POJ3258

题目描述（POJ3258）：跳房子游戏指从河中的一块石头跳到另一块石头，这发生在一条又长又直的河流中，从一块石头开始，到另一块石头结束。长度为 L（$1 \le L \le 10^9$），从开始到结束之间的石头数量为 N（$0 \le N \le 50\,000$），从每块石头到开始位置有一个整数距离 d_i（$0<d_i<L$）。

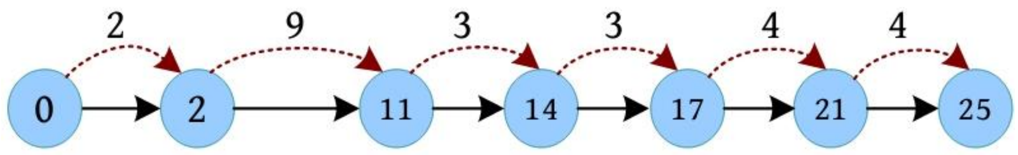
为了玩游戏，每头母牛都依次从起始石头开始，并尝试到达终点的石头，只能从石头跳到石头。当然，不那么灵活的母牛永远不会到达最后的石头，而是掉进河中。约翰计划移除几块石头，以增加母牛必须跳到最后的最短距离。不能删除起点和终点的石头，但约翰有足够的资源移除多达 M 块石头（$0 \le M \le N$）。请确定在移除 M 块石头后，母牛必须跳跃的最短距离的最大值。

输入：第 1 行包含 3 个整数 L、N 和 M。接下来的 N 行，每行都包含一个整数，表示从该石头到起始石头的距离。没有两块石头有相同的位置。

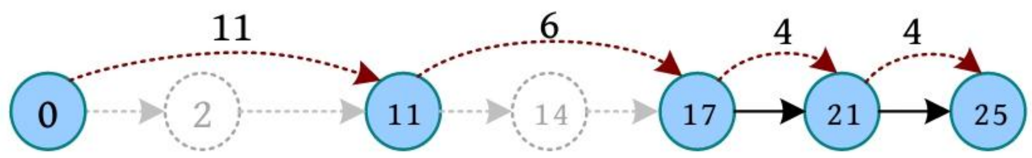
输出：单行输出移除 M 块石头后母牛必须跳跃的最短距离的最大值。

输入样例	输出样例
25 5 2 2 14 11 21 17	4

题解：根据输入样例，构建的图如下图所示。



在移除任何石头之前，跳跃的最短距离都是 2（从 0 到 2）。在移除 2 和 14 石头后，跳跃的最短距离是 4（从 17 到 21 或从 21 到 25）。



1. 算法设计

（1）如果移除的石头数等于总石头数（M=N），则直接输出 L。

（2）增加开始（0）和结束（N+l）两块石头，到开始节点的距离分别为 0 和 L。

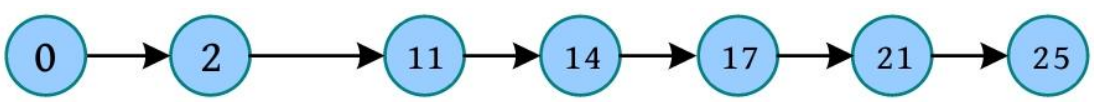
（3）对所有的石头都按照到开始节点的距离从小到大排序。

（4）令 left=0，right=L，如果 right-left>1，则 mid=(right+left)/2，判断是否满足移除 M 块石头之后，任意间距都不小于 mid。如果满足，则说明距离还可以更大，令 left=mid；否则令 right=mid，继续进行二分搜索。

（5）搜索结束后，left 就是母牛必须跳跃的最短距离的最大值。

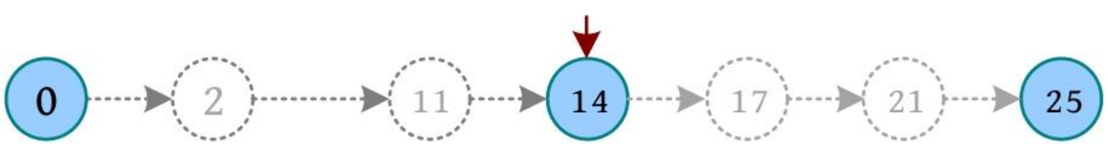
2. 图解

（1）根据输入样例，增加开始和结束两块石头，按照到开始节点的距离从小到大排序。



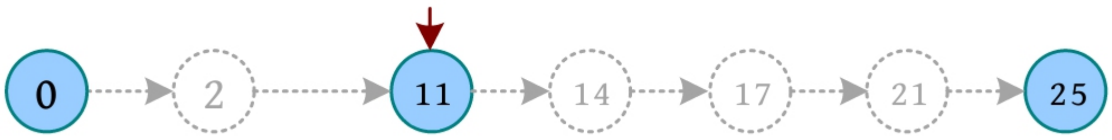
（2）令 left=0，right=L=25，right-left>1，mid=(right+left)/2=12，判断是否满足移除两块石头之后，任意间距都不小于 12。相当于将 3 块石头放置在开始位置和结束位置之间，且满足任意间距都不小于 12。

