HEAP SORT

Sunawar Khan

International Islamic University
Islamabad

January 04, 2016

INTRODUCTION

- What is Full Binary Tree
  - Binary tree in which each node is either a leaf or has degree exactly 2.

INTRODUCTION

- What is Full Binary Tree
  - Binary tree in which each node is either a leaf or has degree exactly 2.
- What is Complete binary tree
  - Binary tree in which all leaves are on the same level and all internal nodes have degree 2

INTRODUCTION

- What is Full Binary Tree
  - Binary tree in which each node is either a leaf or has degree exactly 2.
- What is Complete binary tree
  - Binary tree in which all leaves are on the same level and all internal nodes have degree 2
- What is Heap Sort
  - Combines the better attributes of merge sort and insertion sort.
    - Like merge sort, but unlike insertion sort, running time is $O(nlgn)$.
    - Like insertion sort, but unlike merge sort, sorts in place.

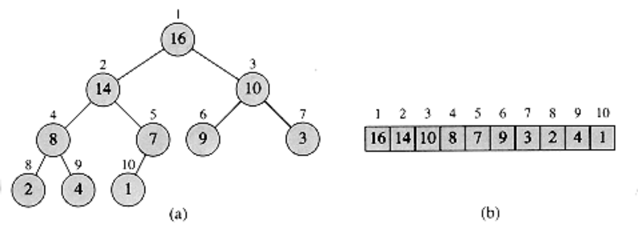
INTRODUCTION

- What is Full Binary Tree
  - Binary tree in which each node is either a leaf or has degree exactly 2.
- What is Complete binary tree
  - Binary tree in which all leaves are on the same level and all internal nodes have degree 2
- What is Heap Sort
  - Combines the better attributes of merge sort and insertion sort.
    - Like merge sort, but unlike insertion sort, running time is $O(nlgn)$.
    - Like insertion sort, but unlike merge sort, sorts in place.
  - Introduces an algorithm design technique
    - Create data structure (heap) to manage information during the execution of an algorithm.

INTRODUCTION

- What is Full Binary Tree
  - Binary tree in which each node is either a leaf or has degree exactly 2.
- What is Complete binary tree
  - Binary tree in which all leaves are on the same level and all internal nodes have degree 2
- What is Heap Sort
  - Combines the better attributes of merge sort and insertion sort.
    - Like merge sort, but unlike insertion sort, running time is $O(nlgn)$.
    - Like insertion sort, but unlike merge sort, sorts in place.
  - Introduces an algorithm design technique
    - Create data structure (heap) to manage information during the execution of an algorithm.
  - The heap has other applications beside sorting.
    - Priority Queues

HEAP-Example



(a) (b)

What is Heap-Properties

- Properties of Heap is
  - Completeness Property
  - Order Proprty

What is Heap-Properties

- Properties of Heap is
  - Completeness Property
  - Order Proprty

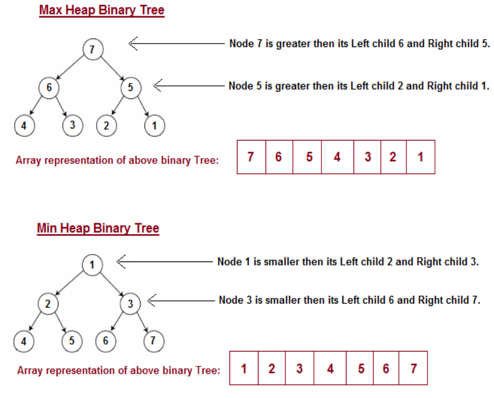
- Types of Heap is
  - Max Heap
    $A[PARENT(i)] \geqslant A[i]$
  - Min Heap
    $A[PARENT(i)] \leq A[i]$

What is Heap-Properties

- Properties of Heap is
  - Completeness Property
  - Order Proprty

- Types of Heap is
  - Max Heap
    $A[PARENT(i)] \geqslant A[i]$
  - Min Heap
    $A[PARENT(i)] \leq A[i]$

HEAP Properties Example



What is Heap-Characteristic

- Height of a node = the number of edges on the longest simple path from the node down to a leaf $\lfloor lgn \rfloor$.

