CSC 212: Data Structures and Abstractions

10: Heapsort

Prof. Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Spring 2025

Practice

- Build a max-heap **from an existing array** (vector)?
  - ✓ show the algorithm
  - ✓ analyze the computational cost
- Can the same task be done in linear time?
  - ✓ buildHeap

buildHeap

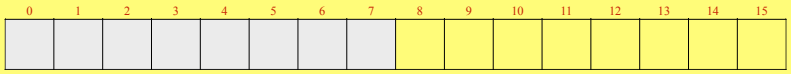
buildHeap

- Algorithm
  1. initialize the heap with the given array (some elements may violate the heap property)
  2. start index **i** at the last non-leaf node => **parent(n-1)**
  3. perform **downHeap** on node **i**
  4. decrement **i**
  5. repeat 3-4 until the root node is reached

https://visualgo.net/en/heap

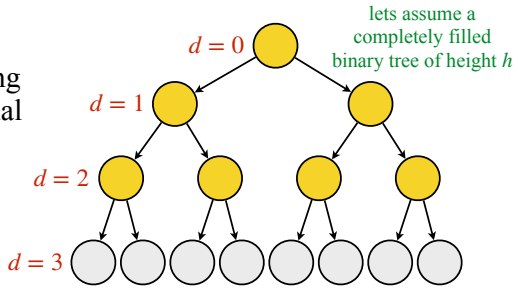
Practice

- Build a max-heap from the following array using BuildHeap
  - 10 42 25 13 17 33 45 50



Analysis

- Total cost
  - sum of the costs of calling **downHeap** for all internal nodes (sum of heights)



$$T(n) = 1(h) + 2(h-1) + 4(h-2) + \ldots + 2^h(0)$$

$$= \sum_{i=0}^{h} 2^i(h-i)$$

$$= h\sum_{i=0}^{h} 2^i - \sum_{i=0}^{h} i2^i = \ldots \ \ldots \ \ldots = \Theta(n)$$

Performance (priority queues)

Method	Unsorted Array	Sorted Array	Binary Heap
Enqueue	O(1)	O(n)	O(log n)
Dequeue	O(n)	O(1)	O(log n)
Max	O(n)	O(1)	O(1)
Size	O(1)	O(1)	O(1)
IsEmpty	O(1)	O(1)	O(1)
Enqueue N	O(n)	$O(n^2)$	O(n)**

(**) assuming we use **buildHeap**

Practice

- What is this function doing?
  - what is the time complexity?
    - depends on the running time of the priority queue operations

```
void foo(std::vector<int>& vec) {

    int n = vec.size();
    std::priority_queue<int> pq;

    for (auto& elem : vec)
        pq.push(elem);

    while (!pq.empty()) {
        vec[--n] = pq.top();
        pq.pop();
    }

}
```

heapSort

heapSort

- Algorithm
  1. use **buildHeap** to create a max-heap from the input array
  2. swap the root with the last element in the array
  3. "remove" the last element from the array (decrement the size)
  4. apply **downHeap** to the new root to restore the heap property
  5. repeat 2-3-4 until the max-heap is "empty"

https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Heapsort.html

Practice

- Apply heapSort to the following array
  - 10 42 25 13 17 33 45 50 20

Analysis

- Step 1
  - cost of **buildHeap** => $\Theta(n)$
- Steps 2-5
  - we apply **downHeap** O(n) times
  - cost => $\Theta(n \log n)$
- Heapsort cost => $\Theta(n \log n)$
  - same asymptotic performance as the example provided before
    - however, this algorithm can run **in-place** (within the original array)
    - it avoids the overhead of copying the elements to/from the priority queue