Introduction to Computer Science
Reichman University

Recitation 11

Overview

- Queue - Generic
- Sorting
- Linked List
  - 2 data LinkedList
  - List Iterator
  - Advanced Linked List Operations

Question 0 - Linked List - Equals

- Given 2 String linked lists, build a function which return true if the elements of the lists are appearing in the same order.

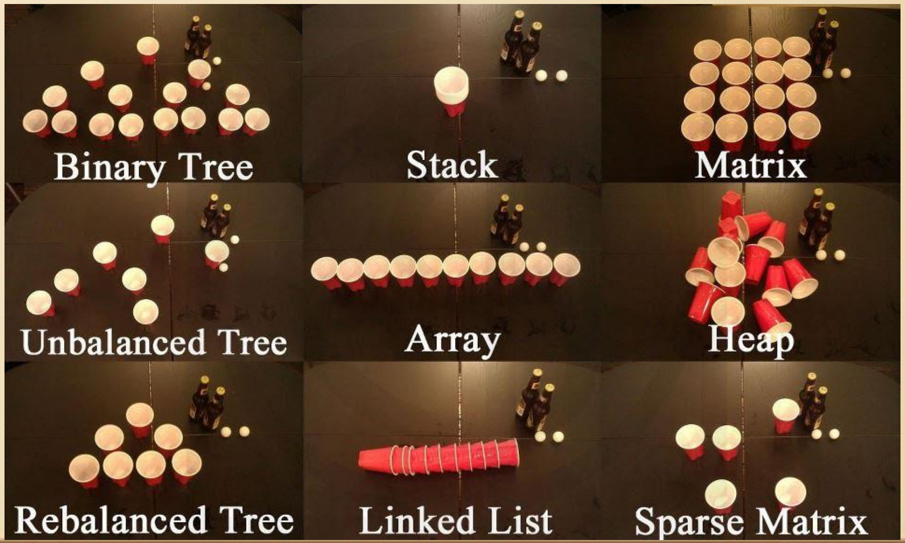
Question 0 - Solution

```java
public boolean equals(LinkedList other) {
       if (this.size != other.getSize()) {
           return false;
       }
       Node myCur = this.first;
       Node otherCur = other.first;
       while (myCur != null && otherCur != null) {
           if (myCur.data != otherCur.data) {
               return false;
           }
           myCur = myCur.next;
           otherCur = otherCur.next;
       }
       return true;
}
```

Introduction to Computer Science
Reichman University

Recitation 11

Queue - Generic



Queue

- Queue is a data structure which operates under the FIFO (first in first out) concept.
- Queue has several methods
- Enqueue – add to the end of the line
- Dequeue – removes from the start of the line

Node Generic

```
public class Node<E> {
      E data;
      Node<E> next;

      public Node(E data) {
            this.data = data;
            this.next = null;
      }

      public Node(E data, Node<E> next) {
            this.data = data;
            this.next = next;
      }

      public String toString() {
            return this.data.toString();
      }
}
```

Queue Generic

```
public class Queue<E> {
      private Node<E> head;
      private Node<E> tail;
      private int size;

      public Queue() {
            this.head = null;
            this.tail = null;
            this.size = 0;
      }
}
```

Queue Generic (Continued)

```
public class Queue<E> {
      public void enqueue(E data) {
            Node<E> newNode = new Node<E>(data);
            if (this.size == 0) {
                  this.head = newNode;
            } else {
                  this.tail.next = newNode;
            }
            this.tail = newNode;
            this.size++;
      }

      public E dequeue() {
            if (this.size == 0) {
                  return null;
            }
            E data = this.head.data;
            this.head = this.head.next;
            this.size--;
            if (this.size == 0) {
                  this.tail = null;
            }
            return data;
      }
}
```

