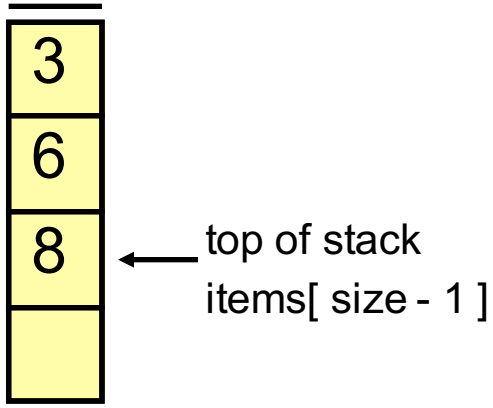
Announcements

MP3 available, due 10/02, 11:59p.

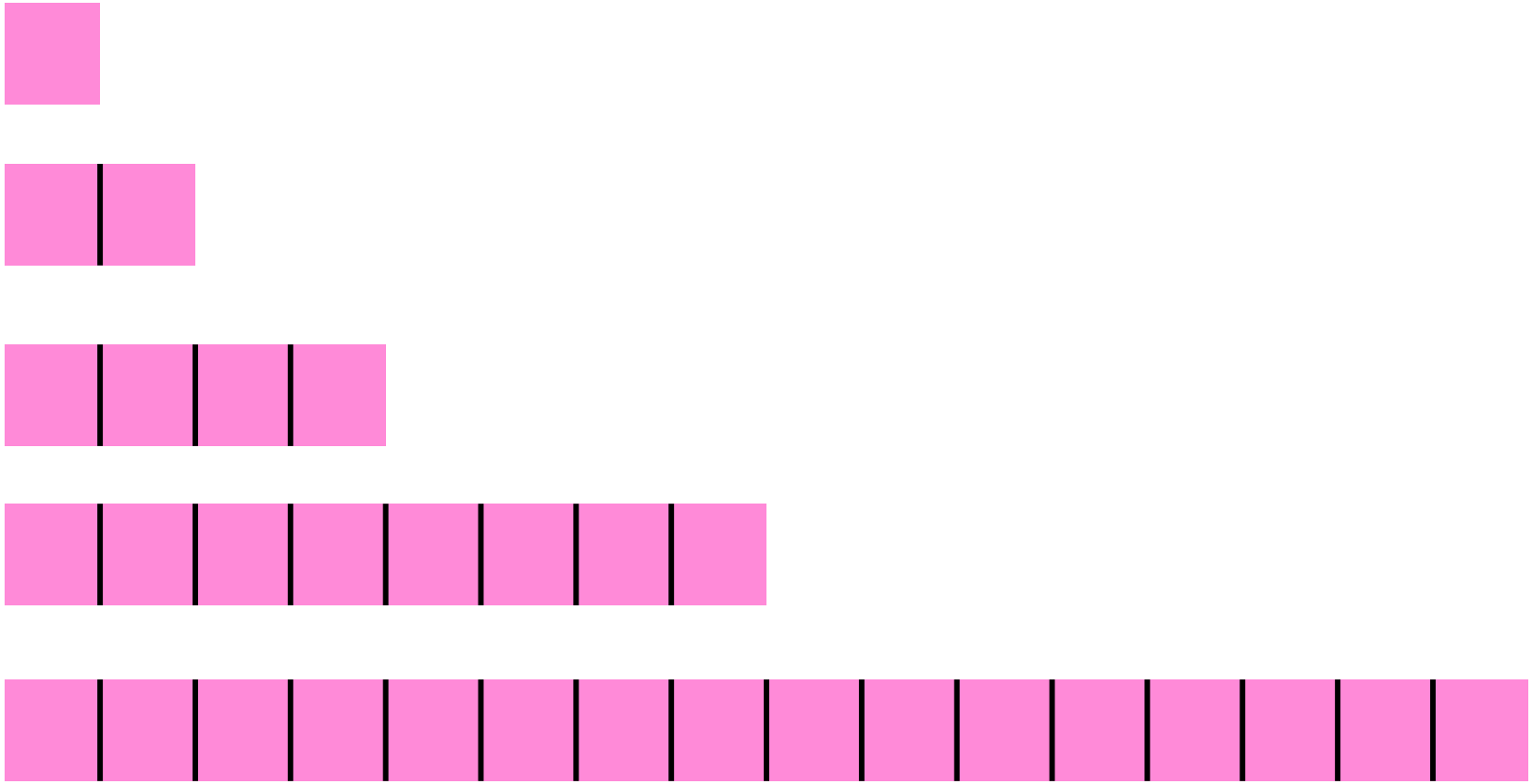
Stack -- array based implementation:

```
template<class SIT>
class Stack {
public:
    Stack();
    ~Stack(); // etc.
    bool empty() const;
    void push(const SIT & e);
    SIT pop();
private:
    int capacity;
    int size;
    SIT * items;
};
```

```
template<class SIT>
void Stack<SIT>::push(const SIT & e){
    if (size >= capacity) {
      // grow array somehow
    }
    items[size] = e;
    size ++;
}
```



Stack-- array based implementation: (what if array fills?)



