Maoting Ren
mren

1. if the guess is too low, the final grade will drops by x, and if the guess is too high, the final grade will drops by y. And if x equals to y, it will be pretty straight forward, we can use binary search to guess the number. And in the worst case, it will search (log n) times, and each time the final grade will drops by x (as x equals to y), so z = xlog n.

But if x is not equals to y, to ensure the loss of each guess is balance, we can divide the array into two halves by the following way.

yn/(x+y)	xn/(x+y)

2.

① Firstly we traverse the array, find all distinctive elements and store them in a hash table named **DistinctiveNumber**, and the last distinctive number **Num**.

② Then we traverse the array again, until find **Num**, we say this position is **Pos1**, and we know all distinctive numbers have appeared when we find **Num**, because **Num** is the last distinctive number.

③ Then we continue to traverse the array from **Pos1+1**, and this time we need a hash table. Put the distinctive numbers we encounter into the hash table until we find all numbers appeared in **DistinctiveNumber** , we say this position is **Pos2**, if we can't find all distinctive numbers from **Pos1+1** to the end, then we can say the minimum length is **2**, because some numbers missing in this interval, so we can construct a sequence with the missing numbers. If **Pos2** is the end position of the array, then the minimum length is **3**.

④ If we found the position **Pos2**, then we continue to traverse the array from **Pos2+1**, we still need a new hash table. Just like the above step, put the distinctive numbers into this hash table until we find all numbers appeared in **DistinctiveNumber**, we say this position is **Pos3**. If we can't find all distinctive numbers from **Pos2+1** to the end, then we can say the minimum length is **3**, because some numbers missing in this interval, so we can construct a sequence with the missing numbers. If **Pos3** is the end position of the array, then the minimum length is **4**.

⑤ Do the same operation to the left array, and we can get our answer in the end.

For an example, [1 2 1 3 2 3 1 2], we can find the distinctive numbers are [1 2 3], and 3 is the last instinctive number, that is **Num,** and **Pos1** is 3, that is all numbers appeared till we find number 3.

```
[1  2  1  3  2  3  1  2]
         ^
```

And we traverse the array again from **Pos1+1**, until [1 2 3] all appeared, we find the **Pos2**

```
[1  2  1  3  2  3  1  2]
               ^
```

Then next step, we traverse the array from **Pos2+1**, and can not find all numbers until end of the array, so we can say the minimum length is 3, as we can find 2 intervals exactly have all numbers.

The main idea of this algorithm is to divide the array into many intervals, and each interval have exactly all distinctive numbers except the last interval, and we know such intervals that have all distinctive numbers can construct all the permutations. As the last interval missing some numbers, so we can construct a sequence never appeared in the array with the missing numbers. And the time complexity is O(N).

3.

a)

if n equal 1, then the expected number of locally maximal elements is 1, else if n >= 1, we should do the following things.

① For the corner element, its expectation to be local max is

sigma(**$1/n^2 * (k-1)/(n^2-1) * (k-2)/(n^2-2)$**, $1 <= k <= n^2$

k is the value of that corner element. And we have 4 corner elements, so the total expectation of the 4 corner elements to be local max is

sigma(**$4/n^2 * (k-1)/(n^2-1) * (k-2)/(n^2-2)$**), $1 <= k <= n^2$

② For the edge element(not include the 4 corner elements), its expectation to be local max is

sigma**$(1/n^2 * (k-1)/(n^2-1) * (k-2)/(n^2-2) * (k-3)/(n^2-3)$**, $1 <= k <= n^2$

k is the value of that edge element, and we have (4n-8) edge elements, so the total expectation of all edge elements to be local max is

sigma(**$(4n-8)/n^2 * (k-1)/(n^2-1) * (k-2)/(n^2-2) * (k-3)/(n^2-3))$**, $1 <= k <= n^2$

③ For the inner element, its expectation to be local max is

sigma**$(1/n^2 * (k-1)/(n^2-1) * (k-2)/(n^2-2) * (k-3)/(n^2-3) * (k-4)/(n^2-4))$**, $1 <= k <= n^2$

k is the value of that inner element, and we have (n^2-4n+4) inner elements, so the total expectation of all inner elements to be local max is

sigma(**$(n^2-4n+4)/n^2 * (k-1)/(n^2-1) * (k-2)/(n^2-2) * (k-3)/(n^2-3) * (k-4)/(n^2-4))$**, $1<= k <= n^2$

So for all elements, we have the following equation:

E_{corner} = sigma(**$4/n^2 * (k-1)/(n^2-1) * (k-2)/(n^2-2))$,** $1 <= k <= n^2$

E_{edge} = sigma(**$(4n-8)/n^2 * (k-1)/(n^2-1) * (k-2)/(n^2-2) * (k-3)/(n^2-3))$,** $1 <= k <= n^2$

E_{inner} = sigma(**$(n^2-4n+4)/n^2 * (k-1)/(n^2-1) * (k-2)/(n^2-2) * (k-3)/(n^2-3) *(k-4)/(n^2-4))$,**$1<=k<=n^2$

and the total expectation of all elements to be local max is

$E = E_{corner} + E_{edge} + E_{inner} = (3n^2 + 3n + 2)/15;$

b)

for one centered particular character, the length of palindrome and its probability is as following equation:

$p_1 = ½$, $length_1 = 1$

$p_2 = ½ * ½$, $length_2 = 3$

$p_3 = ½ * ½ * ½$, $length_3 = 5$

….....

$p_n = (½)^n$, $length_n = 2n-1$

so the expectation is $E = p_1*lenght_1 + p_2*length_2 + p_3*length_3 +...+ p_n*length_n$

$= ½ * 1 + (½)^2*3 + (½)^3*5 +...+ (½)^n*(2n-1)$

and $E – E/2 = ½*1 + (½)^2*3 + (½)^3*5 +...+ (½)^n*(2n-1)$ -

$$((½)^2*1 + (½)^3*3 +...+ (½)^n*(2n-3) + (½)^{n+1}*(2n-1))$$

$$= ½*1 + ½ + (½)^2 +...+ (½)^{n-1} - (½)^{n+1}*(2n-1)$$

$$= ½*1 – (½)^{n+1}*(2n-1) + (½ * (1- (½)^{n-1}))/(1- ½)$$

$$= ½ - (½)^{n+1}*(2n-1) + 1 – 1/2^{n-1}$$

$$= 3/2 - (½)^{n+1}*(2n-1) – 1/2^{n-1}$$

while n is infinite, we can say E – E/2 = 3/2, that is E = 3. So the expectation length is 3.