B 锯钢条

时间限制：1000ms 内存限制：65536kb

通过率：161/184 (87.50%) 正确率：161/863 (18.66%)

题目描述

最近钢条厂的生意很红火，各种英寸的钢条供不应求。现在钢条厂接到北航一份大的订单，需要 n 根钢条，订单中第 i $(1 \leq i \leq n)$ 根钢条的长度是 l_i 英寸。

钢条厂有一种电锯，每开动一次可以把一根钢条锯成两段，锯出 n 根钢条就需要开动 $n-1$ 次电锯，但是每次电锯开动都需要电费等加工费用，而且加工费用是需要锯开的钢条长度的两倍，比如要把一段长 3 英寸的钢条锯开，就要收 6 元。不同的切割次序会产生不同的费用，请设计一个完成订单的钢条切割顺序方案，使得钢条厂付出的加工费用最少。

输入格式

第一行一个正整数 n $(1 \leq n \leq 2 \times 10^5)$，表示订单中需要钢条的个数。

接下来一行 n 个正整数 $l_1, l_2, \ldots, l_n$ $(1 \leq l_i \leq 5 \times 10^4)$，表示订单中每一根钢条需要的长度。

输出格式

一行一个正整数，表示钢条厂付出的最少加工费用。

输入样例

```
3
8 5 8
```

输出样例

```
68
```

样例解释

根据输入可知需要对总长度为 $8+5+8=21$ 英寸的钢条进行 2 次切割，一种最优的切割方案如下：

1. $21 \Rightarrow 8+13$，花费 42 元。
2. $13 \Rightarrow 5+8$，花费 26 元。

总花费 68 元，可以证明不存在更优的方案。

Hint

经典优先队列贪心问题。

优先队列其实本质和堆相似，但是相比于自己手写的堆而言优先队列维护更为方便简单（这也是为什么之前有关堆的题目都会限制 C++），而且将来学习图论也会遇到优先队列优化的 Dijkstra 算法。

C++ 中这些方便编程人员使用的标准模板都已经封装在 STL（标准模板库）中，包括动态变长顺序容器（vector），集合（set），哈希表（unordered_map）等等，感兴趣的同学可以更多了解，在将来的算法学习中 STL 和 `<algorithm>` 函数库将会给同学们提供莫大的的帮助！

按题意，要一根变 n 根肯定需要操作 n-1 次，每次的费用尽可能小，即每次操作的俩钢条总长度尽可能小，就是答案的最少费用

```
#include <iostream>
#include <queue>
using namespace std;
#define ll long long
ll n, temp1, temp2, ans;
priority_queue<ll, vector<ll>, greater<ll>> q;
  // 小根堆
int main(){
    cin >> n;
    for (int i = 1; i <= n; ++i){
        cin >> temp;
        q.push(temp);
    }

    while (l.size() > 1){
         temp1 = q.top();
        q.pop();
      temp2 = q.top();
      q.pop();
        ans += (temp1 + temp2) * 2;
        q.push(l.top() + temp);
    }
    cout << ans << "\n";
    return 0;
}
```

// 钢条据成好几段和把好几段钢条接起来算消耗是等价的

// 每次取出最小的两个合并后 push 回优先队列，最后队列中只剩一个元素表示已经连接好了

自己写优先队列（本题为小根堆）：

```
#include <iostream>
#include <queue>
using namespace std;
long long temp;
long long heap[1000000];
//n 的父节点对应 n/2,子节点为 2n 和 2n+1
long long ans;
int   n, size;
void swap(int* a, int* b) {
        int exchange = *a;
        *a = *b;
        *b = exchange;
    }
void insert(int x, int now) {
        heap[now] = x;
        adjustup(now);
          //维护小根堆
    }
    void pop() {
        heap[1] = heap[size--];
        adjustdown(1);
          //维护小根堆
    }
void adjustup(int now) {
        if (now == 1)
            return;
        if (heap[now] < heap[now / 2]) {
            swap(&heap[now], &heap[now / 2]);
            adjustup(now / 2);
        }
        else
            return;
    }
//最后一个元素与其父节点比较，若小则交换，继续向上比较
void adjustdown(int now) {
        int s = now * 2;
        if (s <= size) {
            if (s<size && heap[s]>heap[s + 1])
                s++;
```

```
            if (heap[now] > heap[s]) {
                swap(&heap[now], &heap[s]);
                adjustdown(s);
            }
        }
    }
//顶部的每次向下和子节点比较，若大则交换，继续向下比较
int main(){
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i){
        cin>>temp;
        insert(temp, i);
//新加元素插入到末端后调整至正确位置
    }
    size = n;
    while (size != 1){
        int num1 = heap[1];
        pop();
        int num2 = heap[1];
        pop();
        insert(num1 + num2, ++size);
        ans +=(num1 + num2)*2;
    }
    cout<<ans;
    return 0;
}
```