A priority queue

- A priority queue is a queue in which each element has a priority.
- Elements with higher priority are served before elements with lower priority.
- It can be implemented as a vector or a linked list. For a queue with \( n \) elements:
  - Insertion is \( O(n) \).
  - Extraction is \( O(1) \).
- A more efficient implementation can be proposed in which insertion and extraction are \( O(\log n) \): binary heap.

**Binary Heap**

- Complete binary tree (except at the bottom level).
- Height \( h \): between \( 2^h \) and \( 2^{h+1} - 1 \) nodes.
- For \( N \) nodes, the height is \( O(\log N) \).
- It can be represented in a vector.

**Heap-Order Property**: the key of the parent of \( X \) is smaller than (or equal to) the key in \( X \).
Two main operations on a binary heap:
- Insert a new element
- Remove the min element

Both operations must preserve the properties of the binary heap:
- Completeness
- Heap-Order property

Binary Heap: insert 14

Insert in the last location
... and bubble up ...
done!

Binary Heap: remove min

Extract the min element and move the last one to the root of the heap
... and bubble down ...
done!
Binary Heap: complexity

• Bubble up/down operations do at most $h$ swaps, where $h$ is the height of the tree and

$$h = \lceil \log_2 N \rceil$$

• Therefore:
  – Getting the min element is $O(1)$
  – Inserting a new element is $O(\log N)$
  – Removing the min element is $O(\log N)$

Containers: Priority Queues © Dept. CS, UPC

Binary Heap: other operations

• Let us assume that we have a method to know the location of every key in the heap.

  • Increase/decrease key:
    – Modify the value of one element in the middle of the heap.
    – If decreased $\rightarrow$ bubble up.
    – If increased $\rightarrow$ bubble down.

  • Remove one element:
    – Set value to $-\infty$, bubble up and remove min element.

Containers: Priority Queues © Dept. CS, UPC

Building a heap from a set of elements

• Heaps are sometimes constructed from an initial collection of $N$ elements. How much does it cost to create the heap?
  – Obvious method: do $N$ insert operations.
  – Complexity: $O(N \log N)$

• Can it be done more efficiently?
Building a heap: implementation

// Constructor from a collection of items
BinaryHeap(const vector<Elem>& items) {
    v.push_back(Elem()); // v is the vector holding the elements
    for (auto& e: items) v.push_back(e);
    for (int i = size()/2; i > 0; --i) bubble_down(i);
}

Containers: Priority Queues © Dept. CS, UPC

Sum of the heights of all nodes:

- 1 node with height \( h \)
- 2 nodes with height \( h - 1 \)
- 4 nodes with height \( h - 2 \)
- \( 2^i \) nodes with height \( h - i \)

\[
S = \sum_{i=0}^{h-1} 2^i (h - i)
\]

A heap can be built from a collection of items in linear time.

Heap sort

```
template<typename T>
void HeapSort(vector<T>& v) {
    BinaryHeap<T> heap(v);
    for (T& e: v) e = heap.remove_min();
}
```

- Complexity: \( O(n \log n) \)
  - Building the heap: \( O(n) \)
  - Each removal is \( O(\log n) \), executed \( n \) times.

Exercise: insert/remove element

Given the binary heap implemented in the following vector, draw the tree represented by the vector.

```
6 7 9 10 11 12 13 15 19 14 21 17 16
```

Execute the following sequence of operations

```
insert(8); remove_min(); insert(6); insert(18); remove_min();
```

and draw the tree after the execution of each operation.
Consider the binary heap of integer keys implemented by the following vector:

![Binary Heap Example](image)

After executing the operations `insert(8)` and `remove_min()` the contents of the binary heap is:

![Updated Binary Heap](image)

Discuss about the possible values of $a$ and $b$. Assume there can never be two identical keys in the heap.

**Exercise: the $k$-th element**

The $k$-th element of $n$ sorted vectors.

Let us consider $n$ vectors sorted in ascending order.

Design an algorithm with cost $\Theta(k \log n + n)$ that finds the $k$-th global smallest element.

**Exercise: bubble-up/down**

Consider the following declaration for a Binary Heap:

```cpp
template <typename T> // T must be a comparable type
class BinaryHeap {
private:
    vector<Elem> v; // Table for the heap (location 0 not used)

    // Bubbles up the element at location i
    void bubble_up(int i);

    // Bubbles down the element at location i
    void bubble_down(int i);
};
```

Give an implementation for the methods `bubble_up` and `bubble_down`. 