B. Heapsort

Time Limit: 1.0 Seconds Memory Limit: 65536K Special Judge Multiple test files

A well known algorithm called *heapsort* is a deterministic sorting algorithm taking $O(n\log n)$ time and $O(1)$ additional memory. Let us describe ascending sorting of an array of different integer numbers.

The algorithm consists of two phases. In the first phase, called *heapification*, the array of integers to be sorted is converted to a *heap*. An array $a[1 \ldots n]$ of integers is called a heap if for all $1 \le i \le n$ the following *heap conditions* are satisfied:

- if $2i \le n$ then $a[i] > a[2i]$;
- if $2i + 1 \le n$ then $a[i] > a[2i + 1]$.

We can interpret an array as a binary tree, considering children of element $a[i]$ to be $a[2i]$ and $a[2i + 1]$. In this case the parent of $a[i]$ is $a[i \text{ div } 2]$, where $i \text{ div } 2 = \text{floor}(i/2)$. In terms of trees the property of being a heap means that for each node its value is greater than the values of its children.

In the second phase the heap is turned into a sorted array. Because of the heap condition the greatest element in the heapified array is $a[1]$. Let us exchange it with $a[n]$, now the greatest element of the array is at its correct position in the sorted array. This is called *extract-max*.

Now let us consider the part of the array $a[1 \ldots n\text{-}1]$. It may be not a heap because the heap condition may fail for $i = 1$. If it is so (that is, either $a[2]$ or $a[3]$, or both are greater than $a[1]$) let us exchange the greatest child of $a[1]$ with it, restoring the heap condition for $i = 1$. Now it is possible that the heap condition fails for the position that now contains the former value of $a[1]$. Apply the same procedure to it, exchanging it with its greatest child. Proceeding so we convert the whole array $a[1 \ldots n\text{-}1]$ to a heap. This procedure is called *sifting down*. After converting the part $a[1 \ldots n\text{-}1]$ to a heap by sifting, we apply extract-max again, putting second greatest element of the array to $a[n\text{-}1]$, and so on.

For example, let us see how the heap $a = (5, 4, 2, 1, 3)$ is converted to a sorted array. Let us make the first extract-max. After that the array turns to (3, 4, 2, 1, 5). Heap condition fails for $a[1] = 3$ because its child $a[2] = 4$ is greater than it. Let us sift it down, exchanging $a[1]$ and $a[2]$. Now the array is (4, 3, 2, 1, 5). The heap condition is satisfied for all elements, so sifting is over. Let us make extract-max again. Now the array turns to (1, 3, 2, 4, 5). Again the heap condition fails for $a[1]$; exchanging it with its greatest child we get the array (3, 1, 2, 4, 5) which is the correct heap. So we make extract-max and get (2, 1, 3, 4, 5). This time the heap condition is satisfied for all elements, so we make extract-max, getting (1, 2, 3, 4, 5). The leading part of the array is a heap, and the last extract-max finally gives (1, 2, 3, 4, 5).

It is known that heapification can be done in $O(n)$ time. Therefore, the most time consuming operation in heapsort algorithm is sifting, which takes $O(n\log n)$ time.

In this problem you have to find a heapified array containing different numbers from 1 to n, such that when converting it to a sorted array, the total number of exchanges in all sifting operations is maximal possible. In the example above the number of exchanges is $1 + 1 + 0 + 0 + 0 = 2$, which is not the maximum. (5, 4, 3, 2, 1) gives the maximal number of 4 exchanges for $n = 5$.

Input

Input contains n ($1 \le n \le 50\,000$).

Output

Output the array containing *n* different integer numbers from 1 to *n*, such that it is a heap, and when converting it to a sorted array, the total number of exchanges in sifting operations is maximal possible. Separate numbers by spaces.

Sample Input

```
6
```

Sample Output

```
6 5 3 2 4 1
```

Note: Special judge problem, you may get "Wrong Answer" when output in wrong format.

Source: Northeastern European 2004

Problem ID in problemset: 2398