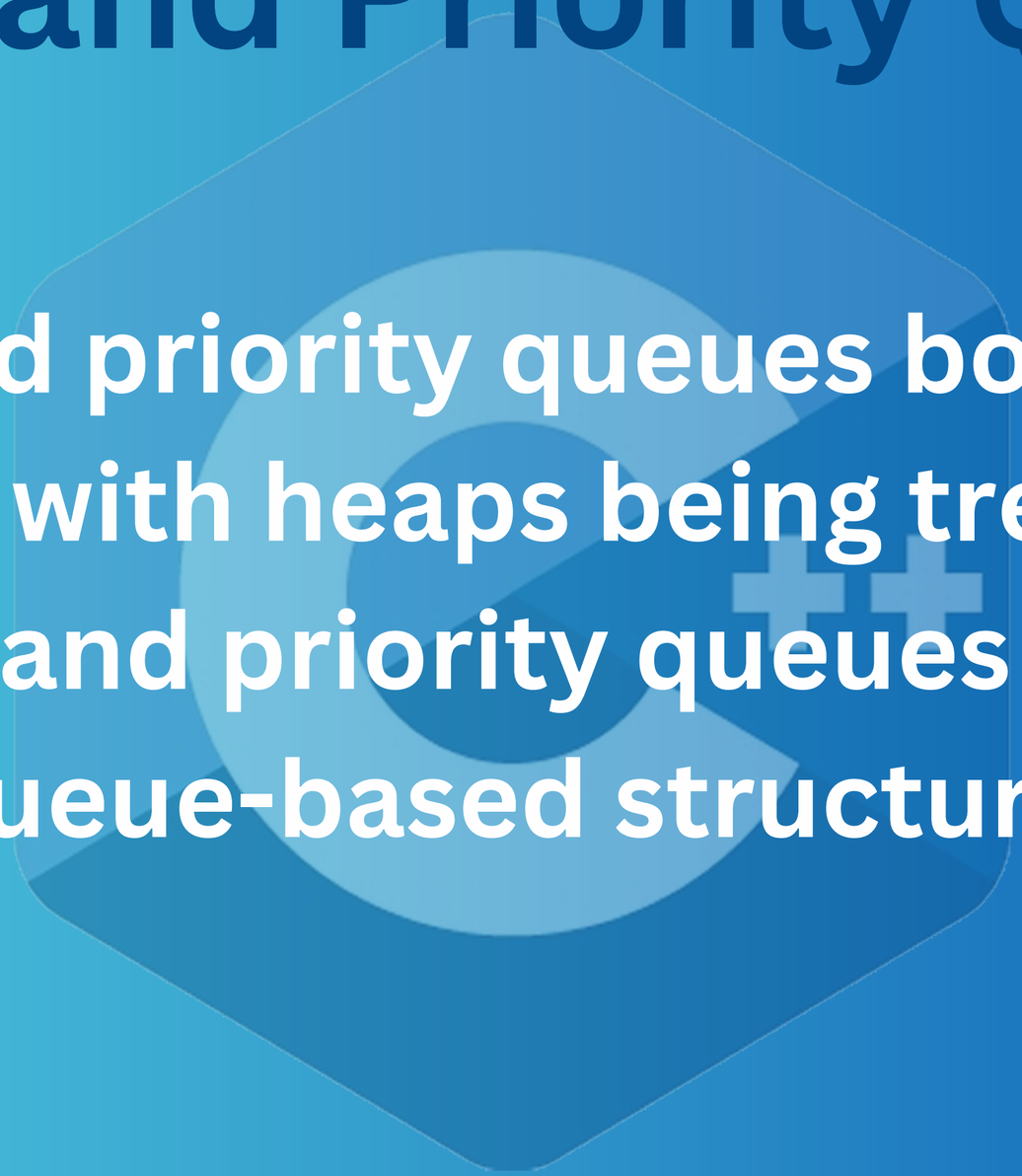
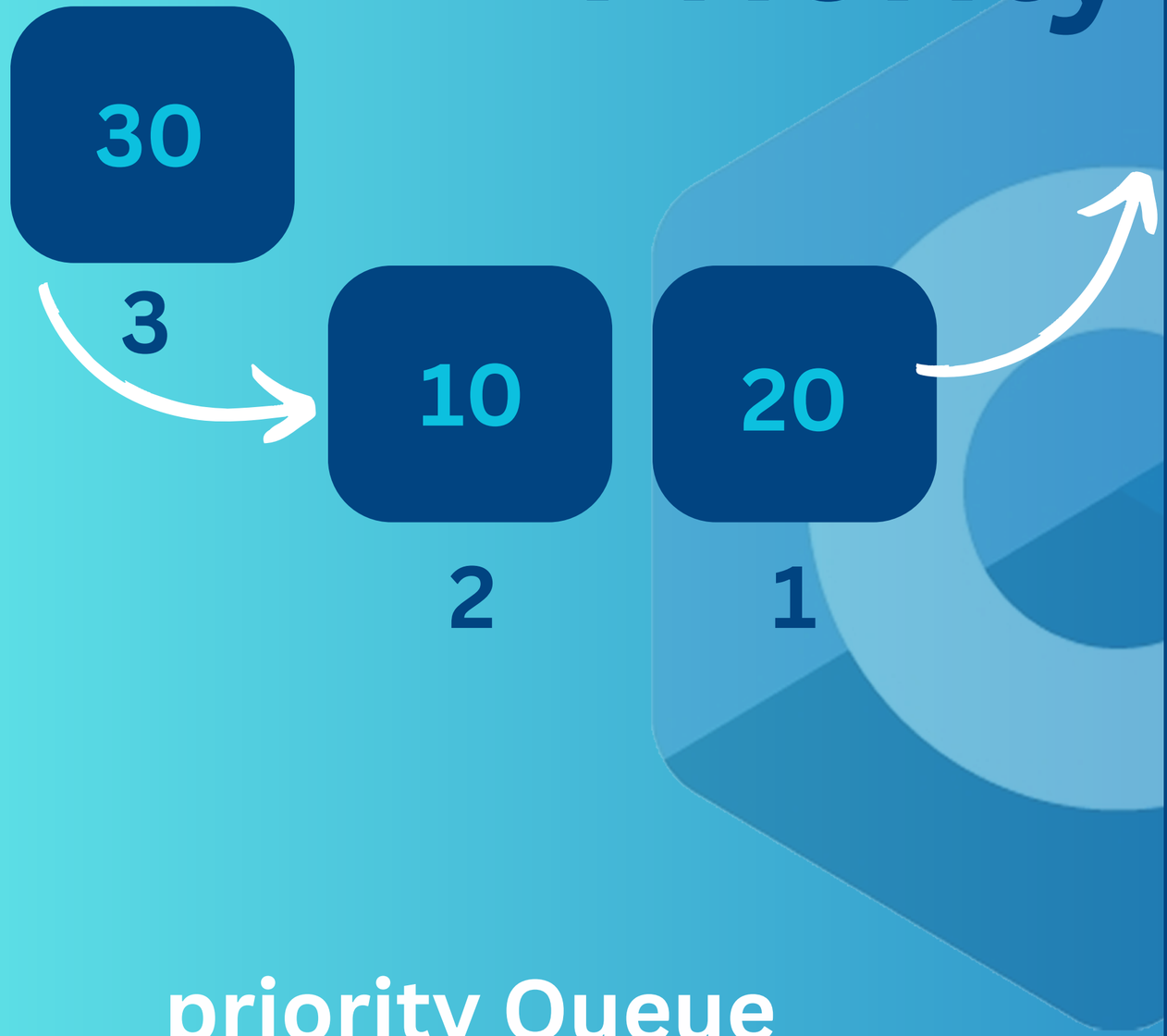
Heaps and Priority Queues



