QUEUES
DATA STRUCTURE SELECTION

The Queue Operations

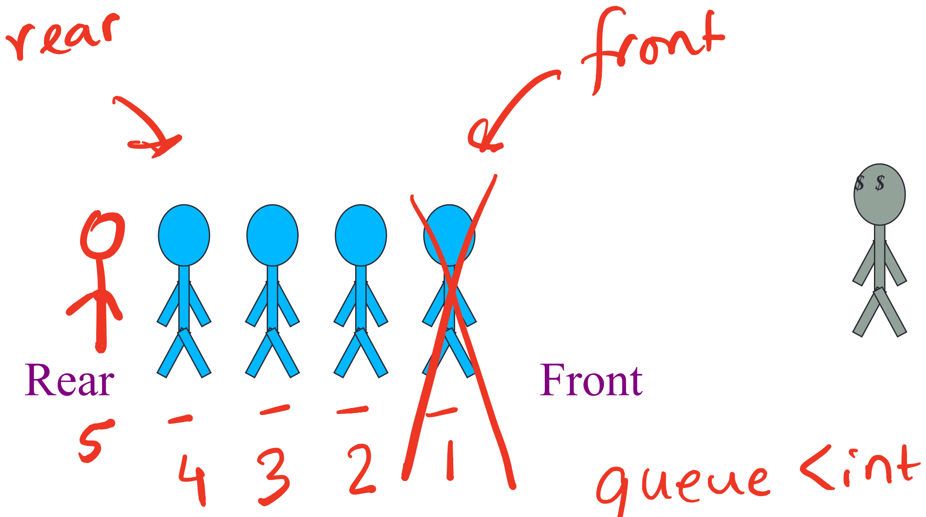
First key "in"
First key "out"
(FIFO)

- A queue is like a line of people waiting for a bank teller.
- The queue has a **front** and a **rear**.

push(5)

pop()

front() // 2



queue<int> q;

The Queue Class

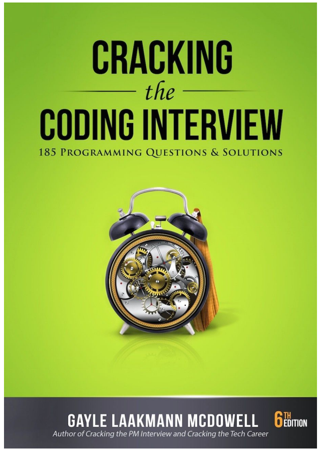
- The C++ standard template library has a queue template class.
- The template parameter is the type of the items that can be put in the queue.

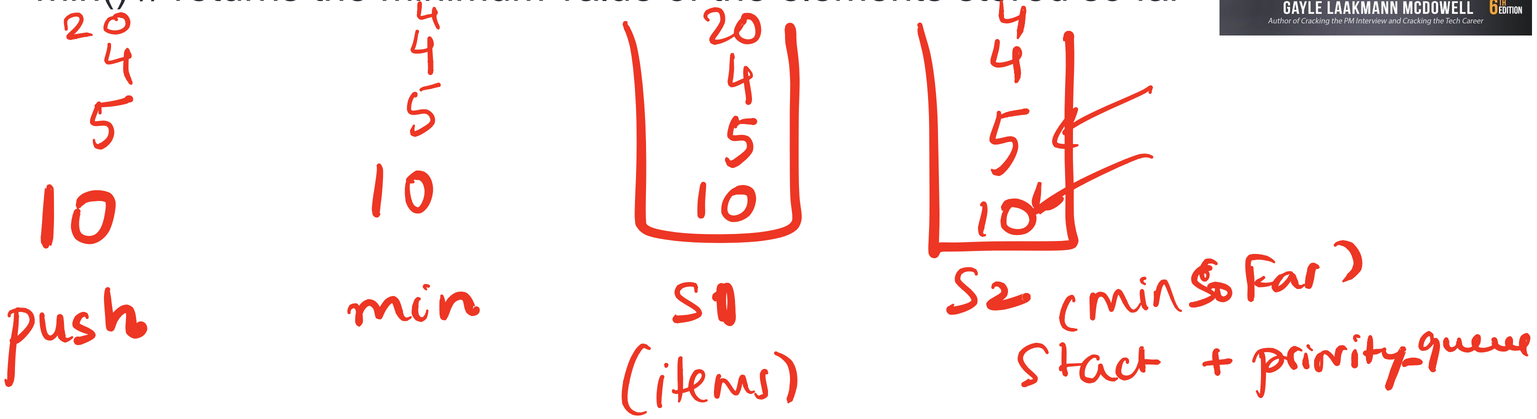
```
template <class Item>
class queue<Item>
{
public:
        queue( );
        void push(const Item& entry);
        void pop( );
        bool empty( ) const;
        Item front( ) const;
        …
```

