Sequential Structures

Heaps[1] and Priority Queues

[1] Stapel är den bästa svenska översättningen

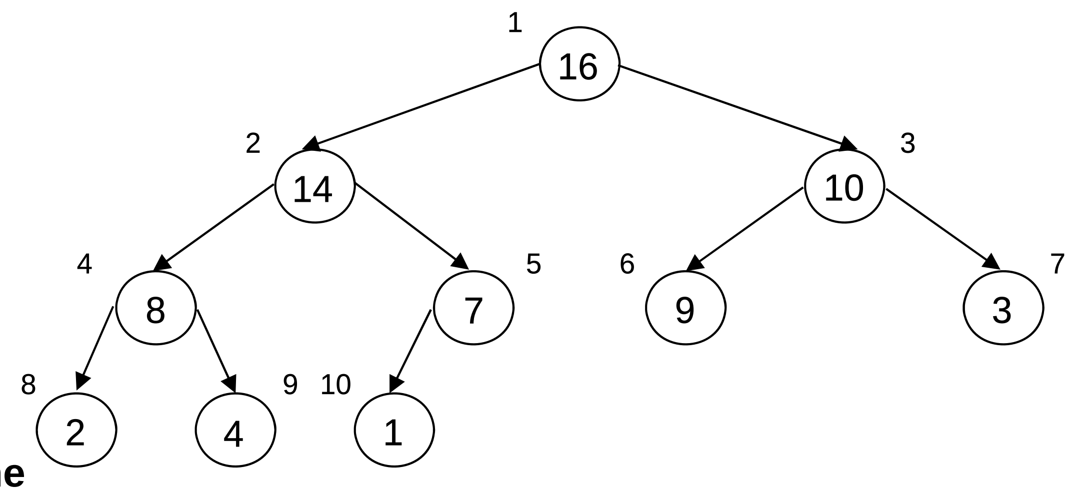
Agenda

- In this lesson:
  - Heaps (Binary)
    - Terminology
    - Organisation
    - Definition/Properties
    - Operations
    - Algorithms
  - Priority Queues
    - Definition
    - Properties
    - Comparison with a “normal” queue
    - Implementation

Heap

- **Terminology**
  - **Left/right child**
  - **Heap-invariant**
  - **Complete Binary Tree**
- **Organisation**
  - **Hierarchical Organisation**
  - **Height order**
    - **decreasing**
    - **increasing**
- **Heap Order Properties**
  - **For each node X the key/value in the parent of X is less/greater than or equal to the key in X, with the exception of the root**
- **Implementation**
  - **Sequence**
  - **More effective with an array**

H



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

A

Heap

- Definitions
  - For some index i, in a heap represented by array A
    - Parent: **Parent(i) = Low(i / 2)[1]**
    - Left Child: **Left(i) = 2i**
    - Right Child: **Right(i) = 2i + 1**
  - Invariant for a descending order heap represented by array A
    - **A[i] >= A[Left(i)] && A[i] >= A[Right(i)] OR**
    - **A[i] >= A[2i] && A[i] >= A[2i + 1]** **i.e. the left/right child values**

[1] integer division – low is rounded down

Heap

- Properties
  - The greatest value is found in the first position in a **descending heap**
  - The smallest value in the first position in an **ascending heap**
  - performance - all operations: **logarithmic O(log n)** except for **Build** whose complexity is **O(n)** and **findMin / findMax** which are constant.

Heap - Operationer

Operation	In	Out
Build	**A**	**H**
Create		**H**
Add	**H** x **v**	**H**
Remove	**H** x **r**	**H**
Find	**H** x **v**	**r**
Size	**H**	**n**
Max[1]	**H**	**v**
Min[2]	**H**	**v**
RemoveMax[1]	**H**	**H**
RemoveMin[2]	**H**	**H**

A minimum number of operations is Min/Max, RemoveMin/Max, Add and isEmpty. Often decreaseKey/increaseKey is required to change the prioroty of an object. In general Remove is often not required. Why?

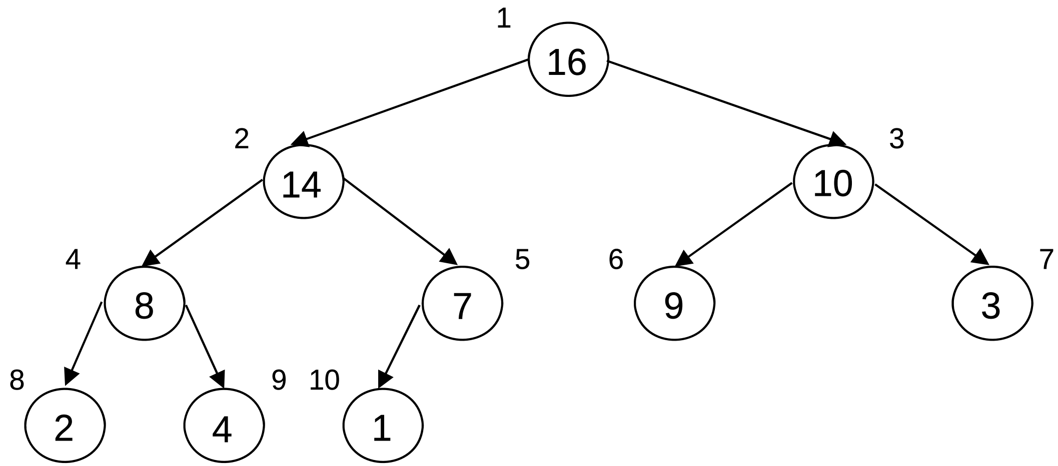
Heap - Operations (contd.)