How does this scheme do on a sequence of n pushes?

Summary:

Linked list based implementation of a stack:

Constant time push and pop.

Array based implementation of a stack:

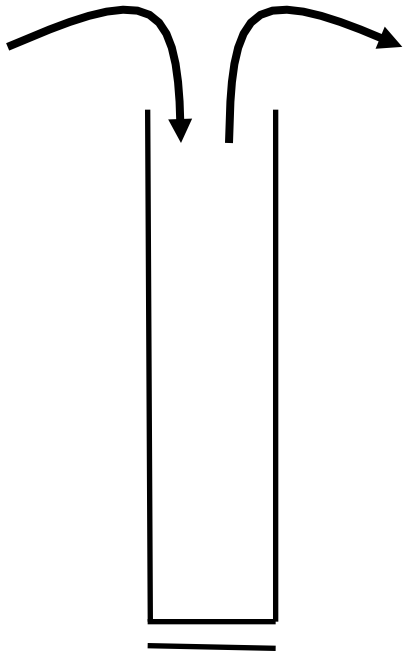
_____ time pop.

_____ time push if capacity exists,

Cost over O(n) pushes is _____ for an AVERAGE of _____ per push.

Why consider an array?

Queues:



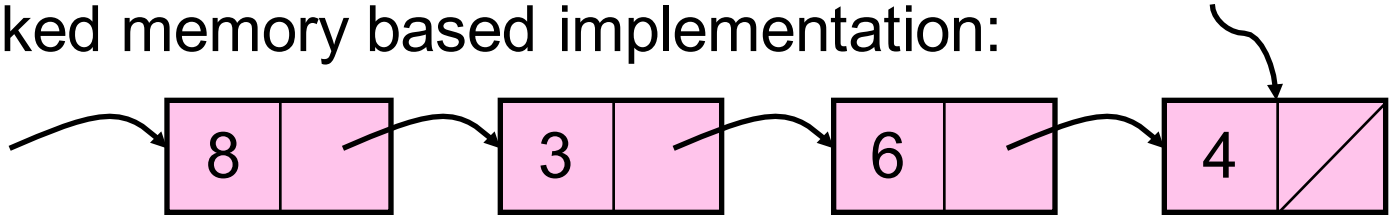
Queue ADT:

enqueue

dequeue

isEmpty

Queue—linked memory based implementation:




```
template<class  SIT>
class Queue {
public:
     // ctors dtor
     bool empty() const;
     void enqueue(const  SIT & e);
     SIT dequeue();
private:
     struct queueNode {
          SIT data;
          queueNode  * next;
     };
     queueNode  * entry;
     queueNode  * exit;
     int size;
};
```

Which pointer is “entry” and which is “exit”?

What is running time of enqueue?

What is running time of dequeue?

Queue array based implementation:

