题意：给你一段区间，需要你求出（在这段区间之类的最小值\*这段区间所有元素之和）的最大值......

例如：

6

3 1 6 4 5 2

以4为最小值，向左右延伸，6 4 5  值为60.......

思路：解决完为这道题目，我才真正明白了单调栈的原理，它就是以某一个值为最小（最大）值，向这个值的两侧延伸，遇到大于它（小于它）的值，就将它延伸的范围扩大，当然，一般来说，要这样做的算法复杂度为o(n^2)，但是借助栈这个玩意，维护其单调增（减），就可以在o(n)的时间复杂度解决这个问题。将一元素加入栈时，先判断它是否大于(小于)栈顶元素，若是大于（小于）栈顶元素，加入栈。（从这里开始只讲维护单调增栈）否则，将栈顶元素出栈，直到栈顶元素小于要加入栈的元素，在此过程中，需要维护向前延伸和向后延伸的问题，当要加入栈的元素之前有n个栈元素出栈，那么说明这n个出栈的元素都会大于或者等于要入栈的元素，此时，我们需要维护入栈元素可以向前延伸多少个元素（相当于记录它的前面有多少个元素比它大），而每个栈顶元素要向出栈了的元素延伸，因为在出栈了的元素一定是比它的大的元素（根据我维护的是单调增栈）......这样，就在o(n)的时间复杂度内解决了上述问题.........

例如：3 1 6 4 5 2

（3，1，1）  （1，2，2）  （6，3，3）  （4，4，4）  （5，5，5）  （2，6，6）

首先每个元素自己本身的前后延伸都为1，把3加入栈，1<3，把3出栈，用1的前延伸加上3的前延伸，如此变为（1，1，2），6<1，入栈，变成（1，1，2）（6，3，3），

4<6，将6出栈，4向前延伸，1向后延伸变成（1，1，3） （4，3，4）

5>4，入栈，变成（1，1，3）（4，3，4）（5，5，5）

2<5，5出栈，2向前延伸，4向后延伸，变成（1，1，3）（4，3，5）                   2还未入栈（2，5，6）

2<4，4出栈，2向前延伸，1向后延伸，变成（1，1，5） （2，3，6）.....

一次类推，会发现最大的结果在（4，3，5）这里这意味着，以4为最小值的区间范围为3————5，也就是6 4 5

注意本题的ans一开始赋值不能为0，要为-1

因为最终的ans可能就是0，这样的话，ans不会被更新，下标还是一开始赋值的-1，-1

就错了

#include<cstdio>

#include<iostream>

using namespace std;

typedef long long ll;

const int INF=1000000007;

int a[100010],l[100010],r[100010];

ll sum[100010];

int main()

{

//freopen("input.txt","r",stdin);

int n;

scanf("%d",&n);

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

{

scanf("%d",&a[i]);

sum[i]=sum[i-1]+a[i];

l[i]=i;

r[i]=i;

}

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

{

while(l[i]>1 && a[l[i]-1]>=a[i])

l[i]=l[l[i]-1];

}

for(int i=n;i>=1;i--)

while(r[i]<n && a[r[i]+1]>=a[i])

r[i]=r[r[i]+1];

ll ans(-1);

int l1(-1),r1(-1);

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

{

if((sum[r[i]]-sum[l[i]-1])\*a[i]>ans)

{

ans=(sum[r[i]]-sum[l[i]-1])\*(a[i]);

l1=l[i];

r1=r[i];

}

}

printf("%lld\n%d %d\n",ans,l1,r1);

return 0;

}