2016级算法第一次上机解题报告

王佳琪 16211002

1. **个人总结**

本次上机考察了动态规划，还有之前学过的归并、快排、队列，以及一些数学上的问题比如错排排列组合和简单的数论问题。

这次的很多题中都体现了两个字：“变通”，比如动态规划的C题，归并查找玲珑数的G题等题，虽然知识或者方法是我们都知道的，但是要想做对，需要一些思想上的变通，归根结底，还是需要真正地理解，才能应对基本问题之外的问题。所以感觉我自己动态规划还需要多练习，还有分治思想也需要好好体会，感觉能用在很多情境下，也很巧妙。快排的E题考的是快排中很细致的过程，这需要很好地理解一些基本方法，不只是会用。这道题不仅让以前快排有点没想清的我彻底明白了原理和每一步骤，而且给了我很好的提示，以后应当先在脑中想清楚每一步，再开始做题。

1. **解题报告**

# 题目代号：[A] The stupid owls

**思路分析:**

注意到题中所说的每只猫头鹰恰好都把信送错以及题目下面的hint，可以知道这是一道考错排的题。要计算的是每个猫头鹰数目错排的概率，可以先计算出每种猫头鹰只数对应的错排的种数，然后除以所有可能情况的种数，即为错排的概率。应注意到的一点是：从7之后的猫头鹰只数，错排的概率在百分号前保留两位小数之后是不变的！这样就可以避免在计算可能情况种数和错排种数时，数字会大到溢出long long的范围。

错排种数递推公式为：

D(n)=(n-1)(D(n-1)+D(n-2)) ，D(1)=0,D(2)=1；

递推公式的推导是按照下面的思路：

n个不同的元素的一个错排公式可以分作两步完成：

第一步：假设我们错排第一个元素，那么它可以从2~n的位置任意选择其中的一个，一共是有n-1种选择。

第二步：错排其余n-1个元素，也是需要分情况和种类的。因为这需要看第一步的结果，如果第一个元素落在第k个位置上，第二步就需要把k号元素进行错排，k号元素错排位置的不同将导致不同的情况会发生：

1.假设k号元素正好落在了第一个元素的位置，那么就可以将第一个元素和第k个元素完全剔除出去，因为相当于只是他们两者互换了位置，其他元素暂时还没有发生变动。留下来的n-2元素进行错排的话，那么我们就可以得到了D(n-2)种 的错排方式。

2.若k号元素不排到第一个元素的位置，我们可以暂时将现在排在k号位置的第一个元素剔除出去，生下来的就只包含k号元素和原来n-2个的元素了。这时，我们可以将原来的第一个元素的位置看做是现在k号元素的原本位置，因为k号元素不能够放在原来的位置上，所以就相当于是原来的n-2个元素和k共计n-1个元素进行完全的错排。那么一共就有D(n-1)种方法。

数目为n的时候情况总数为n!,故错排的概率就是

P(n)=D(n)/n!

**算法分析：**

本题是算法中的错排问题，递推问题。

每个错排均需遍历到，因此基本复杂度为 IMG_256 递归函数其中有n次循环，若不考虑错排规则，则每次循环均调用递归函数，考虑错排规则时调用次数满足 IMG_257 ，因此 ：

IMG_258

**易错点分析：**

1.本题易错点大概在于英语。。实际只要关注关键部分——猫头鹰给所有人全部送错，即可知道是一道考错排的题。

2.还有一个注意点就是可以想到从7之后的猫头鹰只数，错排的概率在百分号前保留两位小数之后是不变的，这样会比较好处理数目比较大的情况。

3.注意计算时采用double型浮点数。

**代码实现：**

#include<iostream>

#include<stdio.h>

using namespace std;

int main()

{

int n;

double num[31];

long long wh[31];

double ans[31];

ans[1] = 0;

num[1]=0;num[2]=1;

wh[1]=1;wh[2]=2;

for(int i = 3;i<30;i++)

{

num[i] = (i-1)\*(num[i-1]+num[i-2]);