What is Heap-Characteristic

- Height of a node = the number of edges on the longest simple path from the node down to a leaf $\lfloor lgn \rfloor$.

- Level of a node = the length of a path from the root to the node. $\lfloor n/2 \rfloor$.

What is Heap-Characteristic

- Height of a node = the number of edges on the longest simple path from the node down to a leaf $\lfloor lgn \rfloor$.
- Level of a node = the length of a path from the root to the node. $\lfloor n/2 \rfloor$.
- Height of tree = height of root node. height h $\leq \lfloor n/2h + 1 \rfloor$

Heap Representations

- Heap can Be Represented As
  - Root of tree is A[1]

Heap Representations

- Heap can Be Represented As
  - Root of tree is A[1]
  - Left

$$A[i] = A[2i]$$

Heap Representations

- Heap can Be Represented As
  - Root of tree is A[1]
  - Left
    $$A[i] = A[2i]$$
  - Right
    $$A[i] = A[2i + 1]$$

Heap Representations

- Heap can Be Represented As
  - Root of tree is A[1]
  - Left
    $$A[i] = A[2i]$$
  - Right
    $$A[i] = A[2i + 1]$$
  - Parent
    $$A[i] = A[\lfloor i/2 \rfloor]$$

Heap Representations

- Heap can Be Represented As
  - Root of tree is A[1]
  - Left
    $$A[i] = A[2i]$$
  - Right
    $$A[i] = A[2i + 1]$$
  - Parent
    $$A[i] = A[\lfloor i/2 \rfloor]$$
  - Heapsize$[A] \leq$ Length$[A]$

Problems

- What are the minimum and maximum numbers of elements in a heap of height h?
- Show that an n-element heap has height $\lfloor lgn \rfloor$.
- Where in a max-heap might the smallest element reside, assuming that all elements are distinct?
- Is an array that is in sorted order a min-heap?
- Is the array with values $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ a max-heap?

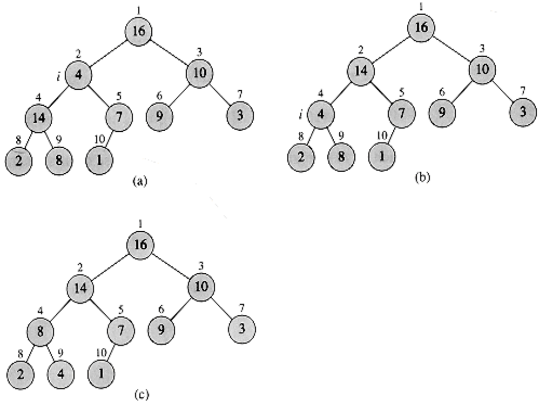
Maintaining The Heap Property

MAXIMUM-HEAPIFY PROCEDURE

MAX-HEAPIFY(A,i)

1. l $\leftarrow$ LEFT(i)
2. r $\leftarrow$ RIGHT(i)
3. if l $\leq$ A.heap-size and A[l]$>$A[i]
4. largest $\leftarrow$ l
5. else largest $\leftarrow$ i
6. if r $\leq$ A.heap-size and A[r]$>$A[largest]
7. largest $\leftarrow$ r
8. if largest $\leftarrow$ i
9. exchange A[i] $\leftrightarrow$ A[largest]
10. MAX-HEAPIFY(A, largest)

Maintain Heap Example



Running Time of MAX-HEAPIFY

- Running time of MAX-HEAPIFY is $O(lgn)$

Running Time of MAX-HEAPIFY

- Running time of MAX-HEAPIFY is $O(lgn)$
- Can be written in terms of the height of the heap, as being $O(h)$ Since the height of the heap is $\lfloor \lg n \rfloor$

Running Time of MAX-HEAPIFY

- Running time of MAX-HEAPIFY is $O(lgn)$
- Can be written in terms of the height of the heap, as being $O(h)$ Since the height of the heap is $\lfloor \lg n \rfloor$
- MaxHeapify takes $O(h)$ where h is the height of the node where MaxHeapify is applied.

  $T(n) = O(lgn)$

Problems

- What is the effect of calling MAX-HEAPIFY(A, i) when the element A[i] is larger than its children?
- What is the effect of calling MAX-HEAPIFY(A, i) for $i > A.heap\text{-}size/2$?
- Show that the worst-case running time of MAX-HEAPIFY on a heap of size n is $\Omega(lgn)$.