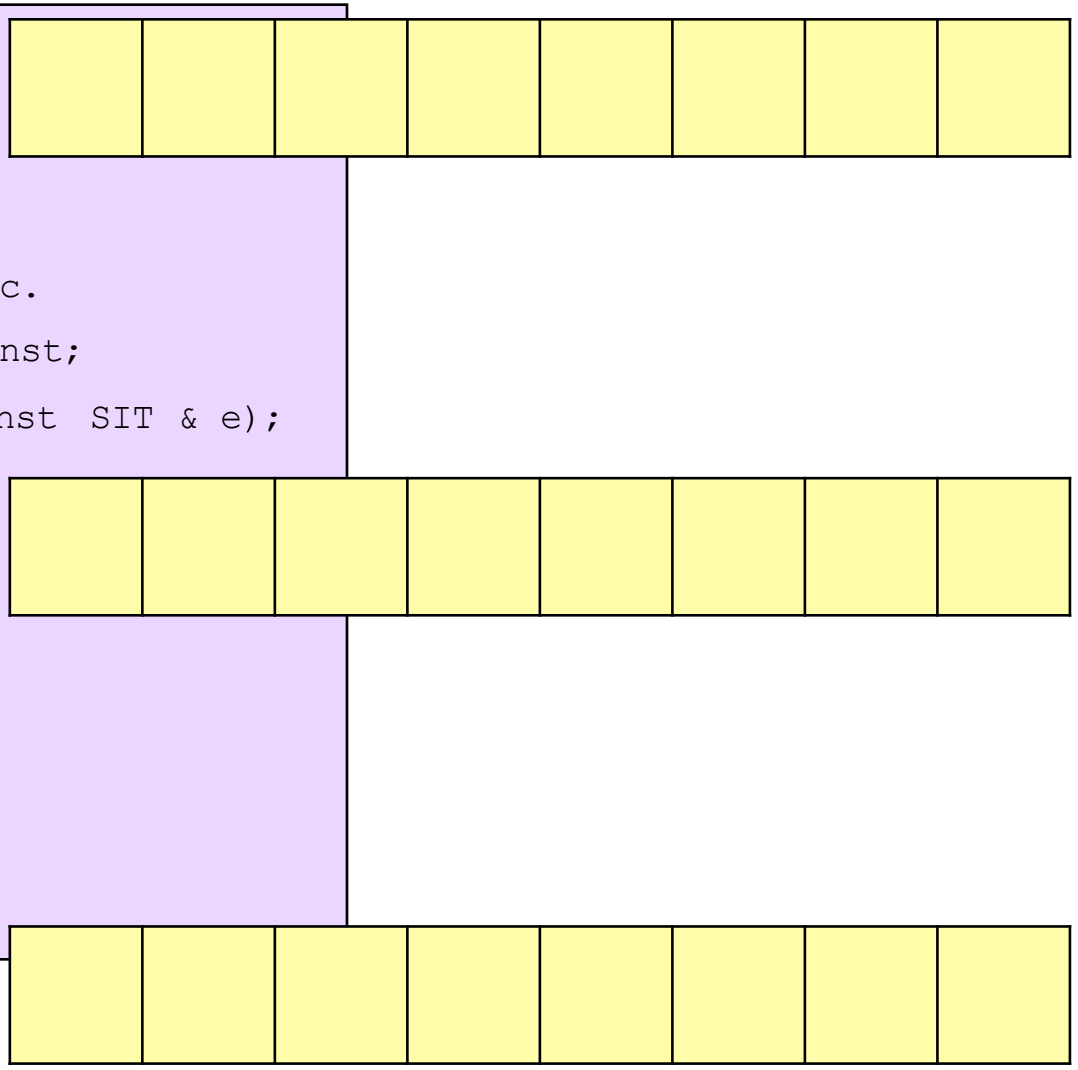
```
template<class SIT>
class Queue {
public:
    Queue();
    ~Queue(); // etc.
    bool empty() const;
    void enqueue(const SIT & e);
    SIT dequeue();
private:
    int capacity;
    int size;
    SIT * items;
    // maybe some other stuff…
};
```

```
template<class SIT>
Queue<SIT>::Queue(){
    capacity = 8;
    size = 0;
    items = new SIT[capacity];
}
```

Queue array based implementation:

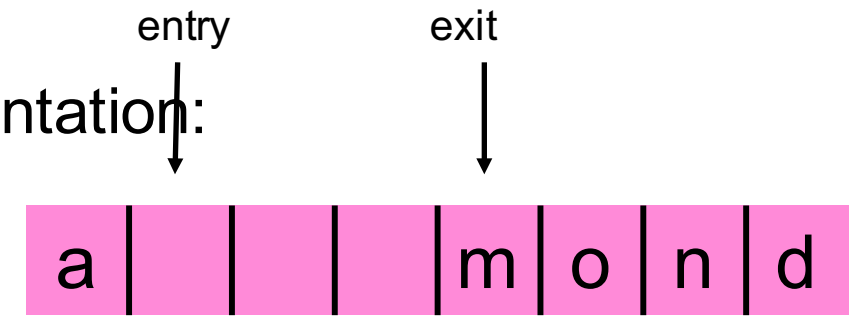
```
template<class SIT>
class Queue {
public:
    Queue();
    ~Queue(); // etc.
    bool empty() const;
    void enqueue(const SIT & e);
    SIT dequeue();
private:
    int capacity;
    int size;
    SIT * items;


};
```



enqueue(3);
enqueue(8);
enqueue(4);
dequeue();
enqueue(7);
dequeue();
dequeue();
enqueue(2);
enqueue(1);
enqueue(3);
enqueue(5);
dequeue();
enqueue(9);

Queue array based implementation:




```
template<class  SIT>
class  Queue  {
public:
     Queue();
     ~Queue();  // etc.
     bool  empty()  const;
     void  enqueue(const  SIT  &  e);
     SIT  dequeue();
private:
     int c
     int s
     SIT * items;
     int entry;
     int exit;
     // some other stuff…
};
```

enqueue(y);

enqueue(i);

enqueue(s);

dequeue();

enqueue(h);

enqueue(a);