EECS3101 notes

Jerry Wu

2023-02-02

Sorting algorithms

As humans, we like order and when things are sorted/organized. Sorting algorithms have many practical applications in the working world like computer graphics, computational geometry, databases, etc. It is a good basis for learning the principles and techniques of algorithm design.
We have a few examples of sorting algorithms from prior knowledge such as::

- Bubble sort
- Selection sort
- Merge sort
- Quick sort

and the list goes on.

Heap sort

"This is a motivating example"-Larry YL Zhang 2023

Heaps are a very efficient data structure to perform operations on (everything is $\log(n)$ and better), so we should learn how they work. Every data structure has an invariant that must be maintained. For example, a sorted list must be sorted.

Binary max heap is a nearly complete binary tree (each level is completely filled and if the last level isn't, all nodes are as far left as possible) that has the max heap property.
Define the indices of an array storing a binary tree as the following:

IMPORTANT: (didn't learn this in EECS2011)

$$LeftChild(i) = 2i$$
$$RightChild(i) = 2i + 1$$
$$Parent(i) = floor(\frac{i}{2})$$

The height of a complete binary tree with n nodes is $\Theta(\log(n))$ as its worst case. So a $\log(n)$ runtime would entail that a tree would be used somewhere.

Max heap property

Simply put, each parent has a greater or equal value than their immediate children (illustration on slide 16). This implies that every node is larger than equal to all its decendants, whereby every subtree of a heap is also a heap.

Operations

> *"I used to spend a lot of time making animations on my old slides"-Larry YL Zhang 2023*

- $Max(Q)$, return the root ($Q(0)$)
- $Insert(Q, x)$, we insert at the leftmost unoccupied position of the bottom row and compare to its parent to swap (if necessary) to satisfy the max heap property.
- $ExtractMax(Q)$, we want to delete the root's value without actually deleting the root itself. So we can bubble it down to the bottom leftmost node (swap with it), and then we can fix it so it satisfies the heap property. We do this by bubbling it down (swap with the **biggest** one) until it is smaller than its parent. Worst case runtime is also $\Theta(\log(n))$ (slide 30)

  > *"If we delete the root, the tree becomes a forest"-Larry YL Zhang 2023*

  > *"Which child is your favourite one?"-Larry YL Zhang 2023*

- $IncreasePriority(Q, x, k)$ Increases the key of node x to k. **LEFT AS AN EXERCISE TO THE READER!**

How do we do heap sort (non descending)? (slide 39)

Simple. We want to extract max ($log(n)$), and do it n times, so the overall runtime is $nlog(n)$. We want to do this all in place (no external data structures).

Take the following array:

$$\{65, 40, 25, 33, 18, 24\}$$

Start by swapping first (65) and last (24). Then decrement the heap size, and finally fix the heap by bubbling down 24. Repeat these steps until array is sorted. **MAKE SURE THAT THE ARRAY IS HEAP ORDERED FIRST BEFORE USING IT!**

"It's almost like magic"-Larry YL Zhang 2023

BuildMaxHeap(A)

Idea 1: We can move everything to an empty array, but this is not in place.

Idea 2: Bubble down until the array is a valid heap. (slide 46 to 55)

"We have a heap built. Now it's time to celebrate!"-Larry YL Zhang 2023
"The end result is 2 lines of code, It's B E A U T I F U L"-Larry YL Zhang 2023
"Maybe I'm getting a bit emotional"-Larry YL Zhang 2023

Worst case analysis of BuildMaxHeap

A complete binary tree with n nodes has $\frac{n}{2}$ leaf nodes. No work will be done on the leaves. Next level we bubble down at most once, next one at most twice, and so on. So we have that::

$$T(n) = \sum_{i=1}^{\log(n)} i\frac{n}{2^{i+1}} \leq n\sum_{i=1}^{+\infty} i\frac{i}{2^{i+1}} = n$$

So this is doable in $O(n)$ time!