Heapsort

Binary Heaps
The Heap Property
Building a Heap
Heapsort
Priority Queues
Illustration of a Max-heap

Parent Left Right
Heap Property
Max-Heapify
Max-Heapify 2
Build-Max-Heap
Build-Max-Heap 1
Build-Max-Heap 2
Heapsort
Heapsort 1
Heapsort 2
Priority Queue
Heap-Extract-Max
Heap-Increase-Key
Max-Heap-Insert
Heap-Increase-Key 2
Parent Left Right

Illustration
 indentation
 Parent Left
 Right

Heap Property
 Max-Heapify
 Max-Heapify 2
 Build-Max-Heap
 Build-Max-Heap 1
 Build-Max-Heap 2
 Heapsort
 Heapsort 1
 Heapsort 2
 Priority Queue
 Heap-Extract-Max
 Heap-Increase-Key
 Max-Heap-Insert
 Heap-Increase-Key 2

\[
\text{PARENT}(i) \quad 1 \quad \text{return } \lfloor i/2 \rfloor
\]

\[
\text{LEFT}(i) \quad 1 \quad \text{return } 2i
\]

\[
\text{RIGHT}(i) \quad 1 \quad \text{return } 2i + 1
\]
Heap Property

There are two kinds of binary heaps: max-heaps and min-heaps. In both kinds, the values in the nodes satisfy a heap property, the specifics of which depend on the kind of heap. In a max-heap, the max-heap property is that for every node $i$ other than the root,

$$A[\text{PARENT}(i)] \geq A[i],$$

that is, the value of a node is at most the value of its parent. Thus, the largest element in a max-heap is stored at the root, and the subtree rooted at a node contains values no larger than that contained at the node itself. A min-heap is organized in the opposite way; the min-heap property is that for every node $i$ other than the root,

$$A[\text{PARENT}(i)] \leq A[i].$$

The smallest element in a min-heap is at the root.
Max-Heapify

\[
\text{MAX-HEAPIFY} (A, i) \\
1 \quad l = \text{LEFT}(i) \\
2 \quad r = \text{RIGHT}(i) \\
3 \quad \text{if } l \leq A.\text{heap-size} \text{ and } A[l] > A[i] \\
4 \quad \text{largest} = l \\
5 \quad \text{else } \text{largest} = i \\
6 \quad \text{if } r \leq A.\text{heap-size} \text{ and } A[r] > A[\text{largest}] \\
7 \quad \text{largest} = r \\
8 \quad \text{if } \text{largest} \neq i \\
9 \quad \text{exchange } A[i] \text{ with } A[\text{largest}] \\
10 \quad \text{MAX-HEAPIFY} (A, \text{largest})
\]
Max-Heapify Illustration

- Illustration
- Parent Left Right
- Heap Property
- Max-Heapify
  - Max-Heapify 2
- Build-Max-Heap
  - Build-Max-Heap 1
  - Build-Max-Heap 2
- Heapsort
  - Heapsort 1
  - Heapsort 2
- Priority Queue
- Heap-Extract-Max
- Heap-Increase-Key
- Max-Heap-Insert
- Heap-Increase-Key 2

(a) Max-Heapify 2
(b) Build-Max-Heap
(c) Heapsort 1
**Build-Max-Heap**

**BUILD-MAX-HEAP** $(A)$

1. $A.heap-size = A.length$
2. for $i = \lceil A.length/2 \rceil$ downto $1$
3. MAX-HEAPIFY $(A, i)$
Build-Max-Heap Illustration, Part 1

Illustration
Parent Left Right
Heap Property
Max-Heapify
Max-Heapify 2
Build-Max-Heap
Build-Max-Heap 1
Build-Max-Heap 2
Heapsort
Heapsort 1
Heapsort 2
Priority Queue
Heap-Extract-Max
Heap-Increase-Key
Max-Heap-Insert
Heap-Increase-Key 2
Build-Max-Heap Illustration, Part 2