}

for(int i=3;i<=12;i++)

wh[i] = wh[i-1]\*i;

for(int i=2;i<=12;i++)

ans[i] = num[i]/(double)wh[i];

for(int i =11;i<=30;i++) ans[i]=ans[12];

while(cin>>n)

{

printf("%.2f",ans[n]\*100);

printf("%%");

printf("\n");

}

}

# 题目代号：[B] ModricWang和数论

**思路分析:**

可以对除数b进行分类讨论：

1. b小于a，那么存在两种情况，b小于(a+1)/2和b大于(a+1)/2，但是b小于(a+1)/2时，存在b的倍数nb，位于(a+1)/2和a之间，a%b =a%nb，所以可以只考虑(a+1)/2~a之间的数，余数可能有（a-(a+1)/2）种。
2. b大于a，那么余数就是a一种。

故可得知，对于所有的正整数b，a%b可能有（a+1）/2种不同的值。

**易错点分析：**

这道题易出现的错误点是容易误以为a%b可能的不同的值的情况是a种。

**代码实现：**

#include<iostream>

using namespace std;

int main()

{

long long num;

cin>>num;

cout<<(num+1)/2+1<<endl;

}

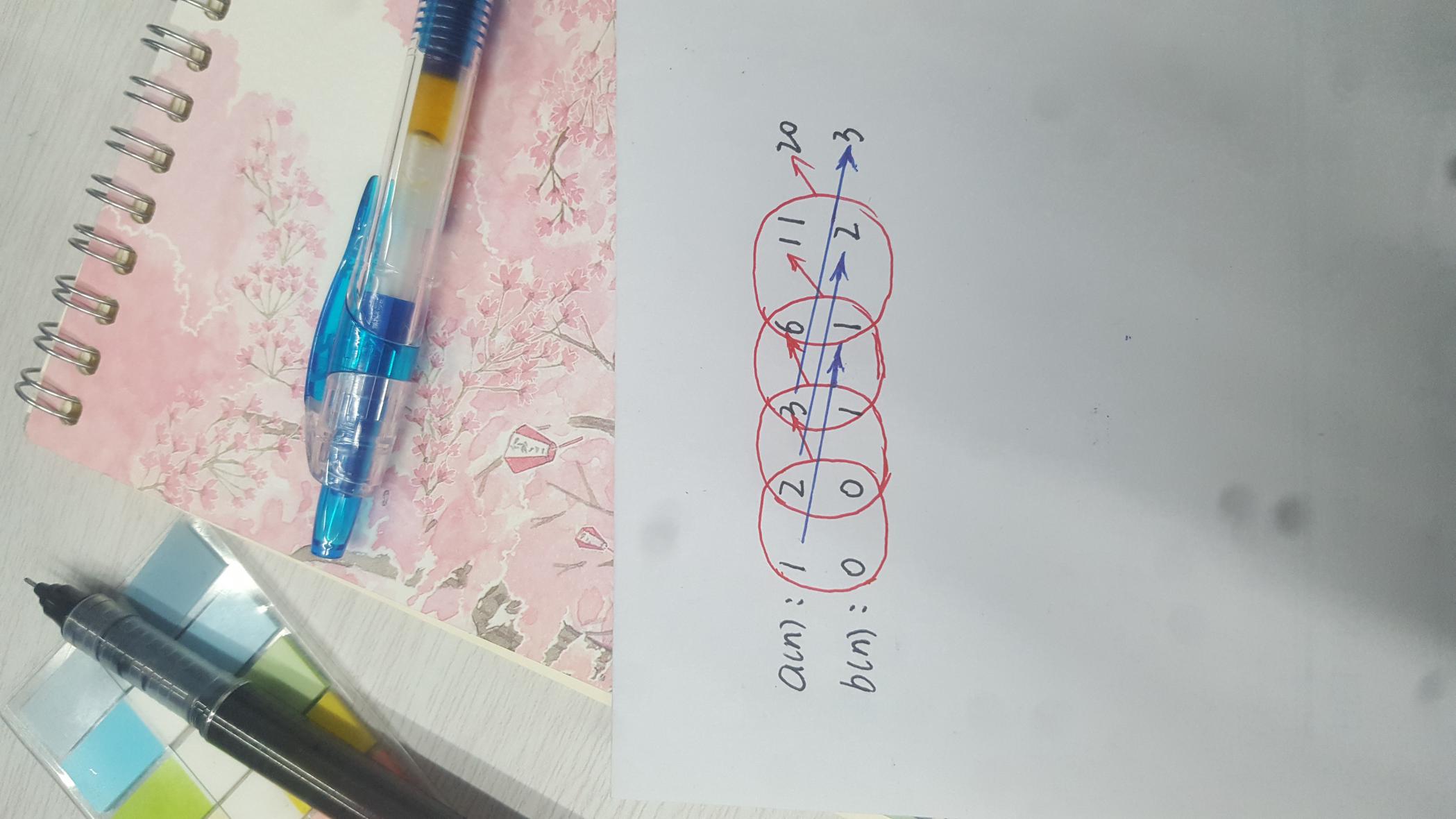
# 题目代号：[C] AlvinZH去图书馆

**思路分析:**

这道题是一道典型的动态规划问题，是由动态规划问题中基本的上楼梯问题发展而来，基本思路是通过拆分问题，定义问题状态和状态之间的关系，使得问题能够以递推（或者说分治）的方式去解决。

本题的解决方法，注意到由 走1或2步 和 走3步 的要求不同可以想到将这两种情况分开考虑，同时实现题目中的限制条件。是把前一步上1或2级到达第n级  和  前一步上3级到达第n级  这两类方法种数看成两个彼此有关联的数列，（因为第二种方法有限定条件：不可连续上两次三级）通过递推公式可以将大问题拆分成小问题，由小问题归出大问题的结果。

具体分析如下：



如图，上面的是a数列  下面的是b数列

a数列：通过前一次上1或2级到达现在的楼梯  
