Q1

```
while(n>0){
    T1 makes a pile P which he believes is 1/n of the whole loot;
    TargetTheif = T1;
    for(TargetTheif proceeds to ask Ti from 2 to n if Ti agrees that P <= 1/n){
        if(Ti says YES){
            if(i == n){
                TargetTheif takes P;
            }else{
                continue;
            }
        }else{
            if(i == n){
                Ti takes P;
            }else{
                Ti reduces the size of P so that he thinks P = 1/n
                TargetTheif = Ti;
                continue;
            }
        }
    }
    n--;
}
```

Q2

There are only 19 posibilities(0-18), So everyone's handshake number is different.
Mr/Mrs 18 shakes hand with everyone except his/her wife/husband. So (0, 18) must be a couple.
If Mary's handshaking number is 0, it means she never shook hands with anyone.
So 18 would not be appeared and Tom would not get 19 different answers.
Same as if Mary shook hands 18 times.
Thus, couple(0, 18) is not Tom and Mary.
Remove this couple.
Everyone's handshaking number would be minus 1 because Mr/Mrs 18 shook hands with everyone.
The range of posibilities would be 0 to 16.
Repeat the analysis above, we could remove 8 couples until the range is 0 to 2.
couple(0, 2) is not Tom and Mary.
So obviously, Tom and Mary shook hands 1 time with Mr/Mrs 2.
Adding the other 8 couples, Tom and Mary shook hands 9 times in this party.

Q3

First, there is only one circular highway around the city. If add a one-way
street, there would be two blocks and one of them is circumnavigated. So we

still consider the circumnavigated one as the circular hihgway and ignore the other part. So we could repeat the process above and no matter how many streets are added, there is one circumnavigated block.

Q4

There are 5 pirates P1,P2,P3,P4,P5

case 1: there are only two left, P4 and P5

P5 will reject the proposal whatever it is and takes 100 bars.

case 2: there are 3 pirates left, P3, P4 and P5

P4 will accept the proposal to avoid case 1. So the proposal made by P3 would be (P3=100, P4=0, P5=0)

case 3: there are 4 pirates, P2, P3, P4 and P5

P3 kowns that if P2 is killed, he could get all gold. So he will reject the proposal.

P2 need two supporters to make his proposal accepted.

P2 will propose (P2=98, P3=0, P4=1, P5=1)

case 4: there are 5 pirates, P1, P2, P3, P4 and P5

Only P1 is killed, case 3 would be achieved. So P2 would reject.

If P1 is killed, P3 would get 0 bar of gold. So if P1 gives him more than 0 bar of gold, he will accept the proposal.

P1 need 2 supporters. He could give P3 1 bar to get one support.

Then he will choose P4 or P5(both are ok), give one of them 2 bars.

So the proposal is (P1=97, P2=0, P3=1, P4=2, P5=0) or (P1=97, P2=0, P3=1, P4=0, P5=2)

Q5

(a) Sorting the array S by merge sort. ----------> Θ (nlog2 n)

```
for(int i = 0; i < n; i++){
    int value = x - S[i];
    Searching value in array S by Binary Search ----------> Θ (log2 n)
}
```

The time complexity is Θ(2nlog2 n)

(b) Store the array S into a hash map. Using the value as the key.

So the time complexity of searching would be O(1).

```
for(int i = 0; i < n; i++){
    int value = x - S[i];
    Searching value from hash map ----------> Θ(1)
}
```

The time complexity is Θ(n).

Q6

```
numbers[] = randomArray() // n numbers
buckets[9][n]
for currentPosition in range(0,8){
    for(number:numbers){
        Get the digit on the currentPosition.the digit may be greater than 10
```

```
        because the number is base n.
        value = (number/power(n, currentPostion))%currentPosition
        buckets[value].append(number)
    }
    move numbers from buckets to Array numbers one by one and clear buckets.
}
```

the time complexity is O(9n)

Q7

Create n buckets and the size of each bucket is 1/n.
Put the numbers into proper bucket one by one.
Reorder the numbers by the order of buckets. If there are many numbers in one bucket, rank them as the input order.
Then calculate as the formula by the new order.
The worst case is that all numbers except one are all in the first bucket. For every two numbers X and Y, |x-y|<1/n. The only one number is in the last bucket, |LastButTwo – Last| < 1
So $\sum_{i=1}^{n-1} |y_{i+1}+y_i| < 2$

Q8

Choose a kid and find a right size shoe. The time complexity is O(n).
Using the kid's foot as a pivot to sort the shoes, put the larger shoes on the right and smaller shoes on the left.
Using the shoe as a pivot to sort the kids too.
Repeat the process like the quick sort and we could sort both kids and shoes.
Then wear the shoes one by one.
The time complexity is O(nlogn).