Small group exercise

Write a ADT called in 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

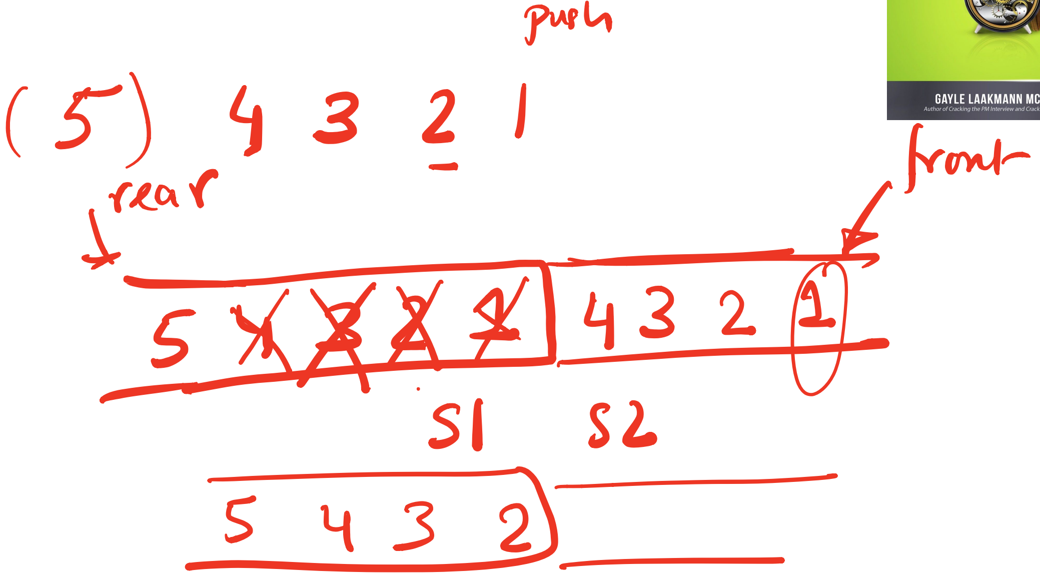

CRACKING the CODING INTERVIEW
185 PROGRAMMING QUESTIONS & SOLUTIONS
GAYLE LAAKMANN MCDOWELL
6TH EDITION
Author of Cracking the PM Interview and Cracking the Tech Career

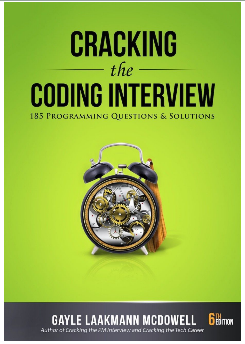


Queue via stacks

Implement a MyQueue class which implements a queue using two stacks

→ push()
→ pop()
→ front()
empty()




CRACKING the CODING INTERVIEW
185 Programming Questions & Solutions
GAYLE LAAKMANN MCDOWELL 6th Edition
Author of Cracking the PM Interview and Cracking the Tech Career

Selecting data structures

$O(N^2)$

```
void mystery(vector<int>& a, int N){
      //Precondition: unsorted vector of size N

      for(int i =0; i<N; i++){ // N times
            int minElem = a[i];
            int index=i;
            for(int j = i+1; j<N;j++){
                  if(a[j]<minElem){
                        minElem = a[j];
                        index = j;
                  }
            }
            int tmp = a[i];
            a[i] = a[index];
            a[index]=tmp;
      }

}
```

$\rightarrow$ N times

$\rightarrow c_1$

$\rightarrow c_3$

$(N-1)c_3 + (N-2)c_3 + \cdots$

$\rightarrow c_2$

$+ \cdots \quad c_3$

$c_3[1+2+3+\cdots\cdots(N-1)] + (c_1+c_2)N$

Practice functors and PQs:

$$C_3 \frac{N(N+1)}{2} + (C_1+C_2)N$$

$$C_3 \frac{N^2}{2} + C_3\frac{N}{2} + (C_1+C_2)N$$

```
int main(){
        int arr[]={10, 2, 80};
        priority_queue<int*> pq;
        for(int i=0; i < 3; i++)
                pq.push(arr+i);

        while(!pq.empty()){
                cout<<*pq.top()<<endl;
              pq.pop();
        }
        return 0;
}
```

What is the output of this code?

A. 10 2 80
B. 2 10 80
C. 80 10 2
[✓ D.] 80 2 10
E. None of the above

Sort array elements using a pq storing pointers

```
int main(){
      int arr[]={10, 2, 80};
      priority_queue<int*> pq;
      for(int i=0; i < 3; i++)
            pq.push(arr+i);

      while(!pq.empty()){
            cout<<*pq.top()<<endl;
           pq.pop();
      }
      return 0;
}
```

How can we change the way pq prioritizes pointers?

Write a custom comparison class

Write a comparison class to print the integers in the array in sorted order

```
int main(){
	int arr[]={10, 2, 80};
	priority_queue<int*, vector<int*>, cmpPtr> pq;
	for(int i=0; i < 3; i++)
		pq.push(arr+i);

	while(!pq.empty()){
		cout<<*pq.top()<<endl;
	    pq.pop();
	}
	return 0;
}
```

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)