- New pseudo operation
  - ***Heapify***
- Recursive operation which
  - Assumes that all children for a given element fulfil the invariant
  - Guarantees that the element fulfils the invariant

Heap - Operations (contd.)

```
Heapify(A, i)
   l = Left(i)              // 2*i
   r = Right(i)             // 2*i+1
   if l <= A.size and A[l] > A[i]
      then largest = l
      else largest = i
   if r <= A.size and A[r] >
      A[largest] then largest = r
   if largest != i then
      swap(A[i], A[largest])
      Heapify (A, largest)
   end if
end Heapify
```

H

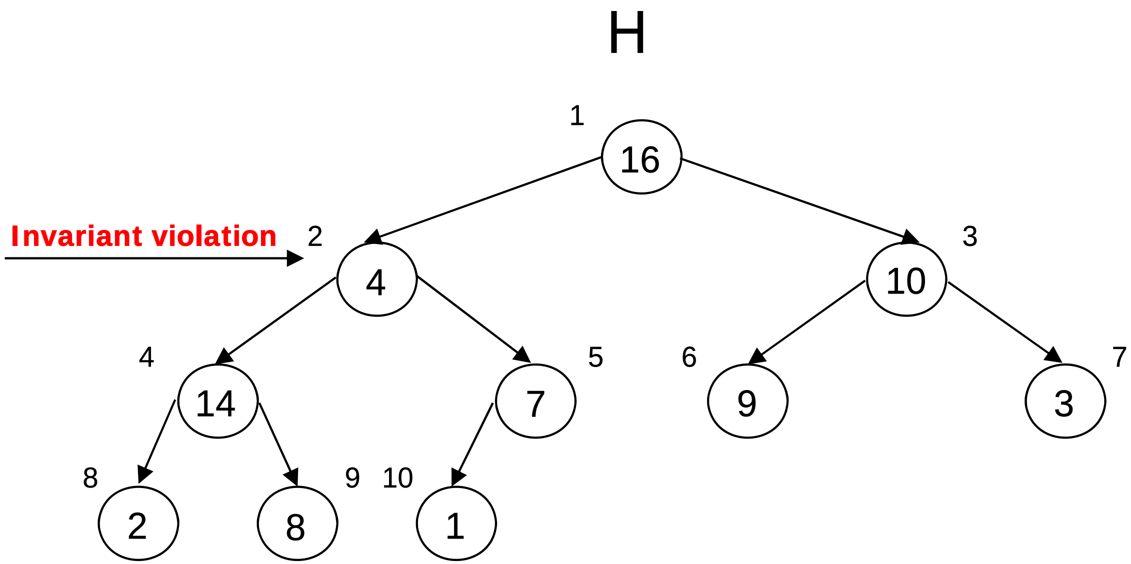


1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

A

Heap - Operations (contd.)

- This structure does NOT fulfil the heap invariant, element 2 violates the invariant
- To restore the invariant, the operation Heapify(A, 2) is applied



1	2	3	4	5	6	7	8	9	10
16	4	10	14	7	9	3	2	8	1

A

Heap - Operations (contd.)

- The operation **BuildHeap** may be defined with the help of **Heapify**

```
BuildHeap(A)
  for i = [A.size / 2]¹ downto 1
    do Heapify(A, i)
end Build
```

- **BuildHeap** takes an arbitrary array and modifies it to a heap

---

[1] **why do we begin with [A.size / 2]?**

Heap - Operations (contd.)

- The operation **Remove** may be defined with the help of **Heapify**.

```
Remove(A, i)
  A[i] = A[A.size]
  A.size--
  Heapify(A, i)
end Remove
```

Heap - Operations (contd.)

- **Add** is implemented without **Heapify**
  - Here we use the fact that the parent is ordered with respect to the child (ascending descending)
  - The parents “bubble” down the tree!

```
Add(A, v)
  A.size++
  i = A.size
  while i > 1 and A[Parent(i)] < v
    do A[i] = A[Parent(i)]
       i = Parent(i)
    end while
  A[i] = v
end Add
```

Priority Queue (PQ)

- Definition
  - A sequence
  - Each element has an associated priority
  - The first element in the queue has the highest priority
    - All other elements have the same or lower priority
    - The order is thus based on the priority

Priority Queue (PQ) (contd.)

- Compared with a “normal” queue
  - Order
    - properties
      - The (normal) queue has FIFO-order
      - The Priority Queue has priority-order
    - Effect
      - The first element added is not necessarily the first element in the queue!

Priority Queue - Operations

Operation	In	Out
Enqueue	**Q** x **v**	**Q**
Dequeue	**Q**	**Q**
Front	**Q**	**v**
IsEmpty	**Q**	True or False

Priority Queue - Implementation

- List
  - Using a linked list
  - Performance - **Linear access O(n)**
- Heap
  - Using a heap i in ascending or descending order depending on how the priority is defined.
  - Performance - **Logarithmic access O(log n)**

Application Area

- Priority Queues are used in
  - Operating systems
    - Process management (= executing programs)
    - Printer queues
  - Generally in systems where priority plays a significant rôle

Summary

- Heap
  - Is implemented using a **sequence**
    - Most efficient with an array (A)
  - **Invariant**: **A[i] >= A[Left(i)] && A[i] >= A[Right(i)]** for a descending order heap
  - Order is defined between parents and children
  - There is NO order between children
  - The position of the lowest/greatest value is always known (depending on the order)

Summary

- Priority Queue
  - This is not really a queue (not FIFO)
    - There are "similar" operations applied to a queue and priority queue but the **semantics is different**
  - The highest priority value comes first, the remaining values are of equal or lower priority

Reference Literature

- Data Structures and Problem Solving Using C++, [Weiss]
  - sid. 755-777
- Introduction to Algorithms, [Cormen, Leiserson, Rivest]
  - sid. 140-152