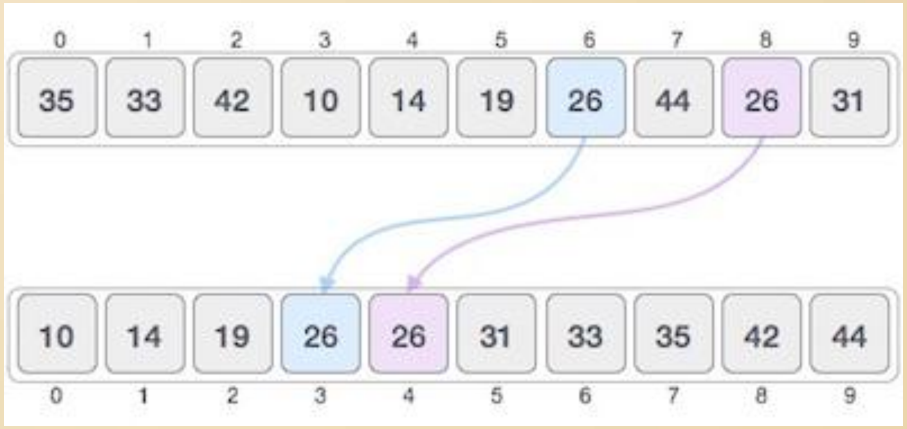
Queue Generic (Continued)

```java
public class Queue<E> {
      public E peek() {
            if (this.size == 0) {
                  return null;
            }
            return this.head.data;
      }
      public int getSize() {
            return this.size;
      }
      public boolean isEmpty() {
            return this.size == 0;
      }
      public String toString() {
            if (this.size == 0) {
                  return "[]";
            }
            StringBuilder sb = new StringBuilder();
            sb.append("[");
            Node<E> current = this.head;
            while (current.next != null) {
                  sb.append(current.data + ", ");
                  current = current.next;
            }
            sb.append(current.data + "]");
            return sb.toString();
      }
}
```

Introduction to Computer Science
Reichman University

Recitation 11

Sorting



Sorting

- Sorting is an important tool, and has many applications.
- When an array is sorted some of the basic operations take much less time:
  - finding the biggest and smallest element takes just one command.
  - using binary search for searching elements.
- There are many different sorting algorithms that defer in their running time.
- Given an array, if we have no prior information concerning the data, the best running time for sorting is $n \cdot \log n$ where n is the number of elements.
- One such Algorithm is Merge-Sort.

Question 1 – Bubble Sort

- Design a program which implements the bubble sort algorithm

- The algorithm works by comparing each item in the list with the item next to it, and swapping them if required.
- In other words, the largest element has bubbled to the top of the array.
- The algorithm repeats this process until it makes a pass all the way through the list without swapping any items.

Question 1 - Solution

```
public static void bubbleSort(int [] arr) {
      int n = arr.length;
      for (int i = 0; i < n - 1; i++){
           for (int j = 0; j < n - i - 1; j++){
            if (arr[j] > arr[j + 1]){
                swap(arr, j, j + 1);
                }
           }
      }
}
```

- What is the complexity?

Question 1 - Solution

```
public static void bubbleSort(int [] arr) {
      int n = arr.length;
      for (int i = 0; i < n – 1; i++){
           for (int j = 0; j < n – i – 1; j++){
            if (arr[j] > arr[j + 1]){
                swap(arr, j, j + 1);
                }
           }
      }
}
```

Question 1, Expansion 1 – Selection Sort

- Design a program implements the selection sort algorithm

- The algorithm works by selecting the smallest unsorted item and then swapping it with the item in the next position to be filled.
- The selection sort works as follows:

  - you look through the entire array for the smallest element.
  - once you find it you swap it (the smallest element) with the first element of the array.

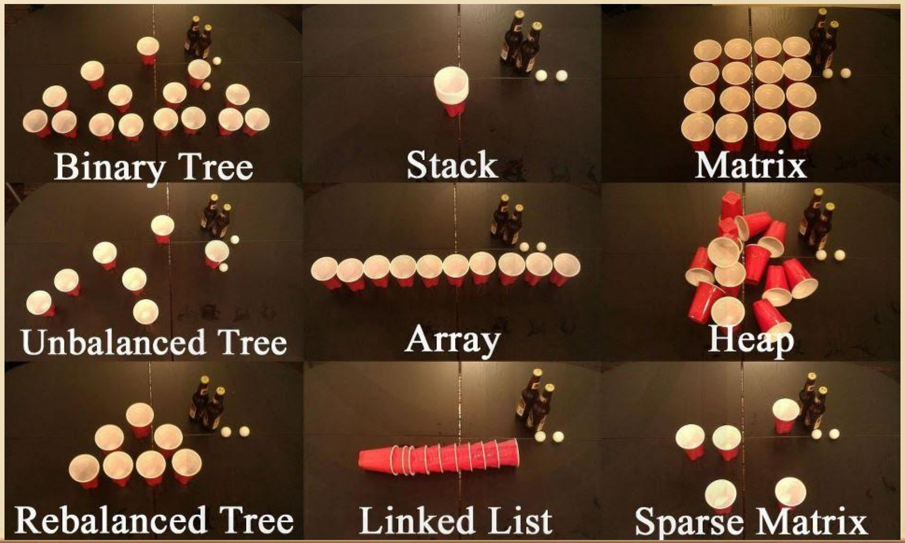
Question 1, Expansion 1 - Solution

```java
public static void selectionSort(int [] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            swap(arr, i, minIndex);
        }
}
```