用 last 记录前一块已放置石头的下标，初始时 last=0，找第 1 个与 last 距离大于或等于 12 的位置，找到 14，放置第 1 块石头，更新 last=3。

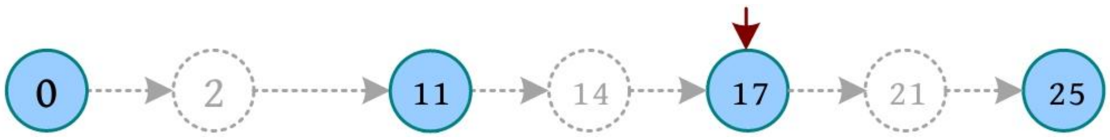


继续找第 1 个与 last 距离大于或等于 12 的位置，未找到，说明无法满足条件。缩小距离，令 right=mid=12，继续搜索。

（3）left=0，right=12，mid=(right+left)/2=6，判断是否满足移除两块石头之后，任意间距都不小于 6。初始时 last=0，找第 1 个与 last 距离大于或等于 6 的位置，找到 11，放置第 1 块石头，更新 last=2。



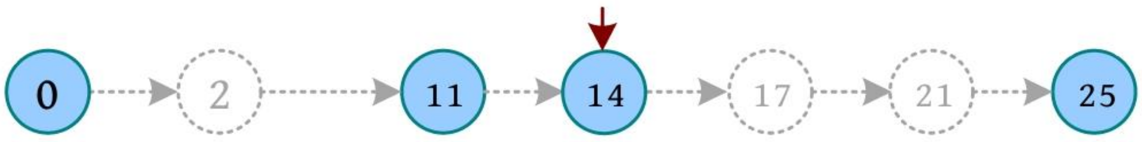
继续找第 1 个与 last 距离大于或等于 6 的位置，找到 17，放置第 2 块石头，更新 last=4。



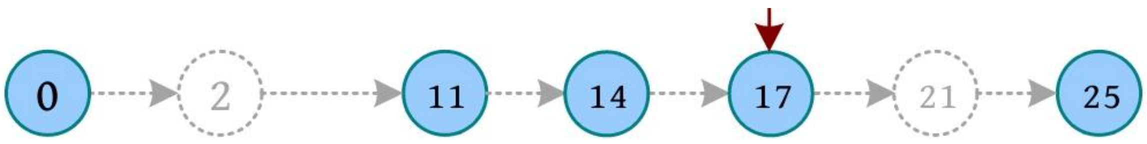
继续找第 1 个与 last 距离大于或等于 6 的位置，未找到，说明无法满足条件。缩小距离，令 right=mid=6，继续搜索。

（4）left=0，right=6，mid=(right+left)/2=3，判断是否满足移除两块石头之后，任意间距都不小于 3。初始时 last=0，找第 1 个与 last 距离大于或等于 3 的位置，找到 11，放置第 1 块石头，更新 last=2。

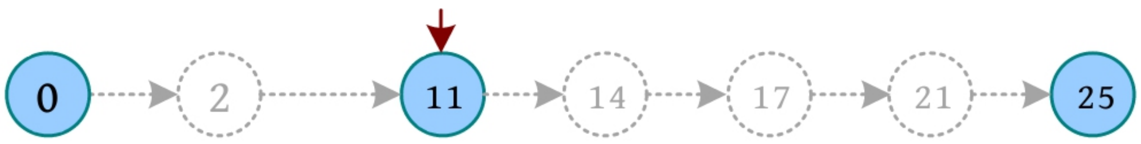
继续找第 1 个与 last 距离大于或等于 3 的位置，找到 14，放置第 2 块石头，更新 last=3。



继续找第 1 个与 last 距离大于或等于 3 的位置，找到 17，放置第 3 块石头，可以放置 3 块石头，满足条件。增加距离，令 left=mid=3，继续搜索。



