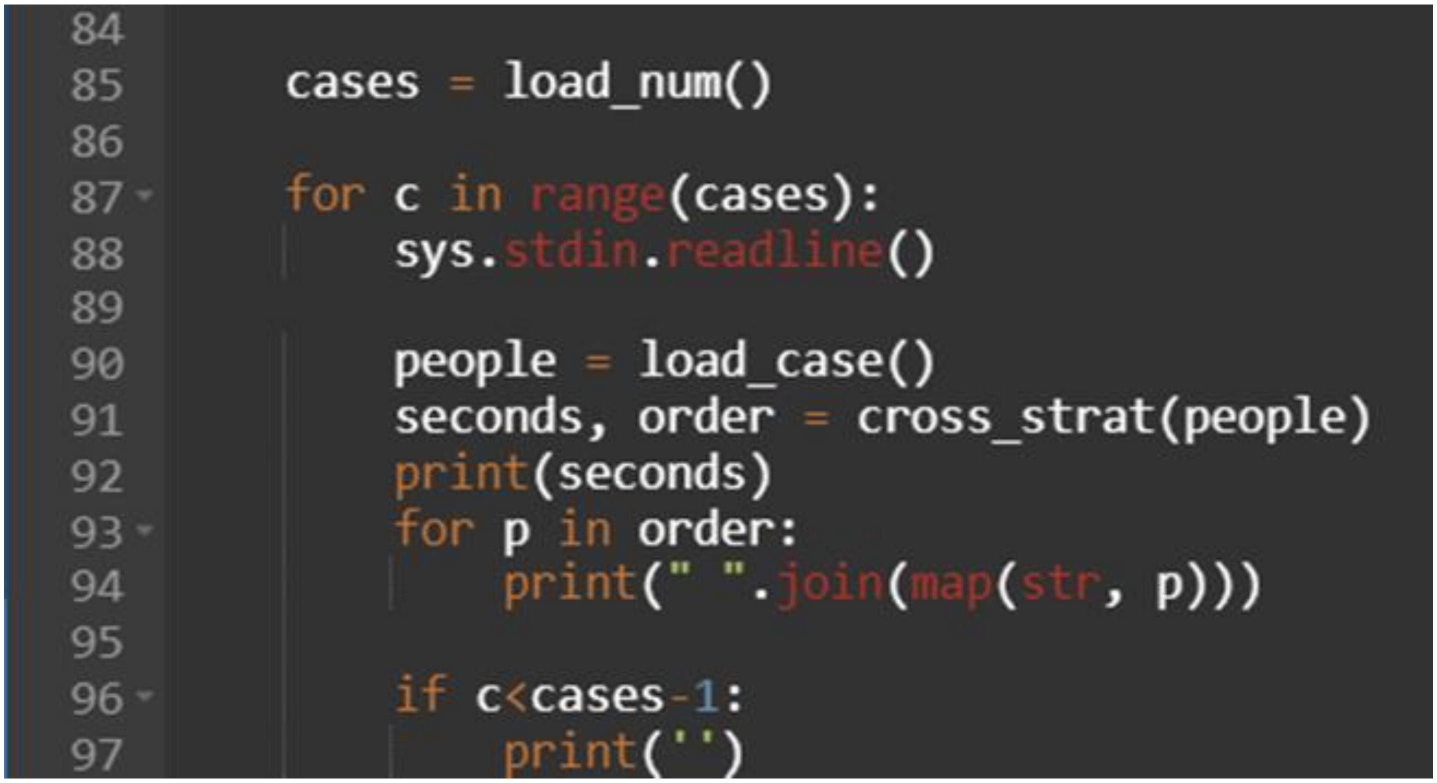
Code:



```python
import sys
from heapq import  heappush, heappop, heapify

BEFORE = True
AFTER = False

def load_num():
    line = sys.stdin.readline()
    if line == ' ' or line == '\n':
        return None

    return int(line)

def load_case():
    npeople = load_num()

    people = []
    for p in range(npeople):
        people.append(load_num())

    return people


def get_cross_candidates(before):

    candidates = []
    l = len(before)

    if l > 3:
        t1 = before[1]+before[0]+before[l-1]+before[1]
        t2 = before[l-1]+before[0]+before[l-2]+before[0]
        if t1 <= t2:
            candidates = [
                    (before[0], before[1]),
                    (before[0],),
                    (before[l-2], before[l-1]),
                    (before[1],)]
        else:
            candidates = [
                    (before[0], before[l-2]),
                    (before[0],),
                    (before[0], before[l-1]),
                    (before[0],)]
```

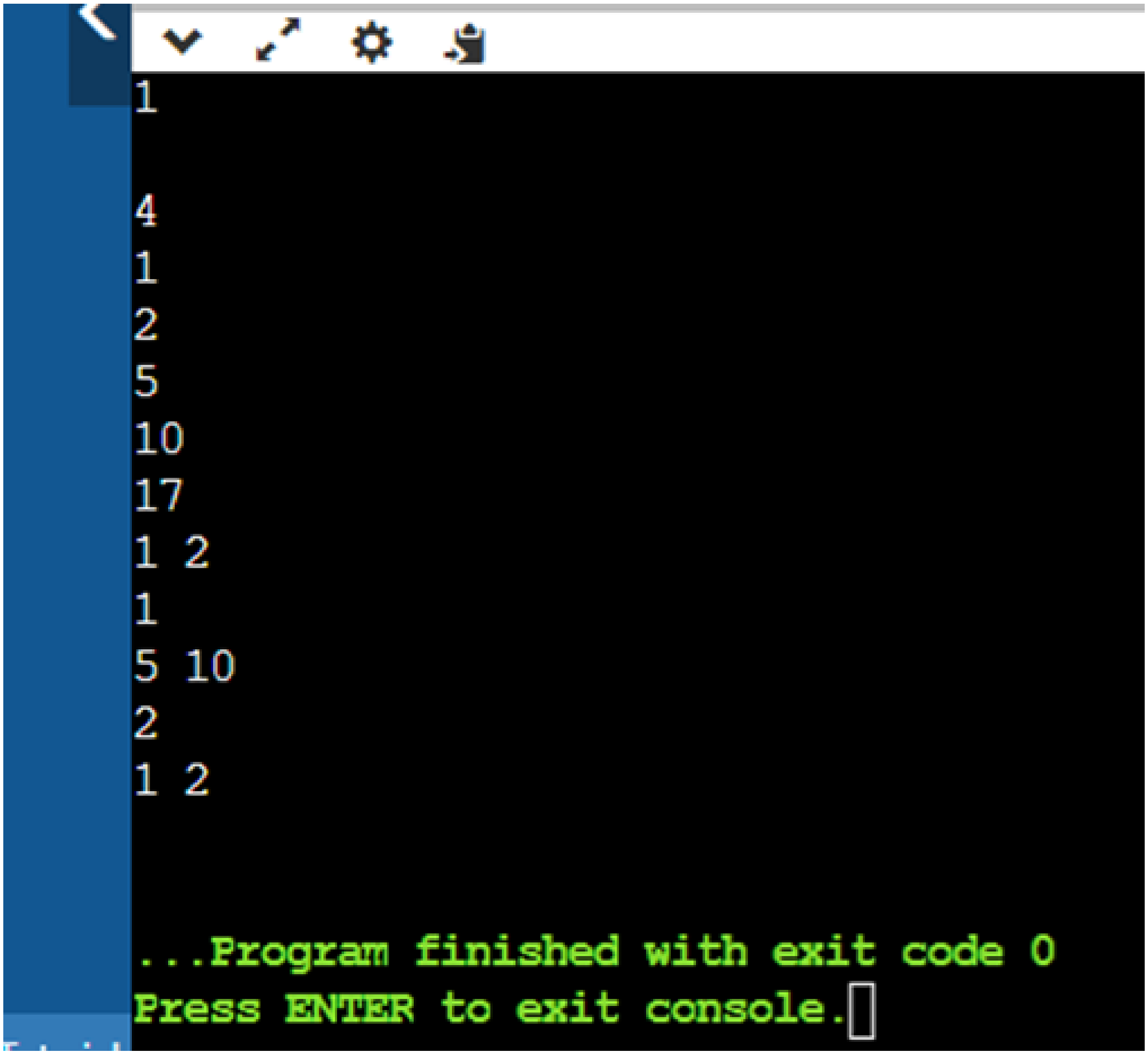


```
                    (before[0],)]
        before.pop()
        before.pop()

    elif l == 3:
        candidates = [
                (before[0], before[1]),
                (before[0],),
                (before[0], before[2])]
        before[:] = []

    elif l == 2:
        candidates = [(before[0], before[1])]
        before[:] = []

    else:
        candidates = [(before[0],)]
        before[:] = []

    return candidates


def cross_strat(people):

    order = []


    before = sorted(people)

    seconds = 0

    while len(before):

        candidates = get_cross_candidates(before)
        for c in candidates:
            seconds += max(c)
            order.append(c)

    return seconds, order

if __name__ == '__main__':

    cases = load_num()
```



```
    cases = load_num()

    for c in range(cases):
        sys.stdin.readline()

        people = load_case()
        seconds, order = cross_strat(people)
        print(seconds)
        for p in order:
            print(" ".join(map(str, p)))

        if c<cases-1:
            print('')
```

Output:-



```
1

4
1
2
5
10
17
1 2
1
5 10
2
1 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:
In this way we can implement the bridge problem using dictionary, loops and conditional statements.