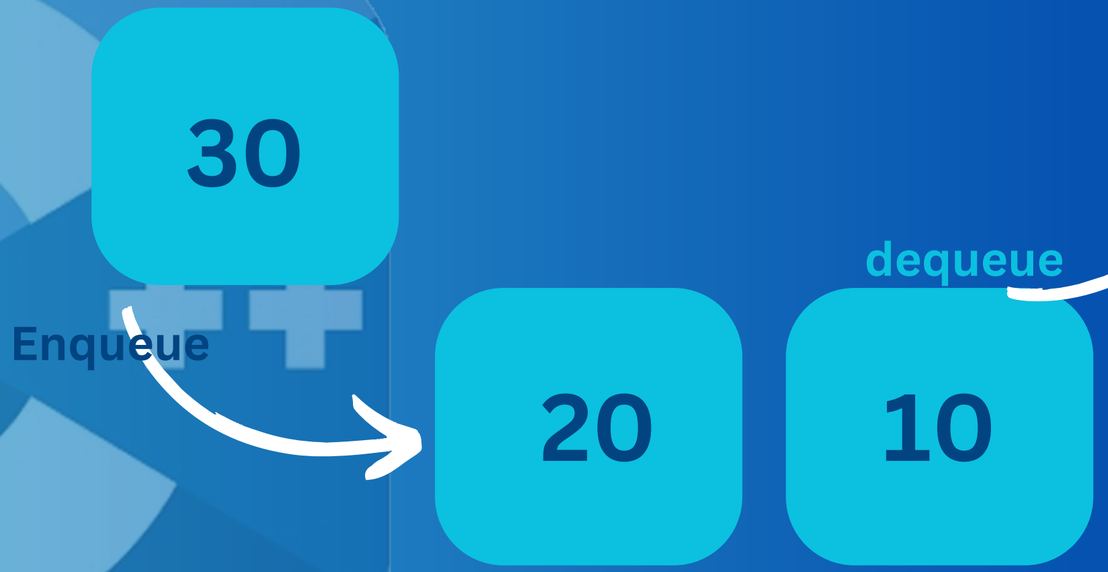
Heaps and priority queues both handle priority, with heaps being tree-based structures and priority queues following a queue-based structure.

