

# 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

2

## buildHeap

## buildHeap

- Algorithm
    1. initialize the heap with the given array
    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
- (some elements may violate the heap property)

<https://visualgo.net/en/heap>

4

## Practice

- Build a max-heap from the following array using BuildHeap

✓ 10 42 25 13 17 33 45 50

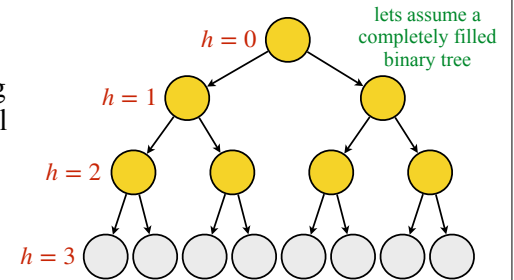
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

5

## Analysis

- Total cost

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



$$\begin{aligned}
 T(n) &= 1(h) + 2(h-1) + 4(h-2) + \dots + 2^h(0) \\
 &= \sum_{i=0}^h 2^i(h-i) \\
 &= h \sum_{i=0}^h 2^i - \sum_{i=0}^h i2^i = \dots \dots \dots = \Theta(n)
 \end{aligned}$$

6

## Performance

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

7

## 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();
    }
}
    
```

8

# 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://visualgo.net/en/heap>

10

## Practice

### Apply heapSort to the following array

✓ 10 42 25 13 17 33 45 50 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

11

## Analysis

### Step 1

✓ cost of **buildHeap**  $\Rightarrow \Theta(n)$

### Steps 2-5

✓ we apply **downHeap**  $O(n)$  times

✓ cost  $\Rightarrow \Theta(n \log n)$

• Heapsort cost  $\Rightarrow \Theta(n \log n)$

12