CSC 212: Data Structures and Abstractions

10: Heapsort

Prof. Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Fall 2025

Practice

- How to build a max-heap **from an existing array** (vector)?
  - ✓ show the algorithm
  - ✓ analyze the computational cost

buildHeap

buildHeap (in linear time)

- Given

  (some elements may violate the heap property)

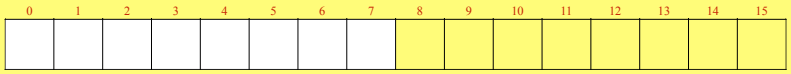
  - ✓ an **unsorted array A of n elements**
- Algorithm
  1. **set** `idx` to the last non-leaf node's position: `parent(n-1)`
  2. **while** `idx >= 0`
     - perform `downHeap` on node `idx`
     - decrement `idx`

Why $\Theta(n)$? starting from the bottom-up means: leaves (second half of array) are already valid heaps, lower nodes have shorter downHeap distances, only the root might bubble down through the full height.

https://visualgo.net/en/heap

Practice

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

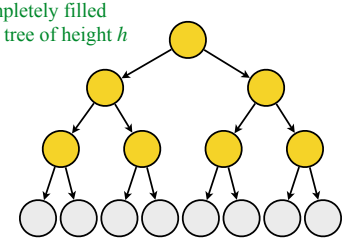



```
set idx to parent of last node
while idx >= 0
   perform downHeap on node idx
   decrement idx
```


Analysis

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

lets assume a completely filled binary tree of height h



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

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

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

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

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/Min	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 the output of this code?
  - also indicate the cost of each line

```
void printHeap(std::priority_queue<int> heap) {
    while (!heap.empty()) {
        std::cout << heap.top() << " ";
        heap.pop();
    }
    std::cout << std::endl;
}

int main() {
    std::priority_queue<int> heap1;  // max-heap by default
    heap1.push(3);
    heap1.push(1);
    heap1.push(4);
    heap1.push(2);
    std::cout << heap1.top() << std::endl;

    std::vector<int> data = {-3, -1, -4, -2};
    std::priority_queue<int> heap2(data.begin(), data.end());
    std::cout << heap2.top() << std::endl;

    printHeap(heap1);
    printHeap(heap2);

    return 0;
}
```

Practice

- Given an array of points in 2D space and an integer k, design an algorithm to return the k points closest to the origin (0,0) — using the Euclidean distance
- Sample input:
  - k = 2
  - points = {{1,3}, {-2,2}, {5,8}, {0,1}}
- Output:
  - {{0,1}, {-2,2}}

Practice

- What is this function doing? — assume a max-pq
  - what is the time complexity?

```
void foo(std::vector<int>& vec) {
    int n = vec.size();
    std::priority_queue<int> pq;

    for (int elem : vec)
        pq.push(elem);

    while (! pq.empty()) {
        n = n - 1;
        vec[n] = pq.top();
        pq.pop();
    }
}
```


heapSort

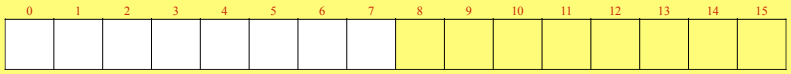
heapSort

- Given (some elements may violate the heap property)
  - an **unsorted array *A* of *n* elements**
- Algorithm
  1. call `buildHeap` on *A* (creates max-heap)
  2. **set** `size` to *n*
  3. **while** `size > 1`
     - swap `A[0]` with `A[size-1]` (move max to its final position)
     - decrement `size` (exclude sorted element)
     - perform `downHeap` on the root

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




```
call buildHeap on A
set size to n
while size > 1
   swap A[0] with A[size-1]
   decrement size
   perform downHeap on the root
```


Analysis

- Total cost
  - cost of **buildHeap** => $\Theta(n)$
  - we apply **downHeap** $\Theta(n)$ times, cost => $\Theta(n \log n)$
  - $T(n) = \Theta(n) + \Theta(n \log n)$

- Heapsort cost => $\Theta(n \log n)$
  - same asymptotic performance as a naive example (`foo`)
    - however, this algorithm can run **in-place** (within the original array)
    - it avoids the overhead of copying the elements to/from the priority queue