Priority Queues



priority Queue

FIFO(First in First Out)



simple Queue

p_queue.cpp U

code > p_queue.cpp > main()

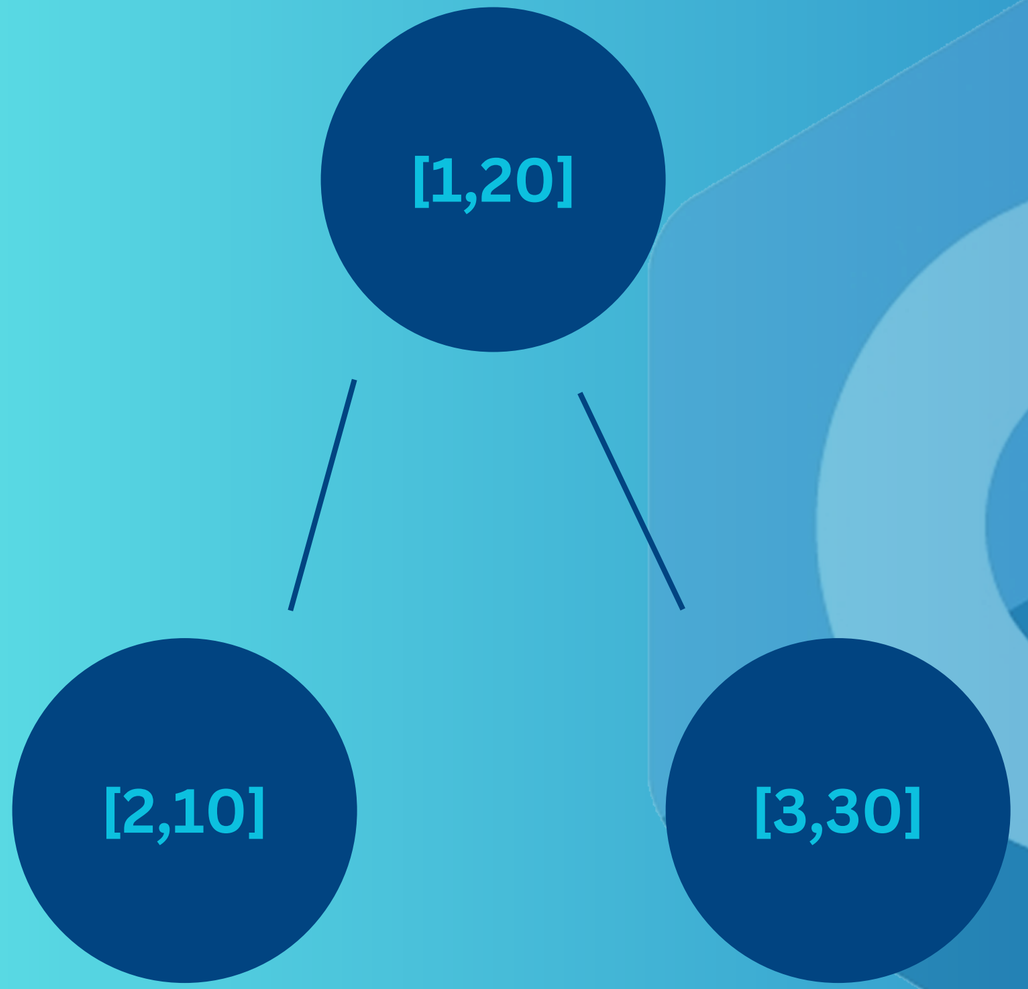
```
33  void print(Node* head) {
34      while (head) {
38  }
39
40  int main() {
41      Node* pq = nullptr;
42      enqueue(pq, 10, 2);
43      enqueue(pq, 20, 1);
44      enqueue(pq, 30, 3);
45
46      print(pq);
47
48  }
```