（5）left=3，right=6，mid=(right+left)/2=4，判断是否满足移除两块石头之后，任意间距都不小于 4。初始时 last=0，找第 1 个与 last 距离大于或等于 4 的位置，找到 11，放置第 1 块石头，更新 last=2。



继续找第 1 个与 last 距离大于或等于 4 的位置，找到 17，放置第 2 块石头，更新 last=4。

继续找第 1 个与 last 距离大于或等于 4 的位置，找到 21，放置第 3 块石头，可以放置 3 块石头，满足条件。增加距离，令 left=mid=4，继续搜索。

（6）left=4，right=6，mid=(right+left)/2=5，判断是否满足移除两块石头之后，任意间距都不小于 5。初始时 last=0，找第 1 个与 last 距离大于或等于 5 的位置，找到 11，放置第 1 块石头，更新 last=2。

继续找第 1 个与 last 距离大于或等于 5 的位置，找到 17，放置第 2 块石头，更新 last=4。

继续找第 1 个与 last 距离大于或等于 5 的位置，未找到，说明无法满足条件。缩小距离，令 right=mid=5，继续搜索。

（7）left=4，right=5，此时 right-left=1，算法结束，输出答案 left=4。

3. 算法实现

判断函数相当于将 n-m 块石头放置在开始位置和结束位置之间，且任意间距都不小于 x。

```
bool judge(int x){ //使移除 m 块石头之后，任意间距都不小于 x
    int num=n-m; //减掉 m 块石头，放置 num 块石头，循环 num 次
    int last=0; //记录前一个已放置石头的下标
    for(int i=0;i<num;i++){ //对于这些石头，要使任意间距都不小于 x
        int cur=last+1;
        while(cur<=n&&dis[cur]-dis[last]<x) //放在第 1 个与 last 距离大于或等于 x 的位置
            cur++; //由 cur 累计位置
        if(cur>n)
            return 0; //如果在这个过程中大于 n 了，则说明放不开
        last=cur; //更新 last 位置
    }
    return 1;
}

int main(){
    cin>>L>>n>>m;
    if(n==m){
        cout<<L<<endl;
        return 0;
    }
    for(int i=1;i<=n;i++)
        cin>>dis[i];
    dis[0]=0;//增加开始点
    dis[n+1]=L;//增加结束点
    sort(dis,dis+n+2);
    int left=0,right=L;
    while(right-left>1){
        int mid=(right+left)/2;
        if(judge(mid))
            left=mid;//如果放得开，则说明 x 还可以更大
        else
            right=mid;
    }
    cout<<left<<endl;
    return 0;
}
```

POJ3104

题目描述（POJ3104）：可以使用散热器烘干衣服。但散热器很小，所以它一次只能容纳一件衣服。简有 n 件衣服，每件衣服在洗涤过程中都带有 a_i 的水。在自然风干的情况下，每件衣服的含水量每分钟减少 1（只有当物品还没有完全干燥时）。当含水量变为零时，布料变干并准备好包装。在散热器上烘干时，衣服的含水量每分钟减少 k（如果衣服含有少于 k 的水，则衣服的含水量变为零）。请有效地使用散热器来最小化烘干的总时间。

输入：第 1 行包含一个整数 n（1≤n≤105）；第 2 行包含 a_i（$1≤a_i≤10^9$，1≤i≤n）；第 3 行包含 k（$1≤k≤10^9$）。

输出：单行输出烘干所有衣服所需的最少时间。

输入样例	输出样例
3 2 3 9 5 3 2 3 6 5	3 2

题解：假设烘干所有衣服所需的最少时间为 mid，如果所有衣服的含水量 a[i]都小于 mid，则不需要用烘干机，自然风干的时间也不会超过 mid。如果有的衣服 a[i]大于 mid，则让所有 a[i]大于 mid 的衣服使用烘干机，让 a[i]不大于 mid 的衣服自然风干即可。

假设衣服 a[i]>mid，用了 t 时间的烘干机，对剩余的时间 mid-t 选择自然风干，那么 a[i]=k×t+mid-t，t=(a[i]-mid)/(k-1)。只需判断这些 a[i]大于 mid 的衣服使用烘干机的总时间有没有超过 mid，如果超过，则不满足条件。

1. 算法设计

（1）按照 a[i]从小到大排序。

（2）如果 k=1，则直接输出 a[n-1]，算法结束。

（3）进行二分搜索，l=1，r=a[n-1]，mid=(l+r)>>1，判断最少烘干时间为 mid 是否可行，如果可行，则 r=mid-1，减少时间继续搜索；否则 l=mid+1，增加时间继续搜索。当 l>r 时停止。

