COMP3121 Assignment 1

Ziyi Cui z5097491

1.(a).

```
S.mergeSort()                   #Sort the integer array by mergeSort
For i in s                      #Go through the array
    goal = x – s[i]             #get the goal which is equal to the result of
                                # x subtract the current integer
    If binarySearch(S, goal)    #Search the goal by binary search
        return True             #If we find the goal then we find there are 2
                                #integers in the array, and sum of them equal
                                #to x
Return False                    #If we do not find the goal after go over the
                                #Integer array,then there are no such 2 integers
                                #that sum of them equal to x
```

1.(b).

```
dic = {}                        #Init an empty dictionary
For i in S                      #Loop though the integer array(S):
    goal = x – S[i]             #GET the goal which we need to find in the dic
    If goal in dic              #If we already seen the goal in the array
        Return True             #then we finished the task return Ture
    else
        dic[S[i]] = i           #else we put the current integer in the dic
return False                    #If we do not find the goal after go over the
                                #Integer array,then there are no such 2 integers
                                #that sum of them equal to x
```

We only go through the array exactly once, so the worst case of this algorithm is O(n)

2.

1. Sort the integer array(nums) by merge sort
2. Go through the queries' loop
    a. If R < L, then put 0 to results list and continue to next query
    b. low_index = higherbound(nums, L) #find left index
    c. high_index = lowerbound(nums, R) #find right index
    d. num = high_index – low_index – 1 #calculate the number of

#values between L and R

e. if L < nums[0] #if L < the least value
    i. num += 1 #in the array then add 1
f. if R > nums[-1] #if R > the biggest value
    i. num += 1 #in the array then add 1

g. append the num into results list

3. return the results list

helper functions(aim to find the higher/lower bound index)
Modified binary search function – O(logn)

```
higherbound(nums, goal)
    lo = 0
    hi = len(nums) – 1
    while lo < hi
        mid = (hi+lo)/2
        if nums[mid] > goal:
            hi = mid
        elif nums[mid] <= goal:
            lo = mid + 1
    return hi
lowerbound(nums, goal)
    lo = 0
    hi = len(nums) – 1
    while lo < hi
        mid = (hi+lo)/2
        if nums[mid] >= goal:
            hi = mid - 1
        elif nums[mid] < goal:
            lo = mid
    return lo
```

it's doesn't matter to return hi or lo, because lo == hi after the while loop

3.

There are N teams and N friends, so there are 2^N many choice in total
The possibility of friends' subset is P(F) = N/2^N, which means the possibility of we choose distinct subset is P(D) = (N-1)*N/2^N. Hence, we can solve this problem by asking N questions.
For example

Friends: A, B, C, D ……
Teams: 1, 2, 3, 4 …….

Let i = 0
Loop N times:
    Ask does Friends[i] support Team[i]
    If True
        Then, we do not support Team[i]
    Else
        We support Team[i]
After the loop we get the final list of the subset that which teams we support.
In this list, there at least one team we support or not is different from the N friends' subset.
Therefore, we can ensure that our subset is not identical to anyone else.

4.

First of all, we can use the method similar to the MSD radix sort.
We create n buckets with equal interval, and the length of interval is 1/n.
Let k be the value from 0 to n-1. We label buckets with the number k.
If $k/n <= x_i < (k+1)/n$, then we put x_i into bucket with label k.
The cost of putting all the numbers to the n buckets is O(n)
After that, if there are more than 1 elements in the same buckets, then find the maximum value and the minimum value and put them to last of the bucket and first of the bucket.O(n)
Right now we can start to prove:
$\sum_{i=2}^{n} |y_i - y_{i-1}| = \sum_{i=2}^{n} |y_i - y_{i-1}|$(i and i-1 in the same bucket) + $\sum_{i=2}^{n} |y_i - y_{i-1}|$(i and i-1 in the different bucket)
If all the value in the same bucket, then the maximum of $\sum_{i=2}^{n} |y_i - y_{i-1}| = (n-1)*1/n = (n-1)/n < 1$, which is the sum of n – 1 times subtraction times the maximum value of the subtraction.
If all the values in the different buckets, then the maximum of $\sum_{i=2}^{n} |y_i - y_{i-1}| = 1$, which is sum of different intervals.
To sum up, $\sum_{i=2}^{n} |y_i - y_{i-1}| < 1 + 1 < 2$

5.(a).

