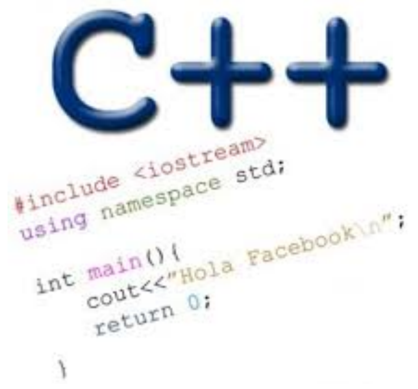
FINAL WRAP UP

Problem Solving with Computers-II



```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```

std::priority_queue template arguments

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

The template for priority_queue takes 3 arguments:

1. Type elements contained in the queue.
2. Container class used as the internal store for the priority_queue, the default is **vector<T>**
3. Class that provides priority comparisons, the default is **less**

Comparison class: A class for comparing objects

```
template <class T>
class myCompare{
        bool operator()(T& a, T& b) const {
                return a > b;
        }
};

int main(){
    myCompare<int> cmp;
    cout<<cmp(20, 10)<<endl;
}
```

If cmp(x, y) returns true, priority queue will interpret this as:

x has __________ priority than y

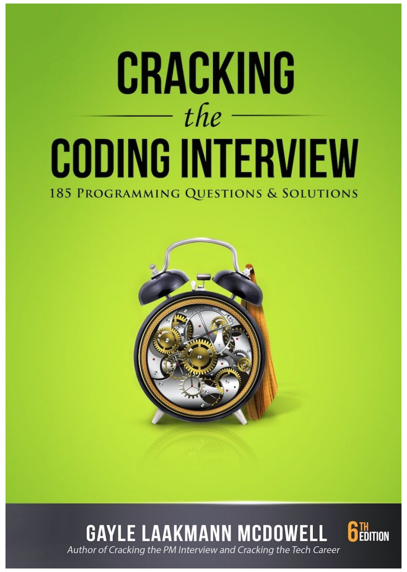
Which element will be at the top of such a priority queue?

std::priority_queue template arguments

```
//Template parameters for a max-heap
priority_queue<int, vector<int>, std::less<int>> pq;

//Template parameters for a min-heap
priority_queue<int, vector<int>, std::greater<int>> pq;
```

Tips for Technical Interviews and Final

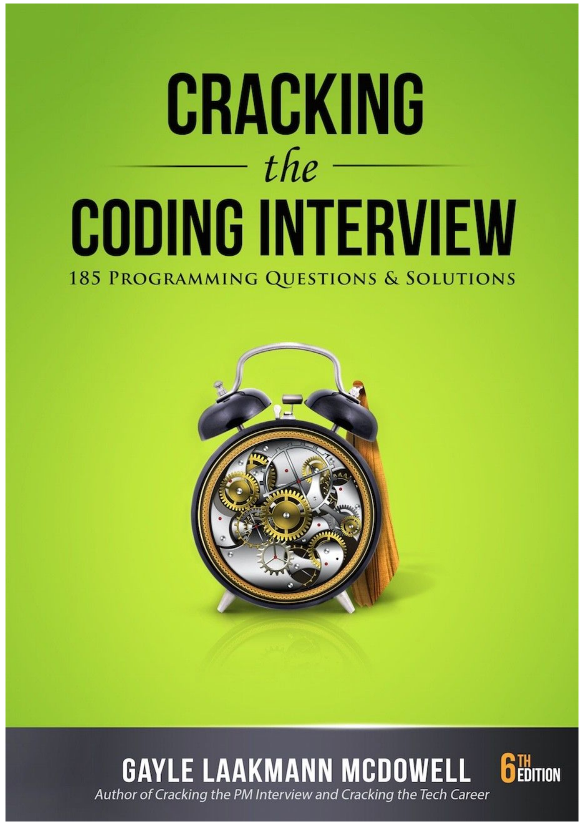

CRACKING
the
CODING INTERVIEW
185 PROGRAMMING QUESTIONS & SOLUTIONS
GAYLE LAAKMANN MCDOWELL
6TH EDITION
Author of Cracking the PM Interview and Cracking the Tech Career

1. Listen carefully
2. Draw an example
3. State the brute force or a partially correct solution
   - then work to get at a better solution
4. Optimize:
   - Make time-space tradeoffs to optimize runtime
   - Precompute information: Reorganize the data e.g. by sorting
5. Solidify your understanding of your algo before diving into writing code.
6. Start coding!

Interview practice!

Write a ADT called minStack that provides the following methods

- push() // inserts an element to the “top” of the minStack
- pop() // removes the last element that was pushed on the stack
- top () // returns the last element that was pushed on the stack
- min() // returns the minimum value of the elements stored so far


CRACKING the CODING INTERVIEW
185 PROGRAMMING QUESTIONS & SOLUTIONS
GAYLE LAAKMANN MCDOWELL
6TH EDITION
Author of Cracking the PM Interview and Cracking the Tech Career

Practice the interview tips:

- Draw/solve a small example! (2 min)
  - Think of the most straightforward approach (1 min)
  - Evaluate its performance (1 min)
  - Think of another approach and evaluate it (5 min)
    - Can you trade off space/memory for better runtime?
- Pick the most promising approach and start coding! (10 min)

Data structure Comparison

	Insert	Search	Min	Max	Delete min	Delete max	Delete (any)
Sorted array							
Unsorted array							
Sorted linked list (assume access to both head and tail)							
Unsorted linked list							
Stack							
Queue							
BST (unbalanced)							
BST (balanced)							
Min Heap							
Max Heap							

Data structure Comparison

	Insert	Search	Min	Max	Delete min	Delete max	Delete (any)
Sorted array	O(N)	O(logN)	O(1)	O(1)	O(N) if ascending order, else O(1)	O(1) if ascending, else O(N)	O(logN) 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(logN)	O(logN)	O(logN)	O(logN)	O(logN)	O(logN)	O(logN)
Min Heap	O(logN)	Not supported	O(1)	Not supported	O(logN)	Not supported	O(logN)
Max Heap	O(logN)	Not supported	Not supported	O(1)	Not supported	O(logN)	O(logN)