Q9

```
quick sort array1                    ----------> Θ(nlog2 n)
for(number : array2)
    binary search number in array1   ----------> Θ(log2 n)
```

the time complexity is O(2nlog2 n)

Q10

```
sum = 0
for(number : array)
    sum += number
result = sum - 4950
```

Q11

Because the total quantity of petrol on all stations is enough, there must be a station which has enough petrol to next station. We could move the petrol form its next station to it and delete the next station. Now we have reduce one station and nothing else changed. Repeat deleting the station until there is only one station. This station would be the one as a starting point.

Q12

(1) f(n)=n^2 g(n)=(n-2log_2^n)(n+cosn)
1/2n^2 < (n - 2 log_2^n)(n + cos(n)) < 2*n^2
Thus, f(n) = Θ(g(n))

(2) $n^{100} = 2^{\frac{n}{100}}$

n > C, f(n) < g(n)
f(n) = O(g(n))

(5) $\sqrt{n} = 2^{\sqrt{log_2^n}}$

$\sqrt{2^x} = 2^{\sqrt{log_2^{2^x}}}$

$2^{x/2} = 2^{\sqrt{x}}$

X= 4
X > 4, f(n) > g(n)
f(n) = Ω(g(n))

(6) $n^{100} = 2^{(log_2^n)^2}$

$2^{100n} = 2^{n^2}$

n = 100
X > 100, f(n) < g(n)
f(n) = O(g(n))

(7) $n^{1.001} = nlog_2^n$
$2^{1.001x} = 2^x x$
$2^x 2^{0.001x} = 2^x x$
$2^{0.001x} = x$
X > C, f(n) > g(n)
f(n) = Ω(g(n))

Q13

```
for(int i = 0; i < array.length; i += 2){
    a = array[i];
    b = array[i+1];
    if(a<b){                          ------------------------> comparison
```

```
            if(i == 0){
                min = a;
                max = b;
            }else{
                if(a < min){          ------------------------> comparison, except first two numbers.
                    min = a;
                }
                if(b > max){          ------------------------> comparison, except first two numbers.
                    max = b;
                }
            }
        }else{
            if(i == 0){
                min = b;
                max = a;
            }else{
                if(b < min){          ------------------------> comparison, except first two numbers.
                    min = b;
                }
                if(a > max){          ------------------------> comparison, except first two numbers.
                    max = a;
                }
            }

        }
}
```

The for loop will repeat 2^(n-1) times.
Using 3*2^(n-1) - 2 comparisions.

Q14

Make many groups and each one contains 2 integers.
Compare in each group and put the larger one into a new array.
Repeat to process the new array as the method above.
Get the maximum number and compare 2^n - 1 times.
There are n numbers which compared with the maximum number.
Traverse them and get the greatest one.
Compare time is n - 1.
So the comparison times is 2^n+n-2.
But we do not know which one is the maximum number at the beginning. So we should store every number's comparison list. The space complexity is very large.

Q15

(a)
```
target=0;
//find the person who may be the celebrity.
for(int i = 1; i < n; i++){
    if(target knows i){                    ----------------------------> ask question
        target = i;
    }else{
        continue;
    }
}
boolean isCelebrity = true;
for(int i =0; i <n, i != targetl; i++){
    if(i does not know target || target knows i){ ------------>  ask 2 questions
        isCelebrity = false;
        break;
    }
}
```

So the in the worst situation, we need ask 3(n-1) questions.

(b) When we find the person who may be the celebrity, we could make every two as a group and ask them question. By this way, we could romve n/2 persons. Then reapeat until there is the only one who may be celebrity. After that we ask n-1 questions.

Next we should get the answer whether everyone knows him and he does not know any one. In question (a), we ask 2*(n-1) questions. But we could avoid to asking the persons who was the group member with this "celebrity". So we only need to ask 2*(n-1) - logn.

So the total number of questions is 3n-logn-3.