Heap (data structure)
From Wikipedia, the free encyclopedia

In computer science, a **heap** is a specialized tree-based data structure that satisfies the **heap property**: if B is a child node of A, then \( \text{key}(A) \geq \text{key}(B) \). This implies that an element with the greatest key is always in the root node, and so such a heap is sometimes called a **max-heap**. (Alternatively, if the comparison is reversed, the smallest element is always in the root node, which results in a **min-heap**.) There is no restriction as to how many children each node has in a heap, although in practice each node has at most two. The heap is one maximally-efficient implementation of an abstract data type called a priority queue. Heaps are crucial in several efficient graph algorithms such as Dijkstra's algorithm, and in the sorting algorithm heapsort.

A heap data structure should not be confused with the **heap** which is a common name for dynamically allocated memory. The term was originally used only for the data structure. Some early popular languages such as LISP provided dynamic memory allocation using heap data structures, which gave the memory area its name\(^1\).

Heaps are usually implemented in an array, and do not require pointers between elements.

The operations commonly performed with a heap are:

- **create-heap**: create an empty heap
- **find-max** or **find-min**: find the maximum item of a max-heap or a minimum item of a min-heap, respectively
- **delete-max** or **delete-min**: removing the root node of a max- or min-heap, respectively
- **increase-key** or **decrease-key**: updating a key within a max- or min-heap, respectively
- **insert**: adding a new key to the heap
- **merge**: joining two heaps to form a valid new heap containing all the elements of both.

## Contents

- 1 Variants
- 2 Comparison of theoretic bounds for variants
- 3 Applications
- 4 Implementations
- 5 See also
- 6 References

### Variants

- 2-3 heap
- Beap
- Binary heap
- Binomial heap
- Brodal queue
- D-ary heap
- Fibonacci heap
- Leftist heap
- Pairing heap
- Skew heap
- Soft heap
- Ternary heap
- Treap

Comparison of theoretic bounds for variants

The following time complexities[1] are amortized (worst-time) time complexity for entries marked by an asterisk, and regular worst case time complexities for all other entries. O(f) gives asymptotic upper bound and Θ(f) is asymptotically tight bound (see Big O notation). Function names assume a min-heap.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>create-heap</td>
<td>Θ(1)</td>
<td>Θ(1)</td>
<td>Θ(1)</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>findMin</td>
<td>Θ(1)</td>
<td>O(log n)</td>
<td>Θ(1)</td>
<td>O(1)*</td>
<td>Θ(1)</td>
</tr>
<tr>
<td>deleteMin</td>
<td>Θ(log n)</td>
<td>Θ(log n)</td>
<td>O(log n)*</td>
<td>O(log n)*</td>
<td>O(log n)</td>
</tr>
<tr>
<td>insert</td>
<td>Θ(log n)</td>
<td>O(log n)</td>
<td>Θ(1)</td>
<td>O(1)*</td>
<td>Θ(1)</td>
</tr>
<tr>
<td>decreaseKey</td>
<td>Θ(log n)</td>
<td>Θ(log n)</td>
<td>Θ(1)*</td>
<td>O(log n)*</td>
<td>Θ(1)</td>
</tr>
<tr>
<td>merge</td>
<td>Θ(n)</td>
<td>O(log n)**</td>
<td>Θ(1)</td>
<td>O(1)*</td>
<td>Θ(1)</td>
</tr>
</tbody>
</table>

(*) Amortized time
(**) Where n is the size of the larger heap

Applications

The heap data structure has many applications.

- Heapsort: One of the best sorting methods being in-place and with no quadratic worst-case scenarios.
- Selection algorithms: Finding the min, max, both the min and max, median, or even the k-th largest element can be done in linear time (often constant time) using heaps.[4]
- Graph algorithms: By using heaps as internal traversal data structures, run time will be reduced by polynomial order. Examples of such problems are Prim's minimal spanning tree algorithm and Dijkstra's shortest path problem.

Full and almost full binary heaps may be represented in a very space-efficient way using an array alone. The first (or last) element will contain the root. The next two elements of the array contain its children. The next four contain the four children of the two child nodes, etc. Thus the children of the node at position n would be at positions 2n and
\(2n+1\) in a one-based array, or \(2n+1\) and \(2n+2\) in a zero-based array. This allows moving up or down the tree by doing simple index computations. Balancing a heap is done by swapping elements which are out of order. As we can build a heap from an array without requiring extra memory (for the nodes, for example), heapsort can be used to sort an array in-place.

One more advantage of heaps over trees in some applications is that construction of heaps can be done in linear time using Tarjan's algorithm.

**Implementations**

- The C++ Standard Template Library provides the make_heap, push_heap and pop_heap algorithms for heaps (usually implemented as binary heaps), which operate on arbitrary random access iterators. It treats the iterators as a reference to an array, and uses the array-to-heap conversion.
- The Java 2 platform (since version 1.5) provides the binary heap implementation with class java.util.PriorityQueue\(\langle E\rangle\) in Java Collections Framework.
- Python has a heapq (http://docs.python.org/library/heapq.html) module that implements a priority queue using a binary heap.
- PHP has both maxheap (SplMaxHeap) and minheap (SplMinHeap) as of version 5.3 in the Standard PHP Library.
- Perl has implementations of binary, binomial, and Fibonacci heaps in the Heap (http://search.cpan.org/perldoc?Heap) distribution available on CPAN.

**See also**

- Stack (data structure)
- Sorting algorithm

**References**


Categories: Heaps (structure)