CS 310: Heapify and HeapSort

Chris Kauffman

Week 14-2
Logistics

We’re almost done!

<table>
<thead>
<tr>
<th>Day</th>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon</td>
<td>12/5</td>
<td>Heap sort</td>
</tr>
<tr>
<td>Wed</td>
<td>12/7</td>
<td>Review</td>
</tr>
<tr>
<td>Fri</td>
<td>12/9</td>
<td>HW4 Due, <strong>Codefest</strong></td>
</tr>
</tbody>
</table>

Reading: Weiss

- 21: PQ/Binary Heap
- 6.9: PQ Interface

Goals Today

- Heapsort: $O(N \log N)$ sorting
- Heapify: $O(N)$ build heap
Great for building Priority Queues:

<table>
<thead>
<tr>
<th>Op</th>
<th>Worst Case</th>
<th>Avg Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>findMin()</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>insert(x)</td>
<td>$O(\log N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>deleteMin()</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
</tbody>
</table>

What else can one do with binary heaps?
Exercise: Sort data using a Binary Heaps/PQ

- Sort an array of stuff with a PQ/Binary heap
- Define the following method

```java
// Sort an array in place using a priority queue
public static <T extends Comparable>
void heapSort(T data[]){ ... }
```

- T is Comparable
- data[] unsorted
- return: nothing, but data must be sorted after method finishes
- Use pq.insert(x) and pq.deleteMin()
Out of Place Sort Sucks

Initial solution required data duplication
- Copy from data to pq, then back
- Out of place sorting, double memory requirement
For large arrays this hurts
- Want truly in place sorting
- Don’t make copy, avoid $O(N)$ space overhead
- Suggest a solution: no duplication possible?
In-Place Sorting with Heaps: Three Problems

1. 1-indexing of heaps
   - Not a problem: heap root at index 0
   - Formulas for left(i), right(i), parent(i)?

2. Removing min
   - Have a heap, repeatedly Remove the min
   - Where to put it?

Exercise: How do you solve these two problems?

3. Broken Heap
   - Sorting methods start with unsorted array
   - Heap property doesn’t hold - how to fix it?
1. Formulas for Different Root Locations

Root at 1

```java
static int root(){ return 1; }
static int left(int i){ return i*2; }
static int right(int i){ return i*2+1; }
static int parent(int i){ return i / 2; }
```

Root at 0

```java
static int root(){ return 0; }
static int left(int i){ return i*2+1; }
static int right(int i){ return i*2+2; }
static int parent(int i){ return (i-1) / 2; }
```
2. In Place Heap Sort

If we have a heap already...

Space Available

- Remove an element from a heap
- Now open space at end of array (percolate down)
- Put the removed element at end of array
- Repeat until empty

Min and Max Heap

Above process orders the array

- For **Min-Heap** will result in biggest to smallest
- For **Max-Heap** will result in smallest to biggest
Problem 3: Unsorted array to Heap

- How does one go from an unsorted array to a heap ordered array?
Heapify, a.k.a. buildHeap()

Converts an existing array into a heap (!)

```java
public void buildHeap() {
    for (int i = parent(this.size); i >= root(); i-- ){
        this.percolateDown( i );
    }
}
```

Build the heap bottom up, repeated percolateDown(i)

- Start one level above bottom (where for size N heap?)
- Work right to left, low to high
- If small guy is down low, will bubble up
Heapify Example

Level 3: Initial, \texttt{percolateDown([63])}

Level 3: \texttt{percolateDown([45])}, \texttt{percolateDown([12])},
Heapify Example

Level 3: `percolateDown([20])`, Level 2: `percolateDown([21])`,

Level 2: `percolateDown([47])`, Level 1: `percolateDown([92])`,
Build me a Heap as Quick as You Can

How complex is Heapify / buildHeap()? 
▶ Discuss with a neighbor

Try
▶ How many small moves versus big moves?
▶ Derive an expression for the worst possible number of moves
Complexity of Heapify

Heap size $n$, height $h$, assume complete (?)

Measure level from bottom

- Level 1 is bottom, has $2^{h-1}$ nodes,
- Level 2 is second from bottom, has $2^{h-2}$ nodes
- Level $i$ is $ith$ form bottom, has $2^{h-i}$ nodes
- Level $h$ is root, has $2^{h-h} = 1$ node

Each level $i$ node can move $i$ down so

$$\text{moves} = \sum_{i=1}^{h} i \times 2^{h-i} = \sum_{i=1}^{\log_2 n} i \times 2^{\log_2 n - i}$$

$$= \sum_{i=1}^{\log_2 n} i \times \frac{2^{\log_2 n}}{2^i} = n \sum_{i=1}^{\log_2 n} \frac{i}{2^i}$$

$$\leq n \times 2 = O(n)$$

because

$$\sum_{i=1}^{\infty} \frac{i}{2^i} \to 2$$
Summary of Heap Sort

Input: Array a
Output: a is sorted

Build Max Heap on a
for i=0 to length-1 a
tmp = findMax(a)
removeMax(a)
a[length-i-1] = tmp
done