KAZAKH KBTU BRITISH
TECHNICAL
UNIVERSITY

LECTURE 5 – PRIORITY QUEUE & HEAP

Prepared by: Izbassar Assylzhan

Course: Algorithms & Data Structures – Fall 2025

LECTURE 5

KAZAKH KBTU BRITISH TECHNICAL UNIVERSITY

In this lecture, we'll cover extension of queue as priority queue, that gives some weights to stored data. We'll see how heaps provide an efficient way to implement priority queues.

TOPICS WE'LL COVER:

- Priority Queue
- Introduction to Heap
- Heapify Up
- Extract top element
- Practice Problems

GOALS FOR THIS LECTURE:

- Understand how priority queues differ from stacks and regular queues.
- Explain and implement heap-based operations such as insertion and extraction.
- Use C++ STL priority_queue and solving some tasks using it.

PRIORITY QUEUE

A priority queue is a data structure that **stores elements** along with **their priorities** and removes the element **with the highest** (or lowest) priority first, **rather than** following the **FIFO order** of a regular queue.

Operations:

- **insert(x)** – adding the element to priority queue;
- **top()** – getting the top-most element from data structure;
- **extractMax() / extractMin()** – removing the top-most element according to priority of the data structure.

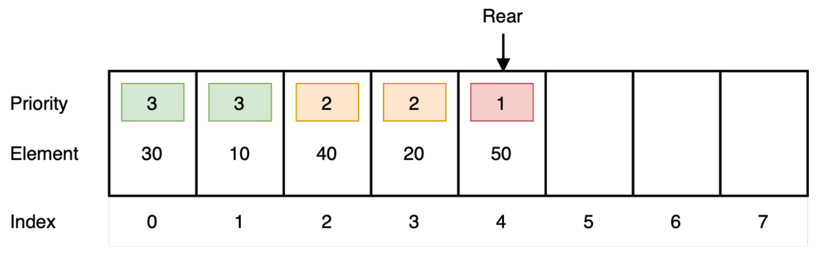


Figure 1 – Priority Queue storage [1]

[1] https://simplerize.com/data-structures/priority-queue

INTRODUCTION TO HEAP

A **binary tree** used to implement priority queues efficiently. There are two kind of heaps as max-heap & min-heap. The difference between two the first one stores parent element as more than children, and latter one less than their children.

Properties:

1. Complete binary tree
2. Heap property (Max-Heap or Min-Heap)

Array representation:

- Left = 2*i + 1
- Right = 2*i + 2
- Parent = (i - 1) / 2

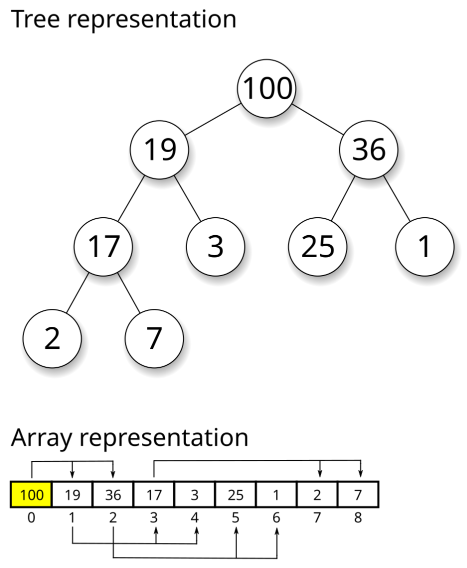


Figure 2 – Heap representation [2]

[2] https://en.wikipedia.org/wiki/Heap_%28data_structure%29

HEAPIFY UP

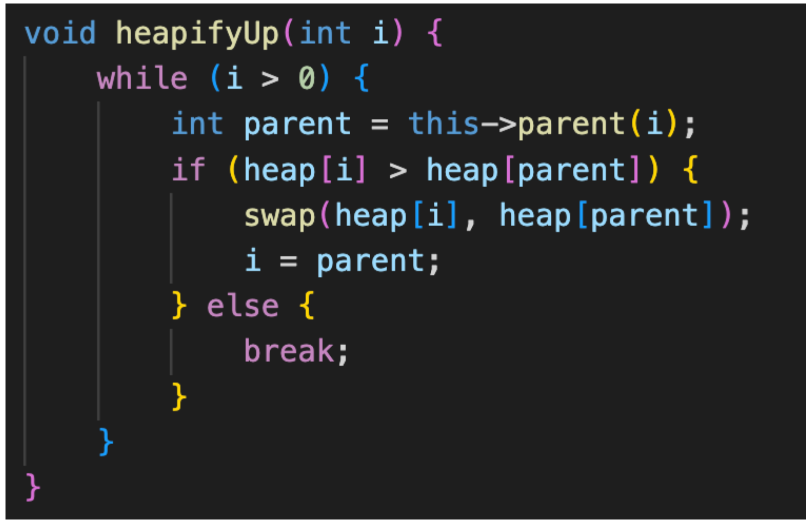
Heapify-Up is used after inserting a new element into the heap.

