

# PRIORITY QUEUES

## DATA STRUCTURE SELECTION

---

# Final exam

- About the final exam: <https://ucsb-cs24-s18.github.io/exam/e03/>
- Review session: Tomorrow (Tuesday- June 5)
  - Phelps 2510
  - Session one: 2p - 3p
  - Session two: 3p -4p
  - Both sessions will be identical
- Diba's office hours and extra hours:
  - Thursday : 11am to 1pm
  - Friday: 3p to 5pm

# Goals of this class

- Object oriented programming
- Data structures
  - Arrays
  - Dynamic Arrays
  - Linked lists (single and doubly linked)
  - Stacks
  - Queues
  - Binary Search Trees
  - Heaps (also known as priority queue)
- Be able to implement each of these data structures in C++
- Be able to use the C++ STL implementations of these data structures in your algorithms.
- Be able to select the right data structure for your problem by knowing:
  - Operations supported by the data structure
  - Big-O running time of these operations (not from memory but through analysis)

## std::priority\_queue (STL's version of heap)

**A C++ `priority_queue` is a generic container, and can store any data type on which an ordering can be defined: for example `ints`, `structs` (`Card`), `pointers` etc.**

```
priority_queue<int> pq;
```

### Methods:

- \* `push()`     //insert
- \* `pop()`     //delete max priority item
- \* `top()`     //get max priority item
- \* `empty()`    //returns true if the priority queue is empty

- You can extract object of highest priority in  $O(\log N)$
- To determine priority: objects in a priority queue must be comparable to each other

# STL Heap implementation: Priority Queues in C++

By default, if  $a < b$ ,  $b$  has higher priority than  $a$

**What is the output of this code?**

```
priority_queue<int> pq;  
pq.push(10);  
pq.push(2);  
pq.push(80);  
cout<<pq.top( );  
pq.pop( );  
cout<<pq.top( );  
pq.pop( );  
cout<<pq.top( );  
pq.pop( );
```

**A. 10 2 80**

**B. 2 10 80**

**C. 80 10 2**

**D. 80 2 10**

**E. None of the above**

# Comparison class

- We call tell priority\_queue how to prioritize items using a *comparison class*
- Comparison class: A class that implements a function call operator.

```
template <class T>
```

```
class less{
```

```
    bool operator()(T& a, T & b) const {
```

```
        return a<b;
```

```
    }
```

```
};
```

The default `std::less` is a comparator class that provides priority comparisons

```
less<int> ls;
```

```
if(ls(a,b))
```

```
    cout<<a <<"has less priority over "<< b;
```

## std::priority\_queue template arguments

The template for priority\_queue takes 3 arguments:

```
template <
    class T,
    class Container= vector<T>,
    class Compare = less <T>
> class priority_queue;
```

- The first is the type of the elements contained in the queue.
- If it is the only template argument used, the remaining 2 get their default values:
  - a **vector<T>** is used as the internal store for the queue,
  - **less is a comparator** class that provides priority comparisons

# Selecting data structures

Application: Sort an array of  $N$  integers



# Data structure Comparison

	Insert	Search	Min	Max	Delete min	Delete max	Delete (any)
Sorted array	$O(N)$	$O(\log N)$	$O(1)$	$O(1)$	$O(N)$ if ascending order, else $O(1)$	$O(1)$ if ascending, else $O(N)$	$O(\log N)$ to find, $O(N)$ to delete
Unsorted array	$O(1)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$
Sorted linked list (assume access to both head and tail)	$O(N)$	$O(N)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(N)$ to find, $O(1)$ to delete
Unsorted linked list	$O(1)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$ to find, $O(1)$ to delete
Stack	$O(1)$ - only insert to top	Not supported	Not supported	Not supported	Not supported	Not supported	$O(1)$ - Only the element on top of the stack
Queue	$O(1)$ - only to the rear of the queue	Not supported	Not supported	Not supported	Not supported	Not supported	$O(1)$ - only the element at the front of the queue
BST (unbalanced)	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$
BST (balanced)	$O(\log N)$	$O(\log N)$	$O(\log N)$	$O(\log N)$	$O(\log N)$	$O(\log N)$	$O(\log N)$
Min Heap	$O(\log N)$	Not supported	$O(1)$	Not supported	$O(\log N)$	Not supported	$O(\log N)$
Max Heap	$O(\log N)$	Not supported	Not supported	$O(1)$	Not supported	$O(\log N)$	$O(\log N)$