CS 2100: Data Structures & Algorithms 1

Priority Queues

Heapsort

Dr. Nada Basit // basit@virginia.edu

Spring 2022

Friendly Reminders

- The University updated the mask policy. As per my Request on Mar 28, 2022 (see Collab), I would greatly appreciate if you would do me a kind favor by **continuing to wear your masks** in CS 2100 (Ridley G008). I know it is a lot to ask, and it is **voluntary**, but I appreciate your understanding.
- If you forget your mask (or mask is lost/broken), I have a few available
  - Just come up to me at the start of class and ask!
- No eating or drinking in the classroom, please
- Our lectures will be **recorded** (see Collab) – please allow 24-48 hrs to post
- If you feel **unwell**, or think you are, please stay home
  - *We will work with you!*
  - At home: eye mask instead! Get some rest ☺

Heapsort

Another sorting algorithm in the better complexity class

(log-linear complexity)

{Reminder} How to Sort?

- Some "straightforward" sorting algorithms
  - Insertion Sort, Selection Sort, Bubble Sort
  - **Each is $O(n^2)$**

- More efficient sorting algorithms
  - Quicksort, Mergesort, Heapsort
  - **Each is O(n log n)**

Best Sorts are O(n log n)

Heapsort

- **Basic idea:** Use a **heap** to sort a list of numbers
  - Two primary ways to do this
  - One is *easier*, but not in-place
  - The other is in-place

Heapsort: Solution 1

Given a list of ***n*** unsorted elements…

1. Instantiate a **heap** (**minHeap** or **maxHeap** depending on which way you want to sort!)
2. Insert ***n*** elements
3. Remove ***n*** elements Done ☺

Each one has an insertion time of log n, and then a removal time of log n

Hence **Θ(n log n)**

But it's **not** a *stable* sort, so it's not used as often as **mergesort**

Heapsort: Solution 2 (in-place)

Overall idea: For an array of size ***n***, use the array from position 1 through heap_size as a **maxHeap**, and from position heap_size+1 to n-1 as a **sorted list.**

[index 1 .. heap_size] MAXHEAP	[index heap_size+1 .. n-1] SORTED LIST

1. [step 1] Turn the unsorted array into a **maxHeap**
2. [step 2] Remove max (`poll()`) ***each*** element one at a time.
   - Move the element that is removed to the back of the array so it is ***in its sorted position***.

Notes: *Need to deal with the indexing from 1 vs 0 issue.*

We are using a maxHeap to sort in ascending order (small -> large)

Heapify!

- **[step 1]**: Given an unsorted array, turn it into a **maxHeap**.
  - How? Start at the back of the array (i.e., the leaves)
  - For each index i, call `percolateDown(i)`
  - This turns array from i to n into a max heap
  - What to do with index 0?

- Let's heapify the following:

12 8 3 9 10 7 4 22

Let's Heapify: 12 8 3 9 10 7 4 22

Let's Draw a maxHeap!

- **The result is** [performing level order / heap form]: [

]

Heapsort: Poll()

- **[step 2]**: Call `Poll()` – same as before, except:
  - **Swap** root with last element in heap, then `percolateDown()`

- Let's step through it!

Heapsort: Poll()

- poll(): *("remove 22 and save at end of list")* **22** swaps with **8**; 8 percolates down:

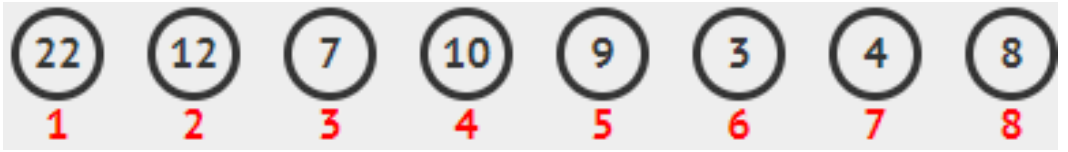
  12 10 7 8 9 3 4 22

- poll(): *("remove 12 and save at end of list")* **12** swaps with **4**; 4 percolates down:

  10 9 7 8 4 3 12 22

- poll(): *("remove 10 and save at end of list")* **10** swaps with **3**; 3 percolates down:

  9 8 7 3 4 10 12 22



Heapsort: Poll()

- poll(): *("remove 10 and save at end of list")* **10** swaps with **3**; 3 percolates down:

  9 8 7 3 4 10 12 22

- poll(): *("remove 9 and save at end of list")* **9** swaps with **4**; 4 percolates down:

  8 4 7 3 9 10 12 22

- poll(): *("remove 8 and save at end of list")* **8** swaps with **3**; 3 percolates down:

  7 4 3 8 9 10 12 22

- poll(): *("remove 7 and save at end of list")* **7** swaps with **3**; 3 percolates down:

  4 3 7 8 9 10 12 22

- poll(): *("remove 4 and save at end of list")* **4** swaps with **3**; 3 percolates down:

  3 4 7 8 9 10 12 22

*Technically one more poll() to do, but not necessary. Our list is sorted in **ascending** order!*

Heapsort: Analysis

- **Heapify():**
  - Basic heap operation, Heapify, runs **O(log n)**
    - Heap has log n levels, and the element being shifted moves down one level of the tree after a constant amount of time
  - Based on this: we need to apply Heapify **roughly n/2 times** *(to each of the internal nodes)*
  - Start at node Math.floor(n/2), call percolateDown()
  - `log(n) * (n/2) = Theta(nlogn)`
- **Poll()**
  - invoke ***n*** times, each one is `log(n)`
  - `n*log(n)`
- **Total (Heapsort):** heapifying + polling ➔ (n/2)log(n) + nlog(n) = **Theta(nlogn)**