How it works (Max-Heap):

1. Insert the new value at the end.
2. Compare it with its parent.
3. If it's larger than the parent → swap them.
4. Repeat until:
    - The parent is larger, or
    - The element becomes the root.

When a new value is added, it is first placed at the end of the array. This may break the heap property, so we move it upward until the structure is valid again.

Time Complexity: O(log n)



```
void heapifyUp(int i) {
    while (i > 0) {
        int parent = this->parent(i);
        if (heap[i] > heap[parent]) {
            swap(heap[i], heap[parent]);
            i = parent;
        } else {
            break;
        }
    }
}
```

Figure 3 – Implementation of heapify-up

EXTRACT TOP ELEMENT

Priority Queue

Introduction to Heap

Heapify Up

Extract top element

Practice Problems

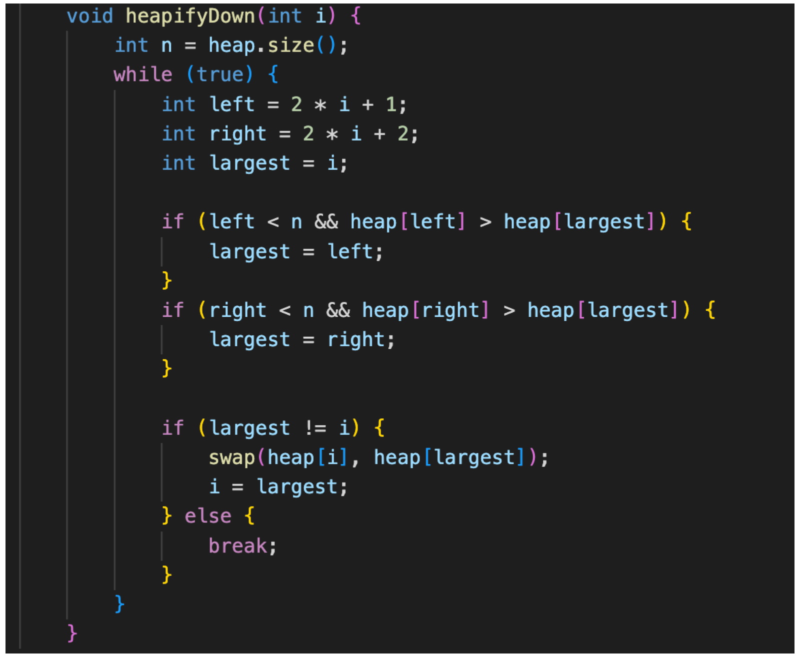
Heapify-Down is used after removing the root (max or min element).

When you remove the root, you replace it with the last element in the heap. This new value may violate the heap property, so you move it downward to restore order.

How it works (Max-Heap):

1. Replace the root with the last element.
2. Compare it with its two children.
3. Swap it with the larger child.
4. Repeat until:
   - Both children are smaller,
   - or You reach a leaf.

Time Complexity: O(log n)



```
void heapifyDown(int i) {
    int n = heap.size();
    while (true) {
        int left = 2 * i + 1;
        int right = 2 * i + 2;
        int largest = i;

        if (left < n && heap[left] > heap[largest]) {
            largest = left;
        }
        if (right < n && heap[right] > heap[largest]) {
            largest = right;
        }

        if (largest != i) {
            swap(heap[i], heap[largest]);
            i = largest;
        } else {
            break;
        }
    }
}
```

Figure 4 – Implementation of heapify-down

PRACTICE PROBLEMS

We have two tasks to solve:

- 2335. Minimum Amount of Time to Fill Cups;
- 703. Kth Largest Element in a Stream.

Q & A