题意： 给定一个长度为 N 的数组，我们可以对其中的数进行修改，每次修改有一个代价，对于一个数 x，修改成 y 之后的代价就是 abs（x-y）。

现在我们要将这个数列变成一个单调不减的，问所需要的代价。

其中 N<=5000,-10^9=<a[i]<=10^9

解决

solution 1

看到这个题目，很容易就想到了 DP，然后有了一个非常暴力的方程。

用 f[i][j]代表前 i 个数，最后一个数的高度为 j 所需要的最少代价。

很容易写出方程 f[i][j]=Min(f[i-1][k]+abs(j-k))

这个方程时间复杂度为 O（N*max*max）,其中 max 是最大的 a[i]，当然，对于负数的情况可以将整个数组向上平移变成正数。

可以看到的是，这个方程的时间复杂度实在太高了，在空间上也难以承受。所以我们只能另寻他法。

solution 2

有了上面的基础，很快有了优化的办法，那就是离散化！

既然总的数据量只有 2000，那么我们可以把每个数在数列中的 rank 作为这个数的大小

那么得到 f[i][j]代表前 i 个数，最后一个数为数列中第 j 大的数的代价。

方程可以写出 f[i][j]=Min(f[i-1][k]+abs(b[j]-a[i]))，其中 b[j]就是第 j 大数的大小。

时间复杂度 o（N^3），还是不能很好的解决问题,所以我们还要寻求其他的办法。

solution 3

然后我开始想这样一个问题，如果说 f[i][j]要大于 f[i][j-1],那么计算这样的 f[i][j]还有什么意义呢？

因为我第 i 个数得到的大小为 b[j-1]的消耗要比得到 b[j]的消耗少，（b 数列是单增的），那么在 i+1 个数的决策的时候，这样的 f[i][j]还有什么意义呢？！(大家可以自己证明一下)

所以我们必须还要压缩状态！

修改我们的方程，用 f[i][j],代表前 i 个数，第 i 个数小于等于 b[j]的最小花费

方程可以写成 f[i][j]=Min(f[i][j-1],f[i-1][j]+abs(a[i]-b[j]))

然后由于转移只是用到了相邻状态，可以用滚动数组来优化空间

即 f[i]=Min(f[i-1],f[i]+abs(a[i]-b[j]))

时间复杂度 O（N^2），空间复杂度 O（N）

这个问题还有更优秀的算法，在此不再做过多讲解，可以用左偏树，有兴趣的同学可以请教 zhazha 大神 。

题目第一次由于对数的大小估计不足最终 RE 了，一定要使用 long long

单调下降的情况只需要把数组翻转再求一次即可。

```
# include <cstring>
# include <cstdlib>
# include <cstdio>
# include <algorithm>
# include <iostream>
using namespace std;
# define N 2020
long long dp[N],a[N],b[N],c[N];
int n;
long long abss(long long x)
{
	if (x>0)
		return x;
	else return -x;
}
long long solve(long long a[])
{
	memset(dp,0,sizeof(dp));
	for (int i=1;i<=n;i++)
		for (int j=1;j<=n;j++)
		{
```

```
                dp[j]+=abss(b[j]-a[i]);
                if (j>1)
                    dp[j]=min(dp[j],dp[j-1]);
            }
        long long ans=dp[1];
        for (int i=2;i<=n;i++)
            ans=min(ans,dp[i]);
        return ans;
}
int main (void)
{
    cin>>n;
    for (int i=1;i<=n;i++)
    {
        scanf("%lld",&a[i]);
        c[n-i+1]=b[i]=a[i];
    }
    sort(b+1,b+1+n);
    long long ans=solve(a);
    ans=min(ans,solve(c));
    cout<<ans<<endl;
    return 0;
```

```
}
```