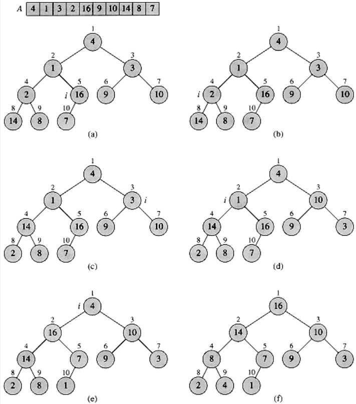
Building a heap

Building Max Heap PROCEDURE

BuildMaxHeap(A)

1. heap-size[A] $\leftarrow$ length[A]
2. for i $\leftarrow$ $\lfloor$length[A]/2$\rfloor$ downto 1
3. do MaxHeapify(A, i)

Build Heap-Example



Running Time of Build-Heap

- Running time: $O(nlgn)$
  This is not an asymptotically tight upper bound

HEAP SORT

- Goal:
  - Sort an array using heap representations

HEAP SORT

- Goal:
  - Sort an array using heap representations

- Idea:
  - Build a max-heap from the array
  - Swap the root (the maximum element) with the last element in the array
  - Discard this last node by decreasing the heap size
  - Call MAX-HEAPIFY on the new root
  - Repeat this process until only one node remains

HEAP SORT PROCEDURE

- Sort by maintaining the as yet unsorted elements as a max-heap.

HEAP SORT PROCEDURE

- Sort by maintaining the as yet unsorted elements as a max-heap.
- Start by building a max-heap on all elements in A.

HEAP SORT PROCEDURE

- Sort by maintaining the as yet unsorted elements as a max-heap.
- Start by building a max-heap on all elements in A.
  Maximum element is in the root, A[1].

HEAP SORT PROCEDURE

- Sort by maintaining the as yet unsorted elements as a max-heap.
- Start by building a max-heap on all elements in A.
  Maximum element is in the root, A[1].
- Move the maximum element to its correct final position.
  Exchange A[1] with A[n].

HEAP SORT PROCEDURE

- Sort by maintaining the as yet unsorted elements as a max-heap.
- Start by building a max-heap on all elements in A.

  Maximum element is in the root, A[1].
- Move the maximum element to its correct final position.

  Exchange A[1] with A[n].
- Discard A[n] it is now sorted.

  Decrement heap-size[A].

HEAP SORT PROCEDURE

- Sort by maintaining the as yet unsorted elements as a max-heap.
- Start by building a max-heap on all elements in A.
  Maximum element is in the root, A[1].
- Move the maximum element to its correct final position.
  Exchange A[1] with A[n].
- Discard A[n] it is now sorted.
  Decrement heap-size[A].
- Restore the max-heap property on A[1..n1].
  Call MaxHeapify(A, 1).

HEAP SORT PROCEDURE

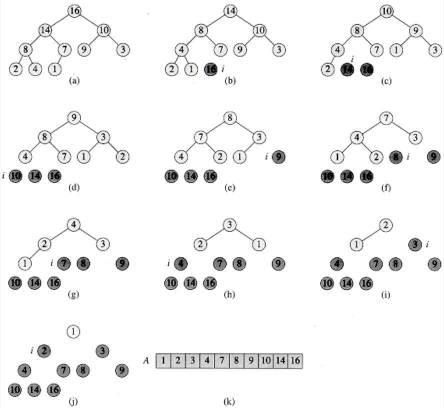
- Sort by maintaining the as yet unsorted elements as a max-heap.
- Start by building a max-heap on all elements in A.
  Maximum element is in the root, A[1].
- Move the maximum element to its correct final position.
  Exchange A[1] with A[n].
- Discard A[n] it is now sorted.
  Decrement heap-size[A].
- Restore the max-heap property on A[1..n1].
  Call MaxHeapify(A, 1).
- Repeat until heap-size[A] is reduced to 2.