Introduction to Computer Science
Reichman University

Recitation 11

Multiple “Data” fields in Node



Multiple data in Node

- Since Node is an object it may have multiple fields which consist relevant data.
- Real life example:
  - When you go online grocery shopping you have in your cart multiple items but if a certain element does appear more than once it written in the same line. There are no duplicates, but I know there is multiple items in the same ListItem.

Multiple data in Node

```
public class NodeTwoData {
   int counter;
   String name;
   NodeTwoData next;

  /** Constructs a node with the given data. The new node will point to
   *  the given node (next). */
 public NodeTwoData(String name, int counter, NodeTwoData next) {
       this.name = name;
       this.counter = counter;
       this.next = next;
  }

  /** Constructs a node with the given data. The new node will point to null , every new item
     will have counter (quantity) 1*/
 public NodeTwoData(String name) {
       this(name, 1, null);
   }
}
```

Question 2 - Constructor

- Design a program in Java where you write a constructor method for a LinkedListTwoData class. The program doesn't receive any other elements.

```
public class LinkedListTwoData {
       private NodeTwoData first;
       private int size;
 }
```

Question 2 - Solution

```
public LinkedListTwoData () {
       this.first = null;
       this.size = 0;
   }
```

Question 2, Expansion 1 – Add First

- Design a program that implements the 'addFirst(String item)' method. This method inserts an element at the beginning of a LinkedListTwoData.

Question 2, Expansion 1 - Solution

```
public void addFirst(String item) {
       NodeTwoData newNode = new NodeTwoData(item);
       newNode.next = this.first;
       this.first = newNode;
       this.size++;
   }
```

Question 2, Expansion 2 – Contains

- The function *contains(String name)* returns true if this list contains an element with that given name in the list.

Question 2, Expansion 2 - Solution

```
public boolean contains(String name){
       NodeTwoData current = this.first;
       while (current != null){
           if (current.name.equals(name)){
               return true;
           }
           current = current.next;
       }
       return false;
}
```

Question 2, Expansion 3 – Index Of

- The function indexOf(String name) . returns the index of the first appearance the given element appears in. The first appearance is element with that given name in the list.
- If the element doesn’t appear in the list, returns -1.

Question 2, Expansion 3 - Solution

```
public int indexOf (String name){
       if (!this.contains(name)) {
           return -1;
       }
       return this.indexOf(name, 0, this.first);
}
private int indexOf (String name, int index, NodeTwoData current){
       if (current.name.equals(name)){
           return index;
       }
       return this.indexOf(name, index + 1, current.next);
}
```

Question 2, Expansion 4 – Update

- The function update(int index) updates the counter by 1. if index is not legal the function throws an exception.

Question 2, Expansion 4 - Solution

```
public void update(int index){
       if (index >= this.size || index < 0){
           throw new IndexOutOfBoundsException("illegal index " + index);
       }
       NodeTwoData current = this.first;
       for (int i = 0; i < index; i++) {
           current = current.next;
       }
       current.counter++; // similar to array in function, so will change in the actual list
}
```

Question 2, Expansion 5 – Update

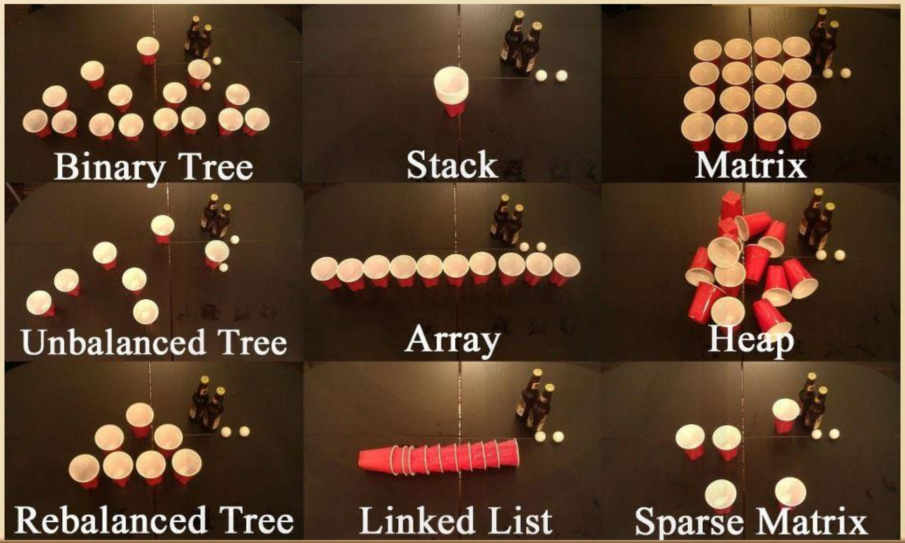
- The function update(String item) updates the counter by 1. if index is not legal the function throws an exception.

