Mid-Term - ECE 242 – Fall 2017
Closed book/notes- no calculator- no phone- no computer

NAME:
ID:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- General questions (30pts)</td>
<td></td>
</tr>
<tr>
<td>2- Sorting (26pts)</td>
<td></td>
</tr>
<tr>
<td>3- Queue and Stack (10pts)</td>
<td></td>
</tr>
<tr>
<td>4- Recursion (12pts)</td>
<td></td>
</tr>
<tr>
<td>5- Linked-List (22pts)</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL (100pts)</strong></td>
<td></td>
</tr>
</tbody>
</table>
1- General questions (30 pts) (brief answers)

a- Looking at the following execution times in seconds, can you guess the Big-O of each algorithm?

<table>
<thead>
<tr>
<th>Number of Items</th>
<th>N=10</th>
<th>N=50</th>
<th>N=100</th>
<th>Big-O?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algo A</td>
<td>1s</td>
<td>25s</td>
<td>100s</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>Algo B</td>
<td>10s</td>
<td>50s</td>
<td>100s</td>
<td>$O(N^1)$</td>
</tr>
<tr>
<td>Algo C</td>
<td>2s</td>
<td>250s</td>
<td>2000s</td>
<td>$O(N^3)$</td>
</tr>
<tr>
<td>Algo D</td>
<td>50s</td>
<td>85s</td>
<td>100s</td>
<td>$O(\log N)$</td>
</tr>
</tbody>
</table>

Rank these algorithms from most efficient to the least efficient (for large N)?

1. $\text{Algo D} \Rightarrow \text{Algo B} \Rightarrow \text{Algo A} \Rightarrow \text{Algo C}$

b- Fill up the following table using Big-O notation for the various algorithms. If some options are not applicable for some algorithms, such as swaps, etc. just left it blank.

<table>
<thead>
<tr>
<th></th>
<th>Comparisons</th>
<th>Shift/Copies</th>
<th>Swaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble sort</td>
<td>$O(N^2)$</td>
<td></td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>Selection sort</td>
<td>$O(N^2)$</td>
<td></td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>$O(N^2)$</td>
<td>$O(N^2)$</td>
<td></td>
</tr>
<tr>
<td>Enhanced Insertion sort</td>
<td>$O(N\log N)$</td>
<td>$O(N^2)$</td>
<td></td>
</tr>
<tr>
<td>List Insertion sort</td>
<td>$O(N^2)$</td>
<td>$O(N)$</td>
<td></td>
</tr>
<tr>
<td>Delete in unordered array</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td></td>
</tr>
<tr>
<td>Binary search in ordered array</td>
<td>$O(\log N)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary search in ordered linked-list</td>
<td>$O(N\log N)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push into a stack</td>
<td></td>
<td></td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Enqueue into a priority queue</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td></td>
</tr>
<tr>
<td>Dequeue a priority queue</td>
<td></td>
<td></td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Shuffle an array</td>
<td></td>
<td></td>
<td>$O(N)$</td>
</tr>
</tbody>
</table>
c- Cite one advantage and one disadvantage of using ordered array vs unordered array?  

Advantage = Fast binary search \(O(\log N)\).  

Disadvantage = Slow insertion \(O(N)\).

d- Cite two main advantages of using linked list rather than arrays?

1. Size of the list can be expanded dynamically.
2. No need to shift/purge items around.

e- What would be the best data structure/strategy/implementation to use to address each particular situation below?

* A store manager would like to keep a detailed inventory count, he/she wants to be able to insert or remove items frequently in/from the inventory, all items will be ranked by price values, and in a booming economy, he/she is looking at the possibility to keep expanding the inventory.

2  Sorted Linked List

* The same store manager is looking at the possibility to sell on-line the current inventory and let customers search for their preferred items?

1  Ordered array

* The on-line sale worked great, so the manager would like to deal with all the orders following a first come/first serve strategy.

1  Queue using linked list

* Some customers bought a privilege membership, how would the store manager adapt to this situation?

1  Priority Queue using linked list

* Once sign in, the customer should have the possibility to keep track of its own purchase and see all the last items (up to 100) he/she recently bought.