Ask A if s/he know other person B, there are 2 results in total:

1. A know B
2. A don't know B

If case 1, then A is not a celebrity

Else, B is not a celebrity

Therefore, we can start asking questions from person 1 to person N

Assume we have a list, which contains people from person 1 to person N

i = 0, j = 1

While len(person) > 1

Ask person[i] if s/he know person[j]

If True

Then we know person[i] is not a celebrity so that we ask another person

Drop person[i] from list, then person[i] become to the next person and person[j] as well

Else

Then we know person[j] is not a celebrity so that we ask another person

Drop person[j] from list, then person[j] become to the next person

(The list size will reduce by one once a question is asked, therefore we only need to ask N-1 questions)

After the loop the last person in the list is the celebrity candidate

After that, we ask the candidate does s/he know all other persons. If he know one or more of them, then no celebrity present(we need ask at most N-1 questions)

Then, we ask all other persons if they know the candidate. If one or more of them don't know the candidate, then no celebrity present (we need ask at most N-1 questions)

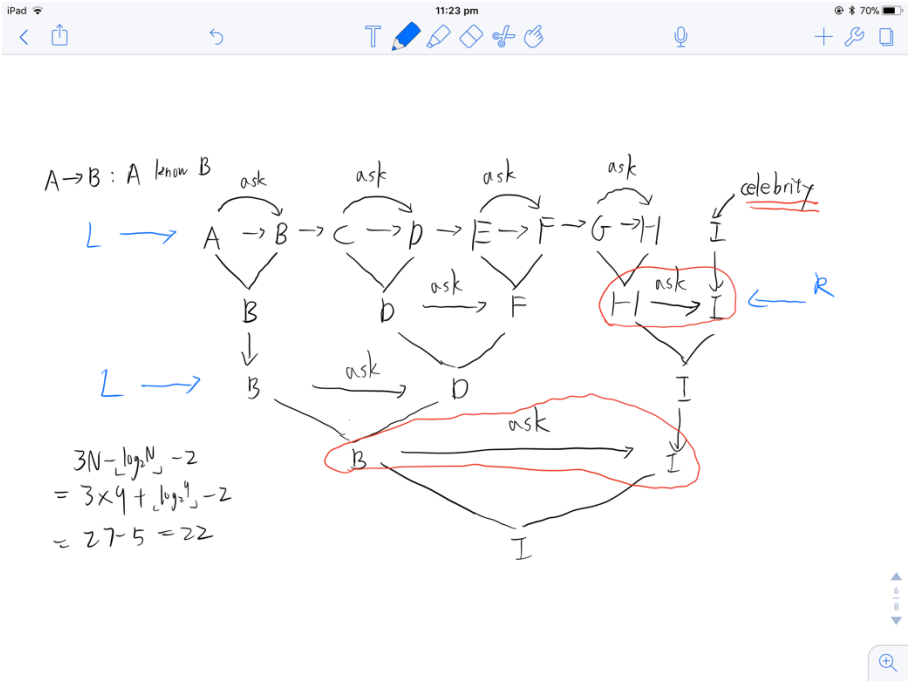
In this solution, we can asks at most 3N – 3 questions to ensure there is a celebrity present or not.

(Improve: The celebrity candidate has been asked by at least 1 person or there at least 1 person has been asked by the celebrity candidate during the first loop. Hence, we can reduce at least one question in the fellow 2N – 2 questions)

5.(b).

The part b is basically similar to part a, but we need to modify the first loop in part a to make sure the questions asked in the first loop can be used in the next two (N - 1) loop.