b数列：通过前一次上3级到达现在的楼梯

下面分析：

a数列是通过前一次上1或2级到达现在的楼梯，b数列是通过前一次上3级到达现在的楼梯，到达第n级总共的方法数就是p(n)=a[n]+b[n]种。

想要到达第n级，有两种方法：

1. 可以从第n-2级上两级或从第n-1级上1级，所以有：

a[n]= p(n-1)+p(n-2)

(其中p(n-1),p(n-2)为到达n-1和n-2级方法种数)

即：a[n]=a[n-1]+b[n-1]+a[n-2]+b[n-2]  
2. 也可以通过上了三级楼梯达到第n级楼梯，这种情况下，通过前一步上三级到达第n级的种数是：

b[n]=a[n-3]

因为之前在n-3级楼梯的时候，a[n-3]代表从第n-3-2级上两级到达或者从第n-3-1级上一级到达，可以保证 上三级的前一步不是上三级（也就是不是连着上了三级两次）符合了题目要求中的：不能连续两次一步上三级。  
所以使b[n]=a[n-3]，故到达第n级总共的方法数就是p(n)=a[n]+b[n]

a[n]与b[n]通过上面的递推公式可以得到。

一些基本的动态规划的问题总结：

<http://blog.csdn.net/u010398493/article/details/52809957>

**算法分析：**

本题考察的是动态规划，递推问题。时间复杂度：O(n)

**易错点分析**：

“甚至可以直接跨过两个石砖，到达第四块石砖，但是不能连续两次这种操作” 这个题目条件的应用是难点。如何限制不使其连续两次走三步。由走1或2步 和 走3步 的要求不同也可以想到将这两种情况分开考虑，同时实现限制条件。

**代码实现：**

#include<bits/stdc++.h>

using namespace std;

long long a[106],b[106];

main()

{

a[1] =1;

a[2] = 2;

a[3] = 3;

b[1] = 0;

b[2] = 0;

b[3] = 1;

for(int i=4; i<=52; i++)

{

a[i] = a[i-1] + a[i-2] + b[i-1] + b[i-2];

b[i] = a[i-3];

}

int n;

while(cin>>n)

cout<<a[n]+b[n]<<endl;