（4）判断最少烘干时间为 mid 是否可行。对所有 a[i]>mid 的衣服使用烘干机，用 sum 累加使用烘干机的时间，如果 sum>mid，则说明不可行，返回 0。当所有衣服都处理完毕时，返回 1。

要特别注意以下事项:

（1）对 t 的结果需要向上取整，因为如果有余数，再用一次烘干机无非就是多 1 个时间，但是如果自然风干，则至少用 1 个时间。

（2）公式中的分母是 k-1，因此在 k=1 时需要单独判断特殊情况，直接输出最大的含水量即可，不然会超时。

2. 算法实现

```
int judge(int x){
    int sum=0;
    for(int i=0;i<n;i++){
        if(a[i]>x)
            sum+=(a[i]-x+k-2)/(k-1);//向上取整，或 ceil((a[i]-x)*1.0/(k-1));
        if(sum>x)
            return 0;
    }
    return 1;
}
```

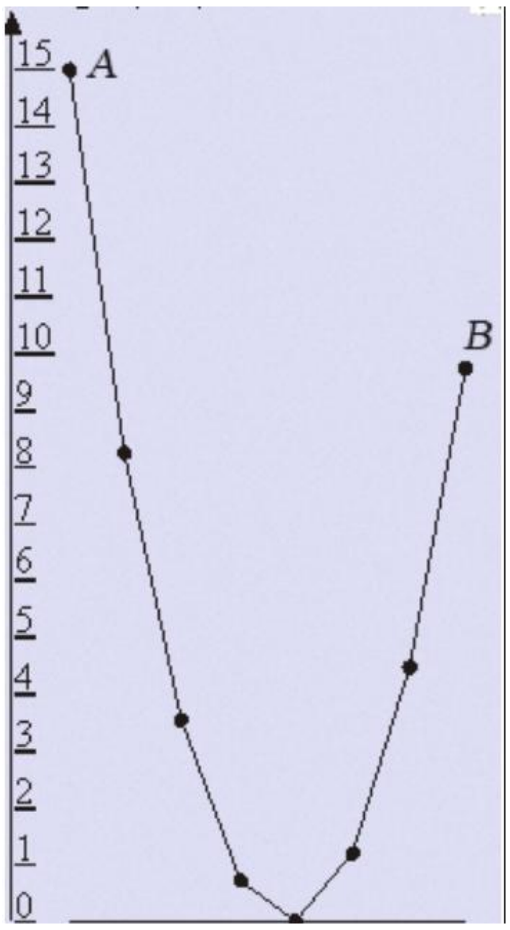
```
void solve(){
    int l=1,r=a[n-1],ans;
    while(l<=r){
        int mid=(l+r)>>1;
        if(judge(mid)){
            ans=mid;
            r=mid-1;//减小
        }
        else
            l=mid+1;//增加
    }
    cout<<ans<<endl;
}

int main(){
    while(~scanf("%d",&n)){
        for(int i=0;i<n;i++)
            scanf("%d",&a[i]);
        scanf("%d",&k);
        sort(a,a+n);
        if(k==1){
            printf("%d\n",a[n-1]);
            continue;
        }
        solve();
    }
    return 0;
}
```

POJ1759

题目描述（POJ1759）：新年花环由 N 个灯组成，每个灯都悬挂在比两个相邻灯的平均高度低 1 毫米的高度处。最左边的灯挂在地面以上 A 毫米的高度处。必须确定最右侧灯的最低高度 B，以便花环中的灯不会落在地面上，尽管其中一些灯可能会接触地面。灯的编号为 1～N，并以毫米为单位表示第 i 个灯的高度为 H_i，推导出以下等式：$H_1=A$；$H_i=(H_{i-1}+H_{i+1})/2-1$，$1<i<N$；$H_N=B$；$H_i\geq 0$，$1\leq i\leq N$。

下图中所示的具有 8 个灯的花环，A=15 和 B=9.75。



输入：输入包含两个数字 N 和 A。N（$3\leq N\leq 1000$）表示花环中灯的数量，A（$10\leq A\leq 1000$）表示地面上最左边的灯的高度（实数，以毫米为单位）。

输出：单行输出 B，精确到小数点右边两位数，表示最右边灯的最低可能高度。

输入样例	输出样例
692 532.81	446113.34

题解：

1. 算法设计

根据高度公式 $H_i=(H_{i-1}+H_{i+1})/2-1$，整理该公式得到 $H_{i+1}=2\times H_i-H_{i-1}+2$，也可以将其写成当前项与前面两项的关系表达式：$H_i=2\times H_{i-1}-H_{i-2}+2$。

