WRAP-UP!
PRIORITY QUEUES
DATA STRUCTURE SELECTION

Slightly different from last time: functors and PQs:

```
int main(){
      int arr[]={20, 5, 92};
      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. 20 5 92
B. 92 5 20
C. 5 20 92
D. 92 20 5
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?

Review: Priority Queues and Comparison Classes

- Default max heap:
  - When the priority_queue needs to figure out if element b has higher priority than element a, it compares them using <
    - a < b
    - <(a, b)
    - less_than(a, b)
    - compare(a, b)
- When you make your own comparison class and give it to a priority_queue, it will use it in the same way: compare(a, b) to test if b has greater priority than a.
  - So if you want a min-heap, compare = greater_than = >

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;
}
```

Midterm 2

- O(H) vs. O(N) for unbalanced BSTs
- Best-base minimum for linked lists (where the list isn’t necessarily sorted)

pa02 running time measurement

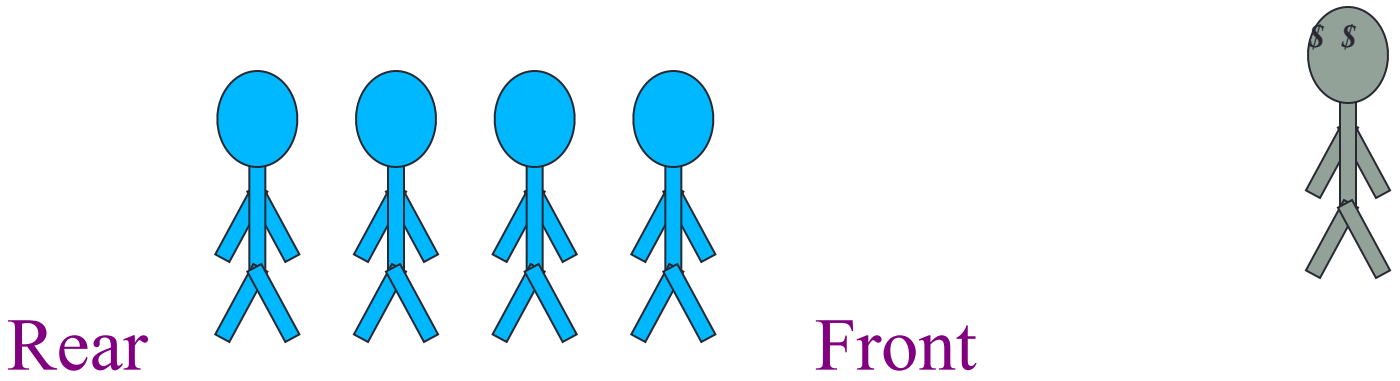
- <ctime> example
  - (see http://www.cplusplus.com/reference/ctime/clock/)

Final exam

- In almost two weeks!

The Queue Operations

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



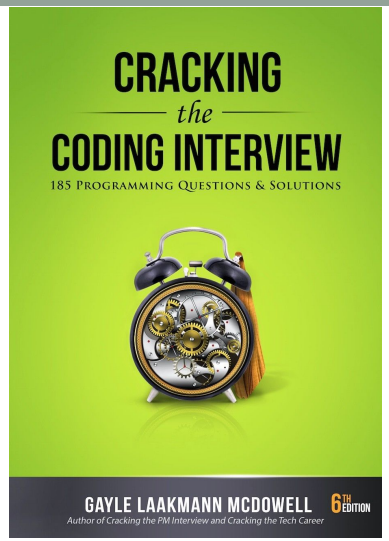
The Queue Class

- 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



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;
	}

}
```

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 (worst case)

	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) (to the end)	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)