1  Stack using array (linked-list possible too).

f- Both Merge sort vs quicksort are divide and Conquer algorithms that run in \(O(N\log N)\). Cite three main differences:

1. Merge sort does not work in-place.
2. Quicksort uses partitioning rather than dividing.
3. Quicksort does not have a worst case (always \(O(n^2)\)).
We consider a class that includes “array” as a private 1D array of `double` of size N.

[12pts] Complete the `findMax` and `swap` methods below.

```java
public int findMax(int start, int end) {
    // return the index of the maximum item in “array” between indexes [start,end]
    int max = start;
    for (int i = start + 1; i <= end; i++) {
        if (array[i] > array[max]) max = i;
    }
    return max;
}
```

```java
public void swap(int i, int j) {
    // swap item “array[i]” with item “array[j]”
    double temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
```

Below you will write a **recursive version** of `selectionSort` (you will use the `swap` and `findMax` methods) that sorts items from the smallest to largest (as seen in class but we used the `findMin` rather than the `findMax` here).

```java
public void recSelectionSort(int start, int end) {
    // base step
    if (start == end) {
        return; // optional
    } else { //recursive step
        swap(findMax(start, end), end);
        recSelectionSort(start, end - 1);
    }
}
```
Quicksort (as seen in class, pivot at the right end). Complete the recursive method below (3 instructions to complete).

```java
public static void recQuickSort(int left, int right) {
    if (right - left <= 0) {
        return; // Base case
    } else {
        System.out.println("pivot " + array[right]);
        int pivotIndex = partition(left, right, array[right]);
        display(); // method that displays the array in-line
        recQuickSort(left, pivotIndex - 1);
        recQuickSort(pivotIndex + 1, right);
    }
}
```

Let us apply the method to the array A below. The method will successively return the pivot value and the display the array after each partitioning.

**Initial array A**

| 12 | 16 | 6 | 2 | 8 | 7 | 14 | 21 | 4 |

1. **pivot = 1**

| 16 | 6 | 2 | 8 | 7 | 14 | 21 | 4 |

2. **pivot = 2**

| 4 | 6 | 2 | 8 | 7 | 14 | 21 | 16 | 14 |

3. **pivot = 4**

| 2 | 6 | 4 | 7 | 8 | 12 | 21 | 16 | 14 |

4. **pivot = 14**

| 2 | 6 | 4 | 14 | 7 | 8 | 12 | 21 | 16 | 14 |

5. **pivot = 21**

| 2 | 4 | 6 | 7 | 8 | 12 | 14 | 16 | 21 |
Let us suppose your prefer instead using ShellSort to sort the array A:
What would be the resulting array just after an increment of 3 is used?

What would be the resulting array just after a new increment of 2 is used?

In General what are the advantages of ShellSort compared to Insertion sort (be precise)?

1) When increment is large, sublist has less items that can move by distance
2) When increment is small, sublists have more items but they are increasingly more suited.

3- Queue and Stack [10pts]

Let us consider a stack. As we have seen with the RPN calculator, it comes in handy to have various utility methods such as copy, swap, etc.

The following code describes the method roll:

```java
public void roll(int j) {
    Object[] temp = new Object[j+1];
    if (size() >= j+1) {
        for (int i = 0; i < j+1; i++)
        {
            temp[i] = pop();
        }
        push(temp[0]);
        for (int i = j-1; i >= 1; i--)
        {
            push(temp[i]);
        }
        push(temp[j]);
    }
}
```
What this method is doing (be precise)?

swap item at the \( (\text{top}) \) with item at \( (\text{top} - 1) \)

What would happen to the output of the following RPN calculator after roll(3) and then roll(1) are entered.

```
5: 3.14
4: 7
3: 4.2
2: 5+i
1: 3.0
0: 4.5+3i
```

[4pts] What is the output of this code?

```java
public static void main(String[] args) {
    Queue myqueue = new Queue(10);
    Stack mystack = new Stack(10);
    for (int i = myqueue.size(); i >= 1; i--) {
        myqueue.enqueue(i);
        myqueue.display(); // display the queue in-line from front to rear
        while (!myqueue.isEmpty()) mystack.push(myqueue.dequeue());
        while (!mystack.isEmpty()) myqueue.enqueue(mystack.pop());
        myqueue.display(); // display the queue in-line from front to rear
    }
}
```

```
10 9 8 7 6 5 4 3 2 1 2
```

4- Recursion [12pts]

