Algorithm Design and Analysis: Homework 2

Due on October 21, 2017 at 9:00am

14784547 luochenqi

Luochenqi

Problem 1

(a).You have already bought your i^{th} toy and you have exactly j toy in your collection,the possible case in previous buying are as follows:
1).after buying the $(i-1)^{th}$ toy,you have already have j toys .
2).after buying the $(i-1)^{th}$ you only have j-1 toy so you need buy a toy in i^{th} round.
As a result,

$$p_{i,j} = p_{i-1,j} * \frac{j}{n} + p_{i-1,j-1} * \frac{n-j+1}{n}$$

for $i \geq 2$ and $1 \leq j \leq n$,with $j \leq i$
$p_{i,j} = 0$for $j > i$
(b)we know $p_{i,0}$ is the possibility that you got nothing after i rounds ,so it is impossible ,so $p_{i,0} = 0$,starting from $p_{1,1} = 1$,we get

$$p_{2,1} = p_{1,1}\frac{1}{n}$$

$$p_{2,2} = p_{1,1}\frac{n-1}{n}$$

with $p_{2,2}$ and $p_{2,1}$ we can get $p_{3,1}$ and so on.
Likewise, given pk,m for m = 1,2,...,k, we could calculate the values of $p_{k+1,m}$ for m = 1,2,...,k+1, using the recursive equations.

Problem 2

Solution
Since the w_i and W are real numbers ,we can not put them into the array index.
In the same time we notice that $n > v_i$ it means $\sum v_i$ is O(n^2).

```
for (int i = 1; i <=n; i++)
{
    for (int j = 1; j <p; j++)
    {
        //f array means the weight of current items and j value
        // we find the minimum of the weight
        if (f[i - 1][j - v[i-1]] + w[i-1]<f[i - 1][j])
            f[i][j]=f[i - 1][j - v[i-1]] + w[i-1];
        if (f[i - 1][j - v[i-1]] + w[i-1]>=f[i - 1][j])
            f[i][j]=f[i - 1][j];

    }
}
int maxm=W;
int maxv=0;
for (int i = 1; i <=n; i++)
{
    for (int j = p-1; j >=1; j--)
    {
    //find the weight smaller than W and find the maximum value.
        if ((f[i][j]<=maxm) &&(j>=maxv))
            maxv=j;

```

```
    }
}
cout<<maxv;
return 0;
```

we can see the two part is both n^3. So it is

$$O(n^3)$$

Problem 3

Counting friends
using quicksort to sort two array in the descending order and for every first element in X,find the order i in Y,counter add i-1.and delete this element in both array and loop.
The time complexity is $O(nlogn)$

```
    int n = ;
    int x[n];
    int y[n];
    count = 0;
    quicksort(x, 1, n);
    quicksort(y, 1, n);//O(nlogn)

    for (i = 1..n) //O(nlogn)
    {
        for (p in y)
        {
            // binarysearch
            // find index i in y and counting it into variable count
            find p == x[i]
            delete this element in both array
        }
    }
```

Problem 4

XOR Convolution
as we define $x^i x^j = x^{i \oplus j}$
$A(x) = a_0 + a_1 x + a_2 x^2 ... + a_n^n$
$B(x) = b_0 + b_1 x + b_2 x^2 ... + b_n^n$
$A(x) \times B(x) = AAN/2B * BN/2 + AB$
we divide them into two pieces,so we calculate n in each time that we have $log_2 n$
so it is O(nlogn)

Problem 5

DNA Pattern Recognition
Solution

simply,we use straightforward searching method ,the time complexity is $(n-m)*m = O(n^{\frac{3}{2}})$,is bigger than $O(nlogn)$
so we use FFT to make $a['A'][i]\&b['A'][len(b)-i]$ and to check if it is matched. its complexity is $O(nlogn)$

Problem 6

2D Inversions
(1) if we do not use divide method. it is $O(n^2)$ because we can use two loop

```
for (i=1..n)
    for (j=1..n){
        if x[i]>x[j] && y[i]>y[j] && i<j {
        count++;
        }
}
```

if list has one element
return (0,l)
divide two list into two halveS A and B
$(r_A, A) = sort - and - count(A)$
$(r_B, B) = sort - and - count(B)$
$(r_AB, L) = merge - and - count(A, B)$
return $(r_A + r_B + r_AB, L)$

$$T(n) = \begin{cases} O(1) & \text{n=1} \\ T(n/2) + T(n/2) + O(n) & \text{otherwise} \end{cases}$$

so

$$T(n) = O(nlogn)$$

(2) if list has one element
return (0,l)
divide two list into two halveS A and B
$(r_A, A) = sort - and - count(A)$
$(r_B, B) = sort - and - count(B)$
$(r_AB, L) = merge - and - count(A, B)$
return $(r_A + r_B + r_AB, L)$ $(r_A, A) = sort - and - count(A)$
$(r_B, B) = sort - and - count(B)$
$(r_AB, L) = merge - and - count(A, B)$

$$T(n) = \begin{cases} O(1) & \text{n=1} \\ T(n/2) + T(n/2) + O(n) & \text{otherwise} \end{cases}$$

so

$$T(n) = O(nlogn)$$

(3)search points in left hand T(n/2)
search points in right hand T(n/2)

sort remaining point by ycoordinate O(nlogn)

$$T(n) = \begin{cases} O(1) & \text{n=1} \\ T(n/2) + T(n/2) + O(nlogn) & \text{otherwise} \end{cases}$$

so

$$T(n) = O(nlog^2 n)$$