}

# 题目代号：[D] 水水的Horner Rule

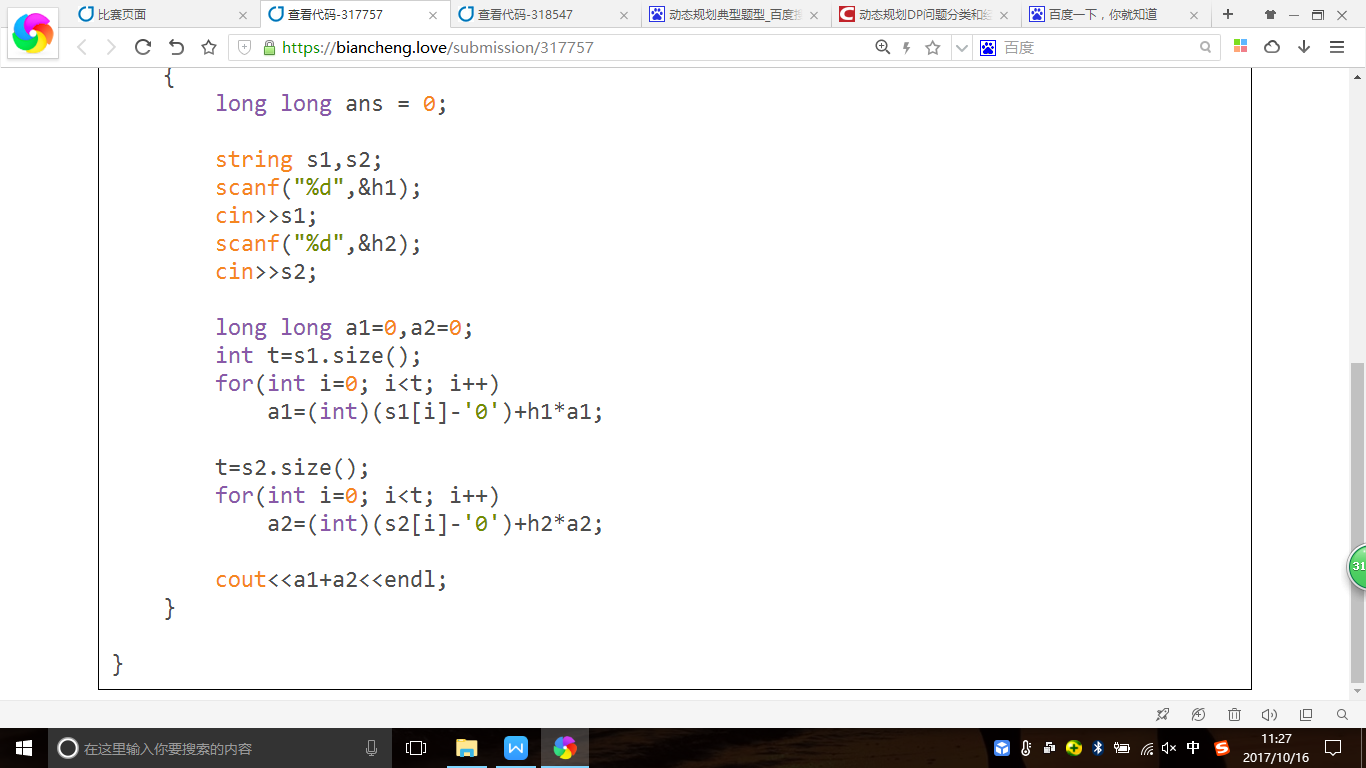
**思路分析:**

个人认为本题的考察点有两点：

1. 对于题中所提到的霍纳（Horner）规则的应用，进行进制间的转换。

2. 高精度问题 由于低进制数位数会很多，如何处理比较大的数据也是一个考点。

第一点的解决思路就是利用题中的公式，从高位到低位逐渐加上位数乘权值，得到结果。



第二点可以利用string变量类型来解决，用string变量输入要转化的数，然后逐位处理。

**算法分析：**

运用循环，不断加每一位的数字\*权值，时间复杂度：O(n)。

**易错点分析：**

注意低进制数位数很多，应使用string输入。采用高精度。

**代码实现：**

#include<iostream>

#include<cstdio>

#include<string>

using namespace std;

int main()

{

int n,h1,h2;

cin>>n;

for(int i=0; i<n; i++)