**Chebyshev polynomials of the first kind** are defined by the recurrence relation:

\[
T_0(x) = 1 \\
T_1(x) = x \\
T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).
\]

As an example, the polynomial \( T_2 \) would be \( T_2 = 2xT_1(x) - T_0(x) = 2x^2 - 1 \)

We assume that we have defined the object class "polynomial", and we can operate the class as follows:

- `polynomial p=new polynomial('2x+1');` \( \Rightarrow \) instantiate a new polynomial \( p=2x+1 \)
- `p.scalar(2);` \( \Rightarrow \) multiply the polynomial \( p \) by the scalar \( 2 \)
- `p3=prod(p1,p2);` \( \Rightarrow \) static method that performs the product of 2 polynomial \( p1,p2 \) and create a new \( p3 \)
- `p3=sum(p1,p2);` \( \Rightarrow \) static method that performs the summation of 2 polynomial \( p1,p2 \) and create a new \( p3 \)
[6pts] Complete the recursive Cheb function below

```c
polynomial cheb(int n) {
    if (n == 0) return new polynomial(1);
    else if (n == 1) return new polynomial(x);
    else
        return (Sum(prod(new polynomial(1), cheb(n-1)), Scalar(2)), cheb(n-2) * Scalar(1));
}
```

[6pts] For the case of n=7, plot the (divide and conquer) structure of the recurrence including the various calls to cheb and the intermediate polynomial accumulated solutions and the final formula. As a reminder here is what we obtained for Fibonacci.

Solution -> $64x^7 - 112x^5 + 56x^3 - 7x$

```plaintext
V7 = 8x - x^2
V6 = 8x - x^2
V5 = 8x - x^2
V4 = 8x - x^2
V3 = 8x - x^2
V2 = 8x - x^2
V1 = 8x - x^2
V0 = 8x - x^2
```
5. Linked-List - Problem [22pts]

We propose to use a doubly ended doubly linked list to store a series of integer. Each link contains a publicly visible integer called data to store its contents integer.

Example with series = 7, 11, 3, 9

![Doubly Linked List Diagram]

The DoublyLinkedList class contains the standard methods insertFirst, insertLast, deleteFirst, deleteLast.

[4pts] Write the constructor for the DoublyLinkedList class that accepts as argument an array “array” containing the list such as: int[] array = {7, 11, 3, 9} for example and initializes the linked-list (as shown above).

```java
public DoublyLinkedList(int[] array) {
    first = null; // optional
    last = null; // optional
    for (int i = 0; i < array.length; i++) { insertLast(array[i]); }
}
```

[4pts] Write the method isAlone that returns true or false if there is only one item in the list.

```java
public boolean isAlone() {
    if (first == null) return false; // special case
    if (first.next == null) return true;
    return false;
}
```

[4pts] Write the method the displayForward method that prints 7 11 3 9 (in line)

```java
public void displayForward() {
    Link current = first;
    while (current != null) {
        System.out.print(current.data);
        current = current.next;
    }
}
```
[6pts] What does the method ‘surprise1’ below do (be precise)?

```java
public void surprise1()
{
    Link p1, p2;
    int temp;
    p1 = first.next;
    while (p1 != null)
    {
        temp = p1.data;
        p2 = p1;
        while (p2.prev != null && p2.prev.data >= temp)
        {
            p2.data = p2.prev.data; // shift procedure
            p2 = p2.prev;
        }
        p2.data = temp;
        p1 = p1.next;
    }
}
```

Would it be possible to implement this method more efficiently? Explain Why? And Explain How? (no need to provide code here).

The insertion is done by shifting items (like an array). It would be possible to improve the approach by reversing the shifting procedure and inserting at the correct position.

[4pts] For this question we assume that the integer values in the linked-list are sorted (just as an example, we get 3, 7, 9, 11 for the list above). What does the method ‘surprise2’ below do?

```java
public int surprise2()
{
    Link p1 = first;
    if (p1 == null) return 0;
    Link p2 = last;
    while ((p1 != p2) && (p1.next != p2))
    {
        p1 = p1.next;
        p2 = p2.prev;
    }
    if (p1 == p2) return (p1.data);
    else return ((p1.data + p2.data) / 2);
}
```

[return the median value of a list of sorted integer. It was 2 probes from both ends]