PROBLEMS DEBUG CONSOLE OUTPUT PORTS SERIAL MONITOR COMMENTS TERMINAL

```
D:\Documents\GitHub\Introduction-to-Data-Structure\code>
```

powershel
cmd

Heap



pushing

after popping

p_queue.cpp U | heap.cpp U

code > heap.cpp > main()

```
73  int main() {
74      Heap pq;
75
76      pq.push(10, 2);
77      pq.push(20, 1);
78      pq.push(30, 3);
79
80      pq.print();
81
82      cout << endl;
83      cout<< "after poping left leave" << endl;
84      cout<< endl;
85
86      auto top = pq.pop();
87      pq.print();
88  }
```

PROBLEMS DEBUG CONSOLE OUTPUT PORTS SERIAL MONITOR COMMENTS TERMINAL

```
D:\Documents\GitHub\Introduction-to-Data-Structure\code>g++ heap.cpp -o heap

D:\Documents\GitHub\Introduction-to-Data-Structure\code>heap.exe
data: 20, priority: 1
data: 10, priority: 2
data: 30, priority: 3

after poping left leave

data: 10, priority: 2
data: 30, priority: 3

D:\Documents\GitHub\Introduction-to-Data-Structure\code>
```

powershell
cmd

Priority Queue

```cpp
// priority queue will priorite the queue based on its priority
#include <iostream>
using namespace std;

struct Node {
    int data;
    int priority;
    Node* next;

    Node(int d, int p) : data(d), priority(p) {}
};
```

```cpp
void enqueue(Node*& head, int data, int priority) {
    Node* n = new Node(data, priority);

    // queue is empty or the new node has the highest priority
    if (!head || priority < head->priority) {
        n->next = head;
        head = n;
        return;
    }

    // correct position to insert the new node
    Node* current = head;
    while (current->next && current->next->priority <= priority) {
        current = current->next;
    }

    n->next = current->next;
    current->next = n;
}
```

```
33  void print(Node* head) {
34      while (head) {
35          cout << "data: " << head->data << ", priority: " << head->priority << endl;
36          head = head->next;
37      }
38  }
39
40  int main() {
41      Node* pq = nullptr;
42      enqueue(pq, 10, 2);
43      enqueue(pq, 20, 1);
44      enqueue(pq, 30, 3);
45
46      print(pq);
47
48  }
```

Text Based Tutorials

https://raufjatoi.github.io/Introduction-to-Data-Structure/

This is it from me :)