{

long long ans = 0;

string s1,s2;

scanf("%d",&h1);

cin>>s1;

scanf("%d",&h2);

cin>>s2;

long long a1=0,a2=0;

int t=s1.size();

for(int i=0; i<t; i++)

a1=(int)(s1[i]-'0')+h1\*a1;

t=s2.size();

for(int i=0; i<t; i++)

a2=(int)(s2[i]-'0')+h2\*a2;

cout<<a1+a2<<endl;

}

}

# 题目代号：[E] ModricWang's QuickSort

**思路分析:**

正如题目说的，本题考的就是快排，但是不同于以往的快排题考的比较宏观，是整体的排序，这道题考的是，相当于是前两次的过程，考的是快排内部比较基本的思路和操作。 如果对于快排这种方法不能完全想透的话，就容易出一些细枝末节的错误，很难把它完全做对。这也是本题的目的所在（我想。。）让做题者更好地理解快排的每一个步骤，深入理解算法。

可以通过只模拟前两步的方法，只进行快排中前两步的排序，然后输出第二步中第二部分的数据。

做完这道题我自身感觉收获还是很大的，因为之前虽然会用快排，但是在快排的某些细微的步骤其实并不能完全想清楚，做完之后，感觉现在理解的明显更深入了。

**算法分析：**

本题考察的是快排的前两步,只进行前两步的操作时间复杂度为O(n)。

完整快排时间复杂度：O(n^2).

**易错点分析：**

1. 注意中点mid位置的选取。

2. 注意使用题中的快排步骤，观察样例的分组方法。

**代码实现：**

#include<iostream>

using namespace std;

int A[1000005];

int main()

{

int n;

cin>>n;

for(int i=0;i<n;i++) cin>>A[i];

int mid=A[n/2];

int low = 0,high=n-1;

while(low<high){

while(low<high && A[low]<=mid)

low++;

while(low<high && A[high]>mid)

high--;

int s=A[low];

A[low]=A[high];

A[high]=s;

}

high = high-1;

//cout<<high<<endl;

int h = high;

low = 0;

mid = A[(high+1)/2];

while(low<high){

while(low<high && A[low]<=mid)

low++;

while(low<high && A[high]>mid)

high--;

int s=A[low];

A[low]=A[high];

A[high]=s;

}

for(int i=high;i<=h;i++)

cout<<A[i]<<" ";

cout<<endl;

