Heap sort

Heap is:

Heap sort

Heap is:

1. a binary tree
2. with heap order
3. and heap shape
4. laid out in an array

Heap sort

Heap is:

1. a binary tree

each node can have a left and a right branch

2. with heap order
3. and heap shape
4. laid out in an array

(Note: “heap” is also used to refer to a memory area used for allocating objects during a program’s execution. The two uses are unrelated.)

[Williams, 1964]

1) Binary tree

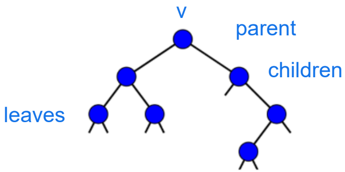
A binary tree is either

- The empty tree

or

- A node **v** and two subtrees (a right and a left).

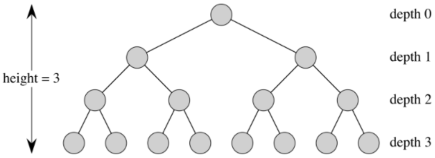
Visualization:



The node **v** is also called the root of the tree. If **v** has a non-empty subtree, the root **u** of this subtree is called a child of v, and v is called u's parent. If both of v's subtrees are empty, v is called a leaf. The lines between children and parents are called edges. The parent/child concept naturally generalizes to ancestor and descendant.

1) Binary tree

- Depth of a node = number of edges to the root
- Height of a node = max number of edges to a leaf
- Height of a tree = height of its root
- Full (Complete) binary tree = a tree where all levels are completely filled



A complete binary tree of height h has

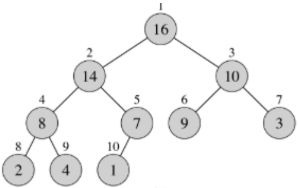
$$1 + 2 + 4 + 8 + \cdots + 2^h = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$ geometric sequence

Nodes (formula A.5 page 1147), of which 2^h are leaves.

2) Heap order

A binary tree with values in all nodes is max-heap ordered if, for any node v with a child u, it holds that:

value in v ≥ value in u



(a)

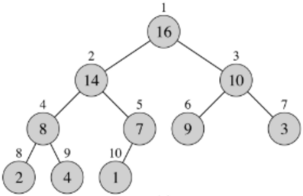
[Note: An equivalent definition is that for any node v with a descendant u, it holds that **value in v ≥ value in u** .]

In a **max-heap ordered** tree, the root contains the largest value in the entire heap.

A tree is **min-heap ordered** if the above holds with ≤ instead of ≥.

3) Heap shape

A binary tree has a heap shape if all levels in the tree are completely filled, except for the last level, where all nodes are positioned as far to the left as possible. (In particular, a full tree has a heap shape).



(a)

For a tree with a heap shape of height *h* with *n* nodes:

heap shape more nodes then in a full tree with one less level

n > number of nodes in a full tree of height h −1 = $2^h - 1$

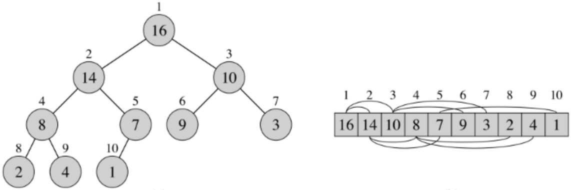
from previous formula
if you 2^(h+1) -1 - h-1
=2h-1

$$n > 2^h - 1 \Leftrightarrow n+1 > 2^h \Leftrightarrow \log_2(n+1) > h$$

upper bound of height of a heap shape

4) Heap represented in an array

A binary tree in heap form can naturally be represented in an array by assigning array indices to nodes using a top-down, left-to-right traversal of the tree's levels.



(a) (b)

Navigation between children and parents in the array version can be performed using simple calculations: A node at position *i* has:

- ► A Parent in the place $\lfloor i/2 \rfloor$
- ► Children in the place 2i and 2i +1

(See the figure above. A formal proof is left for exercise sessions.)

Operations on a heap

We want to perform the following operations on a heap:

- Max-Heapify: Given a node with two subtrees, each of which satisfies the heap property, make the entire subtree of the node satisfy the heap property.
  is to restore or maintain the max-heap property within a binary tree structure. Remember, the max-heap property is: for any node, its value is greater than or equal to the values of its children.
- Build-Max-Heap: Convert n input elements (unordered) into a heap.

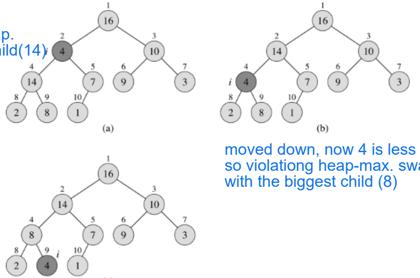
[The names above are for a max-heap. For a min-heap, the same operations exist with "min-" instead of "max-" in the name.]

Max-Heapify

Given a node with two subtrees, each of which satisfies the heap order, make the entire node's tree satisfy the heap order.

