Queues
PRIORITY QUEUES
DATA STRUCTURE SELECTION

Final exam

- About the final exam: https://ucsb-cs24-s19.github.io/exam/e03/
  - PSYCH 1924 (Wed- June 12) 8am-11am

- Review session: Sunday, June 9th, 5p-7p
  - Phelps 2510

The Queue Operations

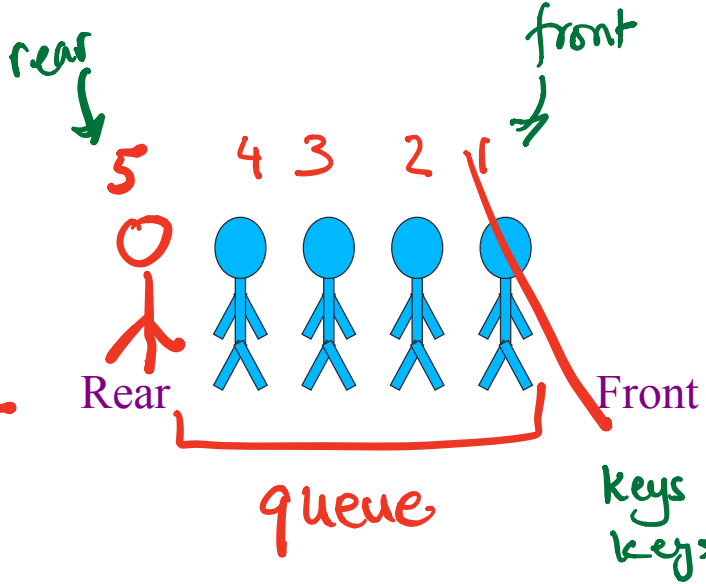
- A queue is like a line of people waiting for a bank teller.
- The queue has a **front** and a **rear**.

push(5)

pop() // 1

empty()

front() // 2



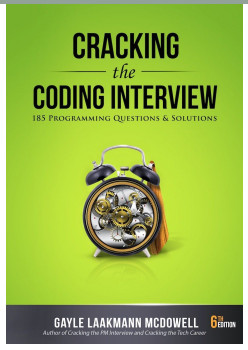
In a queue the first key in is the first key out (FIFO)

keys are deleted from the front
keys are inserted to the rear

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;
        …
```



Queue via stacks

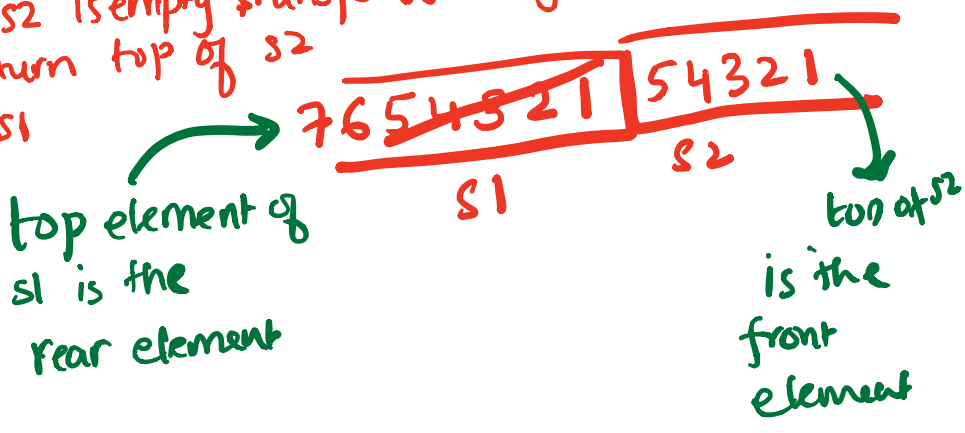
Implement a MyQueue class which implements a queue using two stacks

push → 1 2 3 4 5] → push to s1

front()/pop() → If s2 is empty transfer s1's keys to s2 return top of s2

push 6, 7 → push to s1

empty ()



Data structures

Implement all the ds.

- Linked list
- BST
- Dynamic arrays
- Stack
- Queue
- Heap

OOP

C++

- classes
  - member functions
  - private
  - public

Operator overloading

Big Four

- constructor
- destructor
- copy constructor
- copy-assignment

Running Time

- simple examples involving loops
- + data structures

Data structure Comparison

See last slide

	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							

Selecting data structures

```
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;
      }
}
```

O(1)

(N-i-1) constant time operations

O(1)

overall no. of primitive operations : $c(1+2+3+4+\cdots N-1) = \frac{N(N-1)}{2}$

$= O(N^2)$

Practice functors and PQs:

Address: 0x80 0x84 0x88

```
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

Memory addresses are stored by default we get a max heap so key at the highest address is popped first

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?

Define a compare class

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

```
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;
}
```

```
class cmpPtr {
  bool operator()(int* a, int* b) {
      return *a > *b;
  }
};
```

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)