}

# 题目代号：[F] AlvinZH的儿时梦想——木匠篇

**思路分析:**

题目很复杂，题意很简单。主要是不认真读题会带来很多错点，一定要多读几遍题再做题！

处理本题的方法有两个：

1. 暴力法 这样的话时间复杂度会比较高
2. 单调队列

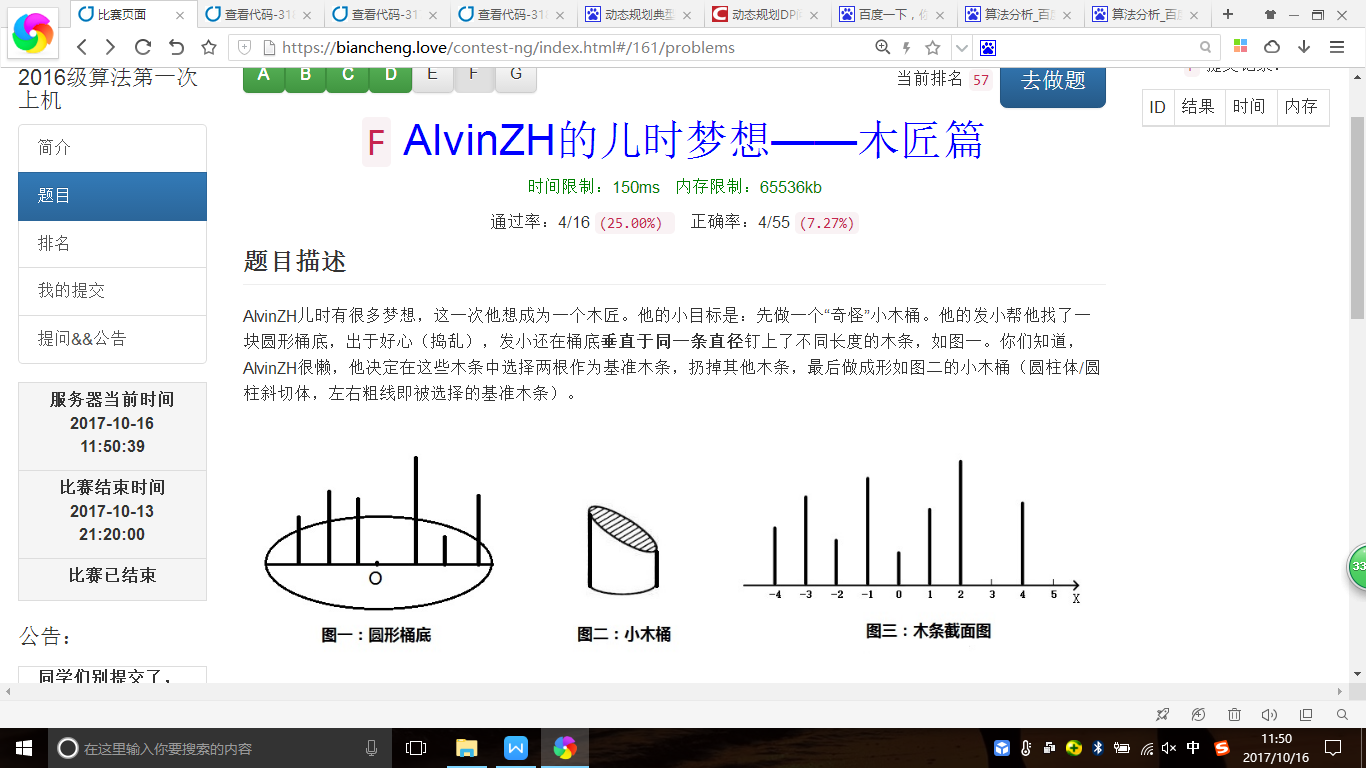
**算法分析：**

1. 暴力法，两循环嵌套时间复杂度 O(n^2)
2. 单调队列优化之后：最坏情况：O(n^2) 最好情况：O(n)

**易错点分析：**

1. 注意解题方法合理，否则容易超时（不过似乎由于数据比较小，暴力解也没问题。。）
2. 注意仔细读题（！！！） 因为没读对题造成的WA往往是最难过的。。。

“**必须从圆心两边各选一根木条，圆心处木条可看作属于两边**”确定了木条的选取方法和圆心处的怎么处理。

“**木桶不可倾斜**”说明了木桶装水的量是要看最短木条，而不能这样装水！

**代码实现：**

#include<stdio.h>

#include <math.h>

struct wood{

double length;

double pos;

};

struct wood leftt[105],rightt[105];

int main()

{

int n,l,r;

double len,mark,mid,V,high,loc,d;

double pi=acos(-1.0);

while(scanf("%d",&n)!=EOF)

{

l = r = 0;

mark = 0;

mid = 0;

for(int i = 1;i<=n;i++)

{

scanf("%lf%lf",&loc,&len);

if(loc<=0){

l++;

leftt[l].pos = loc;

leftt[l].length = len;

}

if(loc>=0){

r++;

rightt[r].pos = loc;

rightt[r].length = len;

}

}

for(int i =1;i<=r;i++)

for(int j = 1;j<=l;j++)

{

if(rightt[i].pos == leftt[j].pos) continue;

if(rightt[i].length<leftt[j].length)

high = rightt[i].length;

else high = leftt[j].length;

d = rightt[i].pos-leftt[j].pos;

mid = high\*d\*d;

if(mid>mark) mark = mid;

}

V = mark/4\*pi;

printf("%.3lf\n",V);

