ALGONQUIN COLLEGE

CST8130 – Data Structures

Professor : Dr. Anu Thomas
Email: thomasa@algonquincollege.com
Office: T314

Queues

Terminology - Queue

- Additions are done to “head” – called “*push*” onto queue, or also called “*enqueue*”
- Deletions are done at “tail” – called “*pop*” off queue, or also called “*dequeue*”
- Hence……FIFO (first in, first out), or LILO (last in, last out)

Why?

When you want to process “in the order” received... i.e. a waiting line

Using Arrays

```
Public class Queue extends Numbers {
        public boolean push (int element) {
            return add (element);
        }

        public int pop () {
            return deleteAtTop()
        }

        public boolean isEmpty() {
            return numItems==0;

        public boolean isFull() {
            return numItems == maxItems;
        }
}
```

Efficiency for Arrays approach

Algorithm

- push O(1) (constant)
- **pop O(n) (NOT GOOD)**
- isEmpty O(1) (constant)
- isFull O(1) (constant)

Memory

- One extra reference for each element
- Block of memory of maxSize elements

Can we improve this?

Is there a way to get this efficiency back??

Yes….keep a top and bottom index and wrap around the array ….

Using Arrays #2

```
public class Numbers2  {
    protected int maxItems = 10;  // default value
    protected int numItems = 0;    // no elements added yet
    protect int top=0, bottom = 0;
    protected int [ ] numbers = new int[maxItems];

      // selected methods here……..
    public boolean add (int newOne) {
        if (numItems == maxItems)
              return false;
          numbers[bottom++] = newOne;
        if (bottom >= maxItems)   // wrap around
              bottom = 0;
        numItems++;
          return true;
    }
    public int deleteAtEnd() {
        if (numItems == 0)
                return -1;
        if (bottom == 0)  bottom = maxItems;
        int temp = number[--bottom];
          numItems --;
        return temp;
     }
```

Continued…

```
public int deleteAtTop() {
        if (numItems == 0)
                return -1;
        int temp = numbers[top++];
        if (top == maxItems)   //wraparound
              top = 0;
        numItems--;
        return temp;
}
```

Efficiency for Dynamically allocated array strategy #2?

Algorithm

- push O(1) (constant)
- **pop O(1) (GOOD)**
- isEmpty O(1) (constant)
- isFull O(1) (constant)

Memory

- One extra reference for each element
- Block of memory of maxSize elements

Using LinkedList

```
public class Queue  extends LList {    // As I published in my earlier notes

        public boolean push (int element) {
              return addAtHead(element);
        }

        public int pop () {
               return deleteAtTail()
        }


        public boolean isEmpty() {
                return head == null;

        public boolean isFull() {
                 return false;
        }

}
```

Efficiency for Linked List implementation?

Algorithm

- push O(1) (constant)
- pop O(1) (constant) ** if keep tail pointer otherwise O(n)
- isEmpty O(1) (constant)
- isFull O(1) (constant)

Memory

- Two extra reference for each element (dll)
- One reference for head, another for tail

Questions?