Illustration
Parent Left Right
Heap Property
Max-Heapify
Max-Heapify 2
Build-Max-Heap
Build-Max-Heap 1
Build-Max-Heap 2
Heapsort
Heapsort 1
Heapsort 2
Priority Queue
Heap-Extract-Max
Heap-Increase-Key
Max-Heap-Insert
Heap-Increase-Key 2

(c) Build-Max-Heap 2
(d) Max-Heapify 2
(e) Max-Heapify
(f) Build-Max-Heap
Heapsort Algorithm

Heapsort (A)

1. BUILD-MAX-HEAP (A)
2. for i = A.length downto 2
3. A.heap-size = A.heap-size - 1
4. MAX-HEAPIFY (A, 1)
Heapsort Illustration, Part 1

Illustration
Parent Left Right
Heap Property
Max-Heapify
Max-Heapify 2
Build-Max-Heap
Build-Max-Heap 1
Build-Max-Heap 2
Heapsort
Heapsort 1
Heapsort 2
Priority Queue
Heap-Extract-Max
Heap-Increase-Key
Max-Heap-Insert
Heap-Increase-Key 2

(a) Heapsort Illustration

(b) Heapsort Illustration

(c) Heapsort Illustration

(d) Heapsort Illustration

(e) Heapsort Illustration

(f) Heapsort Illustration
Heapsort Illustration, Part 2

Illustration
Parent Left Right
Heap Property
Max-Heapify
Max-Heapify 2
Build-Max-Heap
Build-Max-Heap 1
Build-Max-Heap 2
Heapsort
Heapsort 1
Heapsort 2
Priority Queue
Heap-Extract-Max
Heap-Increase-Key
Max-Heap-Insert
Heap-Increase-Key 2

Heapsort
Max-Heapify
Build-Max-Heap
Heapsort

CS 3343 Analysis of Algorithms
A **priority queue** is a data structure for maintaining a set $S$ of elements, each with an associated value called a **key**. A **max-priority queue** supports the following operations:

**INSERT**$(S, x)$ inserts the element $x$ into the set $S$, which is equivalent to the operation $S = S \cup \{x\}$.

**MAXIMUM**$(S)$ returns the element of $S$ with the largest key.

**EXTRACT-MAX**$(S)$ removes and returns the element of $S$ with the largest key.

**INCREASE-KEY**$(S, x, k)$ increases the value of element $x$’s key to the new value $k$, which is assumed to be at least as large as $x$’s current key value.
Heap-Extract-Max

1. if $A$.heap-size < 1
2. 
   error “heap underflow”
3. max = $A[1]$
5. $A$.heap-size = $A$.heap-size - 1
6. MAX-HEAPIFY ($A$, 1)
7. return max
Heap-Increase-Key

\textbf{HEAP-INCREASE-KEY}(A, i, key)

1. \textbf{if} key \textless A[i]
2. \textbf{error} “new key is smaller than current key”
3. \textbf{A[i]} = key
4. \textbf{while} i > 1 \textbf{and} A[\text{PARENT}(i)] < A[i]
5. exchange A[i] with A[\text{PARENT}(i)]
6. i = \text{PARENT}(i)
Max-Heap-Insert

\[
\text{MAX-HEAP-INSERT}(A, \text{key})
\]

1. \(A.\text{heap-size} = A.\text{heap-size} + 1\)
2. \(A[A.\text{heap-size}] = -\infty\)
3. \(\text{HEAP-INCREASE-KEY}(A, A.\text{heap-size}, \text{key})\)
Heap-Increase-Key Illustration

Illustration
Parent Left Right
Heap Property
Max-Heapify
Max-Heapify 2
Build-Max-Heap
Build-Max-Heap 1
Build-Max-Heap 2
Heapsort
Heapsort 1
Heapsort 2
Priority Queue
Heap-Extract-Max
Heap-Increase-Key
Max-Heap-Insert
Heap-Increase-Key
2

\( i \)