Question 2, Expansion 5 - Solution

```
public void addNoDups(String item){
      int index = this.indexOf(item);
      if (index == -1) {
      this.addFirst(item);
      return;
      }
      this.update(index):
}
```

Recitation 11

List Iterator



List Iterator

- One of the most basic operations we had with arrays, was to iterate over them.
- We would still like to have this functionality when using lists.
- An iterator class gives us exactly that.
- The Iterator class has a single field.
  - current – a pointer to the current node in the iteration.
- Iterators are created to fit a specific list, and always start with their current node, as the head of the list.

List Iterator

```
/** Represents an iterator of a linked list. */
public class ListIterator {

    // current position in the list (cursor)
    Node current;

    /** Constructs a list iterator,
     *  starting at the given node */
    public ListIterator(Node cur) {
        this.current = cur;
    }

    /** Checks if this iterator has more
     *  nodes to process */
    public boolean hasNext() {
        return (this.current != null);
    }

    /** Returns the current element in the list
     * and advances the cursor */
    public int next() {
        Node toReturn = this.current;
        this.current = this.current.next;
        return toReturn.data;
    }
}
```

Inside linked list

```
public ListIterator listIterator() {
      return new ListIterator(this.first);
}
```

Question 3 – Sum All Length of List

- Design a program that calculates the sum of all lengths of the String elements in a given LinkedList. The method receives LinkedList.
- We need to use List Iterator.

Question 3 – Solution

```
public int sumLenList(LinkedList l) {
    ListIterator it = l.listIterator();
    int ans = 0;
    while (it.hasNext()) {
        String current = it.next();
        ans += current.length();
    }
    return ans;
}
```

Question 4 - LastIndexOf

- The function last*IndexOf(String e)* returns the index of the last appearance the given element appears in.
- If the element doesn't appear in the list, returns -1.

Question 4 – Solution

```
public int lastIndexOf(String e) {
   int lastIndex = -1;
   int index = 0;
   ListIterator it = this.listIterator();
   while (it.hasNext()) {
   if (it.next().equals(e)) {
           lastIndex = index;
       }
   index++;
   }
   return lastIndex;
}
```

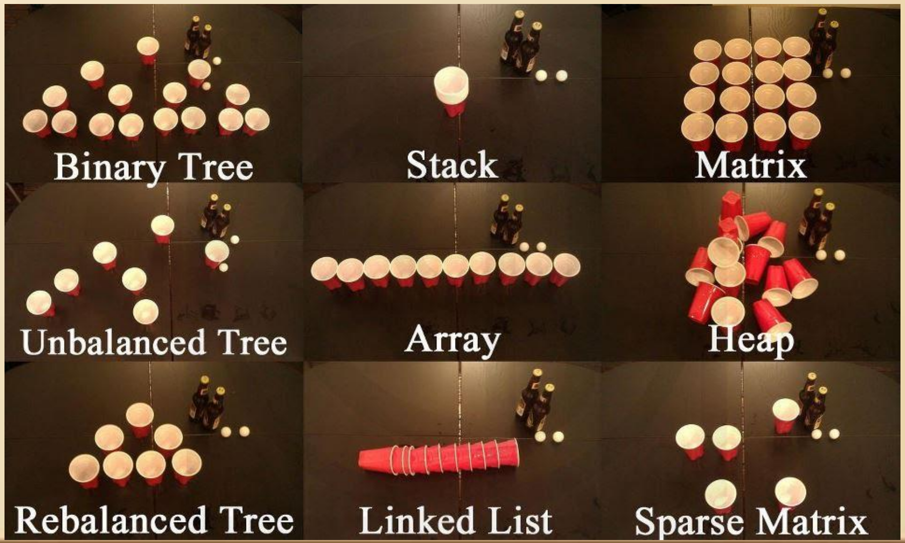
Question 4 – Solution (Better)

```
public int lastIndexOf(String e) {
   int lastIndex = -1;
   int index = 0;
   ListIterator it = this.listIterator();
   while (it.hasNext()) {
   String current = it.next();
   if (current.equals(e)) {
           lastIndex = index;
       }
   index++;
   }
   return lastIndex;
}
```