Follow the picture below:



Form the pairs from left hand side, let A ask B and C ask D …….
From the part a, we can drop one person from a pair by asking one question.
After we select candidate from the pairs, we repeat the last operation but form the pairs from the right hand side(To make sure all people can be involved in the questions at most every 2 rounds, so we need to form pairs from different sides each time). Then, repeat the operations above, we can get I is the celebrity candidate by asking N – 1 questions.
The red circle questions do not need to be asked in the next 2N – 2 questions(in part a), which means we only need to ask another 2N – 4 questions to solve this specific problem.
For the general cases, N numbers can guarantee ceil($\log_2 N$) rounds to find the celebrity candidate. The worst case for this solution is that the number of persons generated by last round always remains odd(N = 5, 9, 17….. are the worst cases).
The number of reused questions >= floor($\log_2 N$) – 1 (equal to floor($\log_2 N$) – 1 are the worst cases)
(N is power of 2 will be the best cases, then we can assume N is not power of 2)
The worst case = 3N – 3 – floor($\log_2 N$) + 1
= 3N - floor($\log_2 N$) – 2

Even the worst cases we still can solve this problem in (3N - floor($\log_2 N$) – 2) questions.

6.(a).

	A	B	C	D
A		1	1	1
B				
C			1	
D				

Assume we have 4 students and 4 pieces of vote paper:

1. B – A 2. C – A 3. D – A 4. D – C (A - B means B vote for A)

We can know how many votes each student received no matter how many the order of name is right or wrong.

Every student can vote for one person. If the order of vote is wrong, then there might have some students vote for more than 2 people. We can know only one vote is voted by this student, so other votes must be others students voted for her/him.

The number of votes the student received = The number of the student vote for others – 1 + The number of votes s/he received.

6.(b).

If there is a student's name only appear on the pieces of paper once, then no one vote for her/him.

The person s/he vote for is on the paper, which has his/hers name.

6.(c).

n! many choice

each student can get exactly 1 vote

6.(d).

O(n^2) solution:

```
Create a 2D array(list)
for vote in votes:
        list[vote.name1][vote.name2] = 1
create an empty_list
```

```
For i in range(N)
    If list[i].empty()
        empty_list.append(i)
```
(every student have to vote for another student, list[i] is empty means student i didn't vote for anyone. Apparently, student i mess up the order of name on the piece of paper)

```
While not empty_list.empty():
    Index = Pop_front(empty_list)
    tuple = [0,0]
    For key in range(N):
        If list[key][index] and len(list[key]) > tuple[1]:
            tuple = [key,len(list[key])]
    key = tuple[0]
    list[key][index] = 0
    list[index][key] = 1
    if list[key].empty()
        empty_list.append(key)
```

Every row of list has exactly one 1-value at the end of loop.
Then, we convert the 2D array to the results list.
Right now we get one solution for this problem.

O(n) solution:

Create 2 list of empty N dictionaries (from_to, to_from)

We store all votes name like this in to the N dictionaries. It can be done in linear time

```
For vote in votes:
    from_to[vote.name1][vote.name2] = 1
    to_from[vote.name2][vote.name1] = 1
```

```
create an empty_list
For i in range(N)
    If from_to[i].empty()
        empty_list.append(i)
```
(every student have to vote for another student, from_to[i] is empty means student i didn't vote for anyone. Apparently, student i mess up the order of name on the piece of paper)

(Now we starting to correct the wrong votes)

```
While not empty_list.empty():
        Index = Pop_front(empty_list)
        tuple = [0,0]
        For key in to_from[index]:
                If len(from_to[key]) > tuple[1]:
                        tuple = [key,len(from_to[key])]
        key = tuple[0]
        to_from[key][index] = 1
        from_to[index][key] = 1
        del to_from[index][key]
        del from_to[key][index]
        if from_to[key].empty()
                empty_list.append(key)
```

At this moment every dictionary has exactly one element.
Then, we convert the from_to to the results list.
Right now we get one solution for this problem.

Let consider time complexity of this solution:
First of all, while loop will never goes to an infinite loop, because the program always pick the votes from the person who has voted the most to other people.
if the len(to_from[index]) in the last for-loop is n-1 then our empty list only contain one element.
If the len(empty_list) is n – 1, then the length of dictionaries in to_from cannot larger than one.
If the len(empty_list) is m, then the length of dictionaries in to_from cannot larger than n - m.
Each loop we can reduce the sum of length of empty_list and to_from[index] at least by 1
Therefore, we can use this solution to solve this problem in linear time O(n)