

CSC 301: Analysis of Algorithms

Lecture 13: Heaps, Priority Queues

William Rebelsky
[rebelskyw]

Grinnell College

Monday 19th February, 2024

Reminders

- Homework 3 is due Wednesday
- Quiz 3 is due Tonight

Table of Contents

1 Heaps

2 Priority Queues

What is a Heap?

- Nearly complete binary tree e.g. [16,14,10,8,7,9,3,2,4,1]
- Tree height: a tree of a single node is height 0.
- Generally a max heap or min heap
- Generally stored as an array

Why is a heap stored as an array?

- No pointers (less overhead)
- Easy to find children

Heap functions

Following conventions in the text, we will be 1 indexed and do everything in terms of max heaps.

- $\text{Parent}(i) = \lfloor i/2 \rfloor$
- $\text{Left}(i) = 2i$
- $\text{Right}(i) = 2i + 1$
- Max-Heapify $O(\lg n)$
- Build-Max-Heap $O(n)$
- Heapsort $O(n \lg n)$
- Max-Heap: Insert, Extract, Increase Key, Maximum
- Great for Priority Queues

Heap Practice

Array: [33,19,20,15,13,10,2,13,16,12].

- Draw it as a tree
- Is it a max heap?

Heap up vs Heap Down

- Book generally uses heap down updates through Max-Heapify: Given a root of two Max-heaps, compare to both of it's leaves, if at least one is larger, swap with the largest leaf. Recurse (or iterate) until it is in the correct spot
- e.g. $[16,4,10,14,7,9,3,2,8,1] \rightarrow [16,14,10,8,7,9,3,2,4,1]$
- Can also add to a heap using a heap up update: Add an element to a heap by placing it in the last position, compare up until it is the right spot.
- I find it easier to add elements using a heap up method, and to ensure the heap property holds using a heap down approach (as the book does)

Building a heap

- Book method: treat the array as a heap, heap down starting at $\lfloor n/2 \rfloor$ and going backwards $O(n)$
- Alternative (naive) method: treat the array as a heap, heap down starting at 1 and going forwards $O(n \lg n)$
- Runtime:
 - Simple: Each call costs $O(\lg n)$ and there are $O(n)$ such calls $= O(n \lg n)$
 - More complex (see text): $O(n)$
 - This is why data structures matter

Heap Practice

Array: [4,1,3,2,16,9,10,14,8,7]

- Build the max heap using the book method (heap down Max-Heapify)
- Try adding the elements 15,11,5,2,4 in that order

HeapSort

Algorithm 1: Heapsort(A, n)

Input: Array A of size n

Output: in place sorting of Array A

```

1 Build-Max-Heap( $A, n$ )
2 for  $i=n$  down to 2 do
3   |   exchange  $A[1]$  with  $A[i]$ 
4   |    $A.\text{heap-size} = A.\text{heap-size} - 1$ 
5   |   Max-Heapify( $A, 1$ )

```

Heapsort

- Build the heap
- Swap the first and "last" element
- Call max heapify on the first element, excluding the last element
- Repeat until done
- $O(n \lg n)$

Heap Practice

Array: [4,1,3,2,16]

- Run heapsort on the above array

Table of Contents

1 Heaps

2 Priority Queues

Priority Queues

ADT-Abstract Data Type: A set of operations and underlying philosophy

- Underlying philosophy: most important out first
- Operations:
 - $\text{Insert}(S, x, k)$: insert element x with key k into set S
 - $\text{Maximum}(S)$: return element of S with largest key
 - $\text{Extract-Max}(S)$: removes and returns element of S with largest key
 - $\text{Increase-Key}(S, x, k)$: increase the value of x 's key to k
- Corresponding philosophy/operations for
Min-Priority-Queue

Priority Queues

All times below do not account for overhead for mapping objects to indices

Underlying Structure	Insert	Maximum	Extract-Max
Heap	$O(1)$	$O(1)$	$O(\lg n)$
Unsorted Array	$O(1)$	$O(n)$	$O(n)$
Sorted Array	$O(n)$	$O(1)$	$O(1 \text{ or } n)$

Priority Queue Practice

How would you implement the following priority queue operations on a heap compared to an unsorted array?

- Insert
- Max
- Extract Max