CSCI-343 Midterm Review

I. Iterators

Implement an iterator over a custom linkedlist class you made that **skips every other element**.

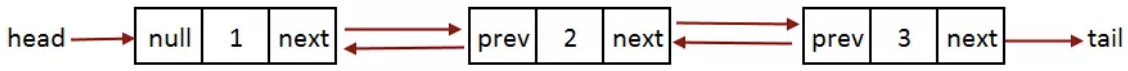
```
public Iterator<T> iterator() {
    return new Iterator<T>() {
      Node<T> currNode = head;
      Node<T> previous = null;
      public boolean hasNext() {
            return (currNode != null && currNode.next != null && currNode.next != null);
      }

      public T next() {
            T data;
            if(currNode != null)  data = currNode.data;
            if (hasNext()) currNode = currNode.next;
            if (hasNext()) currNode = currNode.next;
            return data;

      }
    };
  }
```

II. Doubly Linked List *(represented just by the root node here.*

2.1 Draw a simple representation of a doubly linked list.



Doubly Linked List

2.2 Now write code to determine if that linked list is circular

```
public boolean isCircular(Node<T> list) {
        if (list == null) return true;
        Node head = list;
        while (list != null && list != head)
                list = list.next;
        return (list == head);
}
```

2.4 Draw a simple representation of a doubly linked list, then the result of deleting the tail

2.5 Write code to delete tail of doubly (circular) linked list. realign it accordingly, then return the head

```
public Node<T> deleteTail(Node<T>  list) {
        if (list == null || list.next == null  || list.prev == null) return null;
        Node head = list;
        Node beforeTail = head.prev.prev;
        beforeTail.next = head;
        head.prev = beforeTail;
        return head;
}
```

III. **Arrays**

3.1 Matrices: write a function that transposes a matrix in place, in the shortest time possible, then circle the numbers below that **are actually visited**.

[{1, **2**, **3**, **4}**,

{5, 6, **7**, **8**},

{9, 10, 11, **12**},

{13, 14, 15, 16}]

[1, 5, 9, 13]

[2, 6, 10, 14]

[3, 7, 11, 15]

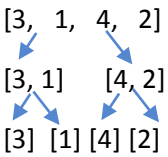
[4, 8, 12, 16]

```
int n = mat.length;
for (int i = 0; i < n; i++)
   for (int j = i+1; j < n; j++) {
      int temp = mat[i][j];
      mat[i][j] = mat[j][i];
      mat[j][i] = temp;
   }
```

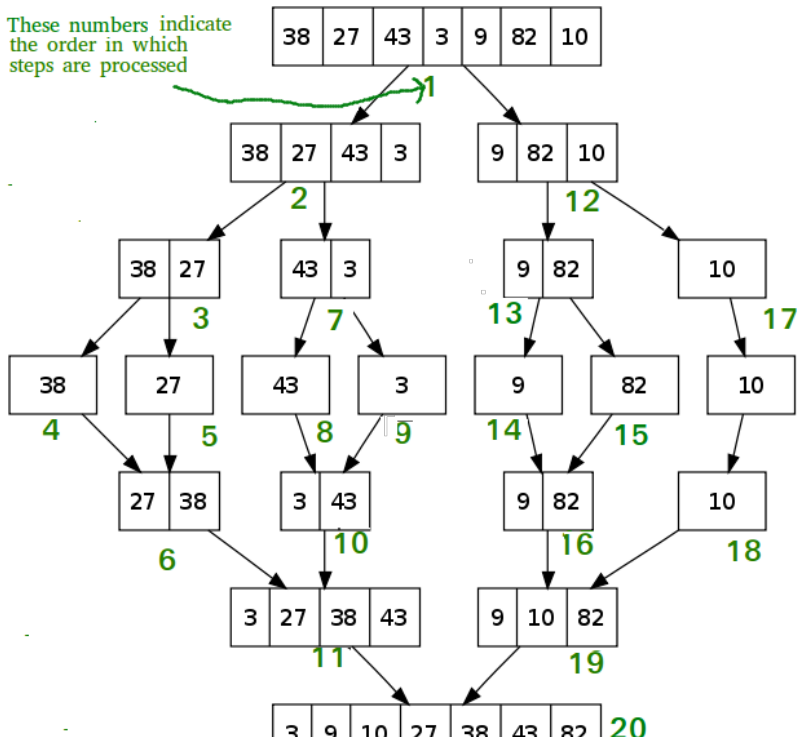
3.2 Merge-sort is an algorithm that recursively halves, sorts, and merges an array.

For example, with an array [3, 1, 4, 2], the first split would yield [3, 1], [4, 2], and subsequently [3], [1], [4], [2]. The next step would be to put it put it back together in sorted order.

Pictorially, this looks like:



Draw the full recursive tree (this means it has a root at the top and bottom) of this algorithm of the array [38, 27, 43, 3, 9, 82, 10]



3.2 using your intuition, with the number of comparisons made, what do you anticipate the run time (big-o complexity) of this algorithm to be and <u>why?</u> (hint: what happens to n every time?)

O (n log n) → See how it halves every time?