- ► Problem: The node's value is smaller than one or both of its children's values.
- ► Solution: Swap places with the child having the largest value, then run Max-Heapify on this child.

4 is smaller than 14,
so violation max-heap.
swap with biggest child(14)



(a) (b) (c)

moved down, now 4 is less than 8,
so violationg heap-max. swap place
with the biggest child (8)

Time: O(height of node).

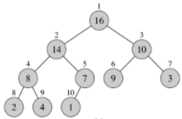
In the worst case, Max-Heapify might have to perform swaps and recursive calls all the way down a path from the starting node to a leaf in its subtree.

Max-Heapify

As pseudo-code (with an incorporated check to ensure you're not looking "too far" in the array, i.e., further than position n):

MAX-HEAPIFY(A, i, n)

$l = \text{LEFT}(i)$
$r = \text{RIGHT}(i)$
if $l \leq n$ and $A[l] > A[i]$
    $largest = l$
else $largest = i$
if $r \leq n$ and $A[r] > A[largest]$
    $largest = r$
if $largest \neq i$
    exchange $A[i]$ with $A[largest]$
    MAX-HEAPIFY$(A, largest, n)$

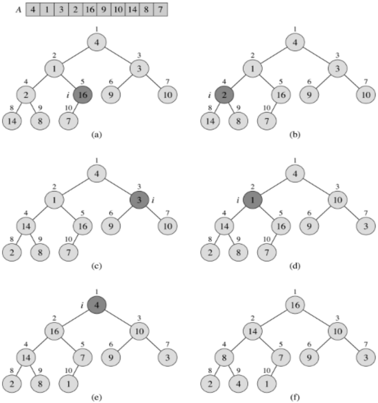


(a)

Build Heap

Create n input elements (unordered) into a heap.

- Idea: arrange the elements in heap form, then bring the tree into heap order from the bottom up.
- Observation: a tree of size one always satisfies heap order.

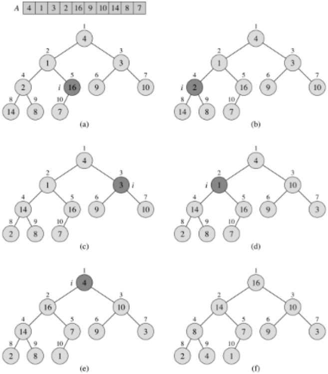


Time: O(n.log2 .n) clearly. Better analysis provides O(n).

Build Heap

As pseudo-code:

```
BUILD-MAX-HEAP(A, n)
 for i = ⌊n/2⌋ downto 1
     MAX-HEAPIFY(A, i, n)
```



Heap sort

A form of selection sort where a heap is used to continually extract the largest remaining element:

Build a heap

Repeat until the heap is empty:

Extract the root (the largest element in the heap)

Set the last element as the new root

Restore the heap structure by performing Max-Heapify on the new root.

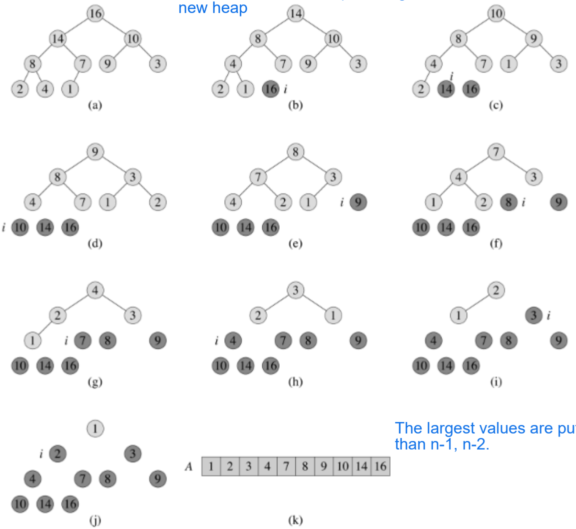
Heap sort

Example:

build the heap

the largest value swap
place with the lowest value
and the largest value is being removed.
Afterwards we use heap max again, to form
new heap

Skriv tekst her



The largest values are put from the end of the array (n), than n-1, n-2.

Heap sort

As pseudo-code:

```
HEAPSORT(A, n)
  BUILD-MAX-HEAP(A, n)
  for i = n downto 2
      exchange A[1] with A[i]
      MAX-HEAPIFY(A, 1, i − 1)
```

swap the first element (top of the array) with the last and left to the biggest element

because a smaller element may be on top of the tree (begining of array)

Time: O(n) + O(n. log n) = O(n. log n)

Three n log n sorting algorithms

	Worstcase	Inplace
QuickSort		$\surd$
MergeSort	$\surd$	
HeapSort	$\surd$	$\surd$

Heapsort runs slower than Mergesort and Quicksort due to the inefficient use of memory (random access).

Introsort [Musser, 1997]: uses Quicksort, but switches to Heapsort during recursion if the recursion becomes too deep. This provides an inplace, worst-case O(n log n) algorithm, with good runtime in practice (this is the sorting algorithm in the standard library STL for C++).