HEAP SORT

// Input: A: an (unsorted) array
// Output: A modifed to be sorted from smallest to largest
// Running Time: $O(nlogn)$ *where* $n = length[A]$

```
HeapSort(A)
1. Build-Max-Heap(A)
2. for i ← length[A] downto 2
3.         do exchange A[1] ↔ A[i]
4.                 heap-size[A] ← heap-size[A]  1
5.                 MaxHeapify(A, 1)
```

Heap Sort Example



Running Time for Heap Sort

- Build-Max-Heap takes $O(n)$ and each of the n-1 calls to Max-Heapify takes time $O(lgn)$.

  $T(n) = O(nlgn)$

Running Time for Heap Sort

- Build-Max-Heap takes $O(n)$ and each of the n-1 calls to Max-Heapify takes time $O(lgn)$.
  $$T(n) = O(nlgn)$$
- The heap sorting algorithm uses two procedures : *BUILD-HEAP* and *HEAPIFY*. Thus, total running time Tsort(n) to sort an array of size n is

Problems

- What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?
- Show that the worst-case running time of HEAPSORT is $\Omega(nlgn)$.

Complexity

- To insert a single node in an empty heap is : $O(1)$.
- To insert a single element in a n node heap is : $O(logn)$.
- To insert n elements in a n node heap is : $O(nlogn)$.
- To delete the largest element in a max heap tree : $O(1)$.
- To delete the smallest element in a max heap tree : $O(logn)$.
- To delete n elements in a max heap tree : $O(nlogn)$.
- To create a heap, time complexity will be : $O(nlogn)$.

Complexity

Now to sort the heap tree requires

- Arrange or insert the elements in a form of heap i.e $O(nlogn)$

- Delete the elements one by one after swapping operation $O(nlogn)$
- This gives Heap sort complexity

$$O(nlogn) + O(nlogn)$$
$$2O(nlogn) = O(nlogn).$$

Heap Procedures for Sorting

- MaxHeapify $O(lgn)$

Heap Procedures for Sorting

- MaxHeapify $O(lgn)$
- BuildMaxHeap $O(n)$

Heap Procedures for Sorting

- MaxHeapify $O(lgn)$
- BuildMaxHeap $O(n)$
- HeapSort $O(nlgn)$

SUMMARY

- We can perform the following operations on heaps:
  - MAX-HEAPIFY $O(lgn)$

SUMMARY

- We can perform the following operations on heaps:
  - MAX-HEAPIFY $O(lgn)$
  - BUILD-MAX-HEAP $O(n)$

SUMMARY

- We can perform the following operations on heaps:
  - MAX-HEAPIFY $O(lgn)$
  - BUILD-MAX-HEAP $O(n)$
  - HEAP-SORT $O(nlgn)$

SUMMARY

- We can perform the following operations on heaps:
  - MAX-HEAPIFY $O(lgn)$
  - BUILD-MAX-HEAP $O(n)$
  - HEAP-SORT $O(nlgn)$
  - MAX-HEAP-INSERT $O(lgn)$

SUMMARY

- We can perform the following operations on heaps:
  - MAX-HEAPIFY $O(lgn)$
  - BUILD-MAX-HEAP $O(n)$
  - HEAP-SORT $O(nlgn)$
  - MAX-HEAP-INSERT $O(lgn)$
  - HEAP-EXTRACT-MAX $O(lgn)$

SUMMARY

- We can perform the following operations on heaps:
  - MAX-HEAPIFY $O(lgn)$
  - BUILD-MAX-HEAP $O(n)$
  - HEAP-SORT $O(nlgn)$
  - MAX-HEAP-INSERT $O(lgn)$
  - HEAP-EXTRACT-MAX $O(lgn)$
  - HEAP-INCREASE-KEY $O(lgn)$

SUMMARY

- We can perform the following operations on heaps:
  - MAX-HEAPIFY $O(lgn)$
  - BUILD-MAX-HEAP $O(n)$
  - HEAP-SORT $O(nlgn)$
  - MAX-HEAP-INSERT $O(lgn)$
  - HEAP-EXTRACT-MAX $O(lgn)$
  - HEAP-INCREASE-KEY $O(lgn)$
  - HEAP-MAXIMUM $O(1)$

Building a heap

- Presented To:- Dr. Arshad Aewan
- Presented By:- SK Ahsan
- Reg No:- 813/FBAS/MSCS/F14
- Topic:- Heap Sort(Procedure)
    - My Dreams Are My Own Drems(Me)