CSE 373: Heaps
(Other operations and variations)

Chapter 6

Heaps: Quick Recap

Heaps:
- structure is a complete binary tree
- each node must be smaller than its descendants

- main operations are Insert() and DeleteMin()
- heaps have an compact array-based representation
Other Operations: DecreaseKey()

DecreaseKey() – lowers a node’s value
(while preserving heap ordering)

DecreaseKey(H, 6);

DecreaseKey() – Continued

running time?
**IncreaseKey ()**

IncreaseKey () – raises a node’s value

IncreaseKey (H, \( \leq 6 \));

![Tree Diagram]

**IncreaseKey () – Continued**

running time?
Delete() – removes a node from the heap

Delete(H, 6);

Delete() – Continued

running time?
Let’s Write a Heap Routine...

BuildHeap ()

BuildHeap () – creates a heap from an array

Straightforward Implementation: Insert ()

elements into an empty heap one at a time

running time?
**BuildHeap() – Continued**

*Better Implementation:* Treat input array as a heap and "percolate down" first n/2 values

```
12 5 11 3 10 6 9 4 8 1 7 2
```

Running time?

---

**BuildHeap() – even more**
**BuildHeap() running time**

```
[Diagram of a heap]
```

**MaxHeaps**

*MaxHeaps*: the dual of the Heaps we’ve defined
- support fast `Insert()` and `DeleteMax()` ops
- work exactly the same as (Min)Heaps

Why is `DeleteMax()` expensive on a normal heap?
What’s the running time?
**d-Heaps**

*d-Heaps*: Just like normal heaps but with \(d\) children rather than 2

**Intuition**: tree will be shallower so ops will be faster

**However**…
- more comparisons need to be made when percolating down
- if \(d\) not a power of 2, finding parent/children will be slower

**What about asymptotic running time?**

**Bottom Line**: 4-heaps *may* outperform 2-heaps

---

**Merging Heaps**

How to merge heaps effectively?

*Straightforward method*: copy both arrays into a single array and use `BuildHeap()` running time?

**Advanced methods**:
- pointer-based imbalanced heaps
  - *leftist heaps* – a bit like AVL trees; \(O(\log n)\) merge
  - *skew heaps* – like Splay trees; \(O(\log n)\) amortized ops
  - *binomial queues* – \(O(\log n)\) merge, but \(-O(1)\) insert