1098. Insertion or Heap Sort (25)

时间限制

100 ms

内存限制

65536 kB

代码长度限制

16000 B

判题程序

Standard

作者

CHEN, Yue

According to Wikipedia:

**Insertion sort** iterates, consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

**Heap sort** divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. it involves the use of a heap data structure rather than a linear-time search to find the maximum.

Now given the initial sequence of integers, together with a sequence which is a result of several iterations of some sorting method, can you tell which sorting method we are using?

**Input Specification:**

Each input file contains one test case. For each case, the first line gives a positive integer N (<=100). Then in the next line, N integers are given as the initial sequence. The last line contains the partially sorted sequence of the N numbers. It is assumed that the target sequence is always ascending. All the numbers in a line are separated by a space.

**Output Specification:**

For each test case, print in the first line either "Insertion Sort" or "Heap Sort" to indicate the method used to obtain the partial result. Then run this method for one more iteration and output in the second line the resuling sequence. It is guaranteed that the answer is unique for each test case. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

**Sample Input 1:**

10

3 1 2 8 7 5 9 4 6 0

1 2 3 7 8 5 9 4 6 0

**Sample Output 1:**

Insertion Sort

1 2 3 5 7 8 9 4 6 0

**Sample Input 2:**

10

3 1 2 8 7 5 9 4 6 0

6 4 5 1 0 3 2 7 8 9

**Sample Output 2:**

Heap Sort

5 4 3 1 0 2 6 7 8 9

[提交代码](https://www.patest.cn/contests/pat-a-practise/1098)

堆排知道原理就简单。。。但是在插排这边出了一些比较玄学的错误，再次说明一下推理过程：

对于vector的了解存在一个漏洞，即vector的.begin()对象是不存任何数据只作为一个开始标记而存在的，所以如果一开始的sort的sort(vector.begin(),vector.begin()+1)的话，只是扫到第一个元素，是不会进行任何的改动的。

同时，将变量insert\_it作为插排的指示位置的标记来做这道题的话，初始值设为1，即错误，初始值设为二，既正确。



屏蔽heap判断，结果如上图，则可以说明1，3，5是测试堆排的。



屏蔽插排检查，结果如图，说明0，2，4是测试插排的。

那问题就来了，对于插排数据2，当insert\_it=1时，只是对于第一次排序结果没有处理，但是直到排序完成为止，其余排序结果是一样的，那么就可以说，第一次排序结果和目标吻合，导致输出。

但是输出错误，也就是说，在后面的排序中，该组数据会出现一个和未处理过的数据一样的排列，而这个排列的结果并非相等于第二次排序的结果。

那有没有这样的一个数据，根据上面的推理，满足下面的所有条件呢？

1. 数组从a[0]-a[n]
2. a[0]-a[1]需要进行改动以后满足增序，即a[0],a[1]交换；
3. 在后续进行到某一步时，该数组会变得和一开始的数据一样

推到这一步，我们可以发现，2和3是矛盾的，因为进行了改动，也就是说a[0]大于a[1],那这样到后面是不可能还原原来的数列的。

问题出在假设有个数组，891729，目标同样是891729

那么设insert\_it值为1时，第一步是891729，第二步也还是891729，但是这个并不符合定义

只有将insert\_it设为2，第一步是891729，但是对于9进行检查过以后，第二步才会变成189729,才符合定义。

#include<iostream>

#include<string>

#include<algorithm>

#include<queue>

#include<vector>

#include<sstream>

#include<stack>

#include<map>

#include<cstring>

#include<climits>

#include<cmath>

#include<fstream>

#define MAX 1001

#define LL long long

using namespace std;

#define MAX 105

int heap\_size,insert\_it=2;

vector<int>origin;

vector<int>insert;

vector<int>heap;

vector<int>target;

void heaplify(int i)

{

int largest = i;

if (2 \* i-1 < heap\_size)

{

if (heap[2 \* i-1]>heap[i-1])

largest = 2 \* i;

}

if (2 \* i < heap\_size)

{

if (heap[2 \* i] > heap[largest-1])

largest = 2 \* i+1;

}

if (largest != i)

{

swap(heap[i-1], heap[largest-1]);

heaplify(largest);

}

else return;

}

void Build\_heap()

{

for (int i = heap\_size / 2 ; i >0; i--)

{

heaplify(i);

}

}

void heapsort()

{

swap(heap[0], heap[heap\_size - 1]);

heap\_size--;

heaplify(1);

}

void insertsort()

{

if (insert\_it < insert.size())

{

sort(insert.begin(), insert.begin() + insert\_it);

insert\_it++;

}

else

return;

}

void output(vector<int>result)

{

cout << result[0];

for (int i = 1; i < result.size(); i++)

{

cout << " " << result[i];

}

}

int main()

{

int num;

cin >> num;

heap\_size = num;

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

{

int temp;

cin >> temp;

origin.push\_back(temp);

}

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

{

int temp;

cin >> temp;

target.push\_back(temp);

}

insert = origin;

heap = origin;

vector<int>fininshed=origin;

sort(fininshed.begin(), fininshed.end());

Build\_heap();

while (1)

{

if (target == heap)

{

cout << "Heap Sort" << endl;

heapsort();

output(heap);

return 0;

}

heapsort();

if (heap == fininshed)

break;

}

while (1)

{

insertsort();

if (target == insert)

{

cout << "Insertion Sort" << endl;

insertsort();

output(insert);

return 0;

}

if (insert == fininshed)

break;

}

while (1) {}

}