**Heapsort**

Andreas Klappenecker

**Heapsort.** A heap is a vector $v[1..n]$ that represents a nearly complete binary tree. The root of the tree is $v[1]$, the left child of the node $v[i]$ is stored at $v[2i]$, and the right child is stored at $v[2i + 1]$.

A max-heap satisfies $v[i] \geq v[2i]$ for all $i$ such that $2i \leq n$, and $v[i] \geq v[2i + 1]$ for all $i$ such that $2i + 1 \leq n$. For instance,

$$(v[1], v[2], \ldots, v[7]) = (6, 2, 5, 0, 1, 4, 3)$$

represents a max-heap. One can create a max-heap from an unsorted vector in linear time.

Heap sort creates a heap from the unsorted input vector, swaps the elements $v[1]$ and $v[n]$ so that the largest element is contained in $v[n]$, restores $v[1..n-1]$ to a heap, and recursively applies the same procedure the heap $v[1..n-1]$. After $n - 1$ iterations, the vector is sorted.

**The Program.** We illustrate this concept by giving an implementation that generates a random input vector and prints each step of the heap sort algorithm. We use C++ and the standard template library for this task.

```cpp
#include<iostream>
#include<algorithm>
#include<vector>
#include<iterator>
#include<time.h>
using namespace std;

int main() {
    int len = 7;
    int i;
    vector<int> v(len);

    (random heap)
    (heap sort)
}
```

The program is contained in the file `heap.cpp`. It creates a random vector of length 7 with integer entries from 0 to 6, builds a heap in linear time, and prints this heap. The details are explained later in the definition of the code.
chunk \(<\text{random heap}\)>. The second part of the program performs the heap sort algorithm and prints each step.

Let us have a look at the details. The implementation creates the vector \(v = (1, 2, \ldots, 7)\), and applies a random permutation. The standard library call \texttt{make_heap} creates a max-heap in linear time and prints the resulting heap.

\begin{verbatim}
<random heap>
  for(i=0; i<len; i++)
    v[i] = i;
  srand(time(NULL));
  random_shuffle(v.begin(), v.end());
  make_heap(v.begin(), v.end());
  cout << "heap ";
  copy(v.begin(), v.end(), ostream_iterator<int>(cout, " "));

<heap sort>
  for(i=len; i>=2; i--)
    cout << endl << "top element " << v[0];
    pop_heap(v.begin(), v.begin()+i);
    cout << ", remaining heap ";
    copy(v.begin(), v.begin()+i-1, ostream_iterator<int>(cout, " "));

A sample run might produce, for instance, the output

heap 6 3 5 1 2 0 4
  top element 6, remaining heap 5 3 4 1 2 0
  top element 5, remaining heap 4 3 0 1 2
  top element 4, remaining heap 3 2 0 1
  top element 3, remaining heap 2 1 0
  top element 2, remaining heap 1 0
  top element 1, remaining heap 0

The algorithms extracts in each step the largest elements, namely 6, 5, 4, 3, 2, 1, 0. After the execution of the algorithm, the vector \(v[1..7]\) contains \((1, 2, \ldots, 7)\), as desired.