}

}

# 题目代号：[G] D&C--玲珑数

**思路分析:**

要找满足i<j且a[i]>2\*a[j]的数对，乍一看很像是之前的寻找逆序数的套路。但仔细一看做的时候有些地方还是很有差别。

使用归并排序计数，因为所谓归并就是将大问题转化为一个一个小的分问题解决，再一步一步归并回大问题。归并的同时进行玲珑数对的计数，首先从最少的两个数开始找，然后排序，每次计数的时候，中间数两边的数都已经按照升序有序 并且 两边的两组数都已经进行过统计，这样的话，既可以达到不重复，也可以数全，查找时，指向前面数的指针i不动，一旦j指向的数满足条件，就后移，直到不满足条件或超出范围，这样的话mid~j之间的数一定满足条件。

下面对比一下 寻找i<j且a[i]>a[j]的逆序数和寻找本题的玲珑数之间做题时的差异：（因为我在这里错了好久。。）

1. **void** Merge(**long** **long** r[],**long** **long** r1[],**int** s,**int** m,**int** t){
2. **int** i=s,j=m+1,k=s;
3. **while**(i<=m&&j<=t){
4. **if**(r[i]<=r[j])
5. r1[k++]=r[i++];
6. **else**{
7. r1[k++]=r[j++];
8. count+=(m-i+1);
9. }
10. }
11. **while**(i<=m)
12. r1[k++]=r[i++];
13. **while**(j<=t)
14. r1[k++]=r[j++];
15. **for**(i=s;i<=t;i++)
16. r[i]=r1[i];
17. }

上面的代码是原来进行归并查找逆序数时的代码，计数是在排序过程中进行。由于现在的题要求的是a[i]>2\*a[j]，如果仍然在原来的地方计数，可能会导致有些数不会被计入。而如果再在内部加一个循环的话会导致超时。最好的方法是在归并排序之前就进行计数：

void Merge(long long \*r,long long \*r1,int s,int m,int t){

int i=s,j=m+1,k=s;

for(i=s;i<=m;i++)

{

while(r[i]>2\*r[j]&&j<=t) j++;

count+=j-m-1;

}

i=s;j=m+1;

while(i<=m&&j<=t){

if(r[i]<=r[j]) r1[k++]=r[i++];

else r1[k++]=r[j++];

}

while(i<=m)

r1[k++]=r[i++];

while(j<=t)

r1[k++]=r[j++];

for(i=s;i<=t;i++)

r[i]=r1[i];

}

**算法分析：**

二分归并法，时间复杂度O(nlogn).

**易错点分析：**

1. 容易继续之前的找逆序数的套路，在不合理的地方进行计数，导致出现遗漏或者其他问题。
2. 在归并之前要先将范围内的数据保存在另一个数组里，在另一个数组里进行操作，如果直接在原来的数组里归并会导致数据顺序变化。

**代码实现：**

#include <stdio.h>

#define maxn 100005

long long count;

long long a[10005];

long long r1[maxn];

long long r2[maxn];

void Merge(long long \*r,long long \*r1,int s,int m,int t){

int i=s,j=m+1,k=s;

for(i=s;i<=m;i++)

{

while(r[i]>2\*r[j]&&j<=t) j++;

count+=j-m-1;

}

i=s;j=m+1;

while(i<=m&&j<=t){

if(r[i]<=r[j]) r1[k++]=r[i++];

else r1[k++]=r[j++];

}

while(i<=m)

r1[k++]=r[i++];

while(j<=t)

r1[k++]=r[j++];

for(i=s;i<=t;i++)

r[i]=r1[i];

}

int MergeSort(long long \*r,int s,int t){

int m;

if(s==t)

return -1;

else{

m=(s+t)/2;

MergeSort(r,s,m);

MergeSort(r,m+1,t);

Merge(r,r1,s,m,t);

}

}

int main(){

int n,i,t,p,q,num;

count = 0;

while(~scanf("%d",&n))

{

for(i=0;i<n;i++) scanf("%lld",&a[i]);

scanf("%d",&t);

for(int i=0;i<t;i++)

{

count=0;

scanf("%d%d",&p,&q);

if(p>=q) {

num = p-q;

for(int j=0;j<num+1;j++)

r2[j]=a[q+j];

MergeSort(r2,0,num);

}

else{

num = q-p;

for(int j=0;j<num+1;j++)

r2[j]=a[p+j];

MergeSort(r2,0,num);

}

printf("%lld\n",count);

}

}

}