Announcements
MP3 available, due 10/03, 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 ++;
}

3
6
8

→ top of stack
items[ size - 1 ]
Stack array based implementation: (what if array fills?)

Analysis holds for array based implementations of Lists, Stacks, Queues, Heaps…

General Idea: upon an insert (push), if the array is full, create a larger space and copy the data into it.

Main question: What’s the resizing scheme? We examine 3.
Stack array based implementation: (what if array fills?)

How does this scheme do on a sequence of n pushes?
Stack array based implementation: (what if array fills?)

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

```cpp
// 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:

```cpp
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();
enqueue(2);
enqueue(1);
enqueue(3);
enqueue(5);
dequeue();
enqueue(9);
Queue array based implementation:

```cpp
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;
    int entry;
    int exit;
    // some other stuff...
};
```
What if array fills?:

entry

exit

a y i s h o n d