（1）二分搜索。初始时，num[1]=A，l=0.0，r=inf（无穷大，通常设为 0x3f3f3f3f），mid=(l+r)/2。判断第 2 个灯的高度为 mid 是否可行，如果可行，则令 r=mid，缩小高度搜索；否则 l=mid，增加高度搜索。

（2）判断 mid 是否可行。令 num[2]=mid，根据公式从左向右推导，num[i]=2×num[i-1]- num[i-2]+2，i=3…n。如果在推导过程中 num[i]<eps，则说明不可行，返回 false。**注意不要写小于 0，否则由于精度问题会出错。**eps 是一个较小的数，例如 1e-7。

（3）可以用 r-l>eps 判断循环条件，也可以搜索到较大的次数时停止，例如 100 次，运行 100 次二分搜索可以达到 10^{-30} 的精度范围。实际上对于输入样例，运行 43 次已经找到答案，为保险起见，尽量执行较多的次数，时间相差不大。

2. 算法实现

```
bool check(double mid){//判断第 2 个灯的高度为 mid 是否可行
   num[2]=mid;
   for(int i=3;i<=n;i++){
      num[i]=2*num[i-1]-num[i-2]+2;
      if(num[i]<eps) return false; //写小于 0，由于精度问题会出错
   }
   ans=num[n];
   return true;
}
void solve(){
   num[1]=A;
   double l=0.0;
   double r=A;//inf
   while(r-l>eps){//for(int i=0;i<100;i++)
      double mid=(l+r)/2;
      if(check(mid))
         r=mid;
      else
         l=mid;
      }
   }
```

POJ1064

题目描述（POJ1064）：有 N 条电缆，长度分别为 L_i，如何从它们中切割出 K 条长度相同的电缆，每条电缆最长有多少米。

输入：输入的第 1 行包含两个整数 N 和 K（1≤N,K≤10 000）。N 是电缆的数量，K 是要求切割的数量。后面是 N 行，每行一个数字 L_i（$1 \le L_i \le 100\ 000$），表示每条电缆的长度。

输出：单行输出电缆切割的最大长度（在小数点后保留两位数字）。如果不能切割所要求数量的电缆，则输出“0.00”（不带引号）。

输入样例	输出样例
4 11 8.02 7.43 4.57 5.39	2.00

题解：本题求解切割出的 K 条电缆的最大可能长度，因为一条电缆有可能切割出多条，因此第 K 条的电缆长度并不是答案。可以假设最大长度为 x，采用二分搜索求解答案。

1. 算法设计

（1）二分搜索。初始时，l=0.0，r=inf，r 也可以被初始化为 N 条电缆中的最大长度。mid=(l+r)/2，判断切割出来电缆的长度为 mid，是否可以切割 K 条。如果可以，则令 l=mid，增加长度搜索，否则 r=mid，减少长度搜索。

（2）判断 mid 是否可行。枚举 N 条电缆，累加每条电缆可以切割出的数量，注意该数量要取整(int)(L[i]/mid)，如果数量大于或等于 K，则表示可行。

（3）可以用 r-l>eps 判断循环条件，也可以在搜索较大的次数时停止，例如 100 次。结束时返回 l。

（4）输出答案。本题要求保留两位小数，切割后不可四舍五入，因此可以扩大 100 倍取下限，然后缩小 100 倍，舍去 2 位小数之后的数字。但是存在特殊情况，例如 1.599 999 99，这样的数近似于 1.60，可以加上一个特别小的数处理该问题，因此返回答案 ans 加上 eps（1e-7）即可。还有一种解决办法是直接返回 r 作为答案，因为循环条件 r-l>eps，r 比 l 大 eps。

2. 算法实现

```
bool judge(double x){//假设切割出来的绳子的长度为x，判断够不够切割
    int num=0;
    for(int i=0;i<n;i++)
        num+=(int)(L[i]/x);
    return num>=k;
}
double solve(){
    double l=0;
    double r=*(max_element(L,L+n));//inf;
    while(r-l>eps){//for(int i=0;i<100;i++){
        double mid=(l+r)/2;
        if(judge(mid))
            l=mid;
        else
            r=mid;
    }
    return l;
}
int main(){
    while(~scanf("%d%d",&n,&k)){
        for(int i=0;i<n; i++)
            scanf("%lf",&L[i]);
        double ans=solve()+eps;
        printf("%.2lf\n", floor(ans*100)/100); //取下限
    }
    return 0;
}
```