Introduction to Computer Science
Reichman University

Recitation 11

Linked Lists – List Operations



Question 5 - Linked List - Add in Lexicographic Order

- In mathematics, the lexicographic or lexicographical order (also known as lexical order, or dictionary order) is a generalization of the alphabetical order of the dictionaries to sequences of ordered symbols or, more generally, of elements of a totally ordered set.
- Given a Lex' ordered String linked list, and a String str. Add str to the linkedlist that after you add the string will still have a Lex' ordered linked list
- Use String.compareTo(), does lex' comp'

Question 5 – Solution

```
public void addLexicographicOrder(String data) {
       if (this.first == null || this.first.data.compareTo(data) >= 0) {
           this.addFirst(data);
           return;
       }
       Node newElem = new Node(data);
       Node prev = null;
       Node current = this.first;
       while (current != null && current.data.compareTo(data) < 0) {
           prev = current;
           current = current.next;
       }
       newElem.next = current;
       prev.next = newElem;
       this.size++;
   }
```

Question 6 - Linked List - Equals

- Given 2 String linked lists, build a function which return true if the elements of the lists are appearing in the same order.

Question 6 - Solution

```
public boolean equals(LinkedList other) {
       if (this.size != other.getSize()) {
           return false;
       }
       Node myCur = this.first;
       Node otherCur = other.first;
       while (myCur != null) {
           if (!myCur.data.equals(otherCur.data)) {
               return false;
           }
           myCur = myCur.next;
           otherCur = otherCur.next;
       }
       return true;
   }
```

Question 7 - Linked List - Sorted list

- Given an unsorted String linked list, build a function which returns a sorted copy of the given linked list.

Question 7 – Solution

```
public LinkedList sortList() {
       if (this.isEmpty()) { // true if this.size == 0;
           return new LinkedList();
       }
       LinkedList sorted = new LinkedList();
       Node cur = this.first;
       while (cur != null) {
           sorted.addLexicographicOrder(cur.data);
           cur = cur.next;
       }
       return sorted;
   }
```

Question 8 - Linked List - Remove Duplicates

- Given a String linked list, build a function which returns a copy of the linked list but every element in the new list appears only once.

Question 8 - Solution

```java
public LinkedList removeDuplicates() {
       if (this.size == 0) {
           return new LinkedList();
       }
       LinkedList result = new LinkedList();
       Node current = this.first;
       while (current != null) {
           if (!result.contains(current.data)) {
               result.addFirst(current.data);
           }
           current = current.next;
       }
       return result;
}
```

Question 9 - Linked List - isPalindrome

- Given a String linked list, the function returns true if it is a palindrome linked list
- <u>Challenge</u>: Do it in one line!

Question 9 - Solution

```
public boolean palindrome() {
       return this.equals(this.reverse());
}
```

Question 9, Expansion 1 - Linked List - Equals unsorted

- Given 2 String linked lists, the function returns true if it is both lists contains the same elements and each element appears in both Lists the same number of times.
- <u>Challenge</u>: Do it in one line!

Question 9, Expansion 1 - Linked List - Equals unsorted

```
public boolean equalsUnsorted(LinkedList other) {
       return this.sortList().equals(other.sortList());
}
```

Question 9, Expansion 2 - Linked List - Equals unsorted 2

- Given 2 String linked lists, the function returns true if it is both lists contains the same elements and each element appears in both Lists not matter how many times, But the set of elements are the same
- Challenge : Do it in one line!

Question 9, Expansion 2 - Solution

```
public boolean equalsUnsorted(LinkedList other) {
       return this.removeDuplicates().sortList().equals(other.removeDuplicates().sortList());
}
```

Question 9, Expansion 3 - Drop 3

- Given a linked list, the function returns a new list which contains the same elements but drops every third element
- The function is recursive!

Question 9, Expansion 3 - Solution

```java
public LinkedList dropThree() {
       return dropThree(this.first, 0, new LinkedList()).reverse();
}

private LinkedList dropThree(Node cur, int counter, LinkedList result) {
       if (cur == null) {
           return result;
       }
       if (counter != 2) {
           result.addFirst(cur.data);
       }
       return dropThree(cur.next, (counter + 1) % 3, result);

}
```