enqueue(h):
  ↓
Now can't add even though there is space at front

SOLN: circular array - "wrap-around" array
end of array wraps around to beginning

Now: enqueue(h):

```
    0 1 2 3 4 5 6
   h ... c d e f g
   ↑     ↑     ↑
back  front  back
  0     3     2
```

Problem? front will go to the end (after some dequeues)

after 4 dequeues:

```
    h ... [ ] [ ] [ ] [ ] [ ] [ ]
   ↑     ↑
back  front
```

After next dequeue():

```
    h [ ] [ ] [ ] [ ] [ ] [ ]
   ↑
back
```
front should point here

How to implement? Insert checks before front++ & back++
In enqueue():

```
if (back == Q.length-1) back = 0
```
In dequeue():

```
if (front == Q.length-1) front = 0
```
**Linked List Implementation of Queues**

Maintain:
- Node front: Pointer to front, initially = null
- Node back: back
- int currentSize: number of elements = 0

```
    a -> b -> c -> d -> null
    ^  ^  ^  
front  back
```

1. **enqueue(e)**
   - Node n = new Node(e)
   - back.next = n
   - back = n
   - currentSize++

2. **dequeue()**
   - Node tmp = front
   - front = front.next
   - currentSize--
   - return front.element

3. **isEmpty()**: return currentSize == 0

4. **size()**: return currentSize.
Recursion (Review) - programming technique where solution to the problem depends on solutions to sub-problems.

2 Parts:
- Base Case: solved without recursion
- Recursive Step: makes progress toward base case

1. Summing values from 1 to n.

Problem: summing from 1 to n.
Sub-problem: "" to values ≤ n.

\[
\sum_{i=1}^{n} i = 1 + 2 + 3 + 4 + \cdots + n
\]

\[
\sum_{i=1}^{n-1} i = n + n-1 + n-2 + \cdots + 1
\]

Write in terms of smaller versions of the same problem (summing):

\[
\sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i \quad \text{(recursion)} \\
= n + \sum_{i=1}^{n-2} i \quad \text{(recursion)}
\]

We can stop expressing as a smaller problem when we get to a point where we can solve the problem directly. Which value of \( n \)? \( n = 1 \rightarrow \text{Base Case} \).
Base Case: For $n=1$, $\sum_{i=1}^{n} i = 1 + \bigcirc$ no recursion!

Write as a recursive function:

```c
int sum(n) {
    if (n == 1)
        return 1;
    else
        return (n + sum(n-1));
}
```

ex: $\text{sum}(5) = 15$
\[5 + \text{sum}(4) = 5 + 10 = 15\]
\[4 + \text{sum}(3) = 4 + 6 = 10\]
\[3 + \text{sum}(2) = 3 + 3 = 6\]
\[2 + \text{sum}(1) = 2 + 1 = 3\]
\[1\]

ex(2): Factorial: $n! = n \times (n-1) \times (n-2) \times \cdots \times 1$
How to write $n!$ in terms of smaller factorials?

\[n! = n \times (n-1)! = n \times (n-1) \times (n-2)! = n \times (n-1) \times (n-2) \times (n-3)!\]

Base Case? $n=1$ (or $n=0$). $\Rightarrow$ then $n! = 1$
factorial \( n \) if \( n = 1 \) return 1 else return \( n \times \text{factorial}(n-1) \)

\[
\begin{align*}
\text{ex: } \text{factorial}(4) &= 24 \\
4 \times \text{factorial}(3) &= 4 \times 6 = 24 \\
3 \times \text{factorial}(2) &= 3 \times 2 = 6 \\
2 \times \text{factorial}(1) &= 2 \times 1 = 2 \\
\text{[ ]}
\end{align*}
\]

3. Sorted list A, of \( n \) numbers. Return true if \( x \) is in list; o/w return false. If \( y > x \), check middle

Time to scan? \( O(n) \)

Better: Binary Search:

\[
\text{BinSearch}(A) = \text{BS}(A[0 \ldots n/2]) \text{ or } \text{BS}(A[n/2+1 \ldots n-1])
\]

Base Case? when just one element in list.

Time: \( n \): Start with list of size \( n \). In the worst case, keep checking until list size is 1

\[
\{ 0(\log n) \}
\]
BinSearch(A, start, end, x)

// if just one element in list
if (start == end)
    if (A[start] == x)
        return true
    else
        return false

→ mid = \frac{start + end}{2}

No else
rec. here

if (A[mid] == x)
    return true
else if (A[mid] < x) //search right
    return BinSearch(A, mid+1, end, x)
else //search left
    return BinSearch(A, start, mid-1, x).
BS(A, 0, 6, 19)
\[ \text{mid} = 3 \quad A[3] = 12 < 19 \Rightarrow \text{right} \]

BS(A, 4, 6, 19) \text{ mid} = 5 \quad A[5] = 20 > 19 \Rightarrow \text{left} \]

BS(A, 4, 4, 19) \text{ start} = \text{end} = 4
\[ A[4] = 19 \quad \text{return true} \]

ex: \( x = 18 \) ?

BS(A, 0, 6, 18)
\[ \text{mid} = 3 \quad A[3] = 12 \leq 18 \Rightarrow \text{right} \]

BS(A, 4, 6, 18) \text{ mid} = 5 \quad A[5] = 20 > 19 \Rightarrow \text{left} \]

BS(A, 4, 4, 18) \text{ start} = \text{end} = 4
\[ A[4] \neq 18 \quad \text{return false} \]
Recall: operation contains(x) takes $O(n)$ for ArrayLists and LinkedLists, arrays since there is no ordering among the elements.

Binary Search Trees (BST) - data structure that has more ordering among elements.

Definitions:
- tree: an ordered collection of nodes
- root: the topmost node
- edge: the connection between nodes
- subtree: a tree contained in a larger tree
- node: a point in the tree
- leaf: a node with no children
- child: b is child of a if b is directly below a
- parent: a "parent" of b is directly above b
- siblings: nodes with same parent
- grandparents: parents of parent
- grandchildren: children of child
path from node $n_i$ to $n_k$ - sequence of nodes $n_1, \ldots, n_k$ such that $n_i$ is parent of $n_k$.

Example: path from $a$ to $k$? $a \rightarrow c \rightarrow h \rightarrow k$

path length - # of edges on the path

Example: $a \rightarrow c \rightarrow h \rightarrow k \Rightarrow 3$

Notice: for every node, there is a unique path from root to that node.

depth of a node $n_i$ - length of path from root to $n_i$.

Example: depth of $b$? = 1, depth of $k$? = 3

height of a node $n_i$ - length of longest path from $n_i$ to a leaf.

Example: height of $b$? = 2, $a$? = 3
Application for Trees

```
My Comp
  |- Project
    |- Docs
      |- CS201
        |- HW1
        |- Lab1
      |- CS202
        |- HW1
        |- Lab1
    |- Pics
      |- CS201
      |- Baby
      |- Pic1
      |- Pic2
    |- Backup
    |- BDay
  |- C
  |- D
```
void listAll(int d)

    print (d spaces)
    print name of node at depth d
    print ("\n")

    if (node at depth d is directory:)
        for each node of this directory:
            listAll(d+1)

Always print parent node first, then children.

Pre-order traversal -
Start at root; for each node:
1. Process node
2. Recursively process children

Run-Time in terms of \( n \) = # nodes? \( \Rightarrow O(n) \)

Another example

pre-order:
\[ a \ b \ e \ f \ j \ c \ g \ h \ k \ i \ d \]

post-order:
\[ e \ f \ j \ b \ g \ h \ i \ c \ d \ a \]