## Mid-Term- ECE 242 – Fall 2016
Closed book/notes- no calculator- no phone- no computer

| **NAME:** |  |
| **ID:** |  |

<table>
<thead>
<tr>
<th><strong>Problem</strong></th>
<th><strong>Score</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1- General questions (14pts)</td>
<td></td>
</tr>
<tr>
<td>2- Big-O (18pts)</td>
<td></td>
</tr>
<tr>
<td>3- Sorting (25pts)</td>
<td></td>
</tr>
<tr>
<td>4- Queue (8pts)</td>
<td></td>
</tr>
<tr>
<td>5- Stack (8pts)</td>
<td></td>
</tr>
<tr>
<td>6- Recursion (10pts)</td>
<td></td>
</tr>
<tr>
<td>7- Linked-List (17pts)</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL (100pts)</strong></td>
<td></td>
</tr>
</tbody>
</table>
1- General questions [14pts] (brief answers)
   a- If memory was not an issue, which sorting algorithms would be the best to use for addressing any situations (among the algorithms that we have seen so far)?
   
   **Merge Sort**

   \[ \text{b- If we assume that the data are already sorted, what would be the big-O complexity of insertion sort and quicksort (using pivot on the right)?} \]

   \[ O(N) \quad O(N^2) \]

   \[ 12 \]

   \[ \text{c- What is the main advantage of using a double-ended feature for a linked-list (be precise)?} \]

   \[ \text{One can insert last w/ } O(1) \]

   \[ 12 \]

   \[ \text{d- Cite two main advantages of using a doubly-linked list?} \]

   \[ \text{Traverse backward.} \]

   \[ \text{deleteLast w/ } O(1) \]

   \[ \text{More flexibility to remove / insert objects.} \]

   \[ 12 \]

   \[ \text{e- Fill up the following table using big-O notation for various insertion algorithms. Important, we consider only the number of comparisons and shift/copies for sorting a single item.} \]

   \[ \begin{array}{|c|c|c|}
   \hline
   \text{Insertion sort using array} & \text{Comparisons} & \text{Shift/Copies} \\
   \text{Enhanced Insertion sort using array} & O(N) & O(N) \\
   \text{Insertion sort using Linked-list} & O(N) & \emptyset \text{ or } O(1) \\
   \text{Enhanced Insertion sort using Linked list} & O(N) & \emptyset \text{ or } O(1) \\
   \hline
   \end{array} \]

   \[ 41 \]

   \[ \text{f- How would you use a linked-list to sort an array?} \]

   \[ \text{Insert one by one the items of the array into a sorted linked-list. Copy back the items (deleteLast or deleteLast) into the array.} \]

   \[ 21 \]
2- Big-O [18pts]

a- Find the Big-O complexity of the following code (write it below each code)

```java
for (int i=1; i<N; i=i+2)
    // do something
    \[ O(\log N) \]

for (int i=0; i<N; i++)
    // do something
    \[ O(N) \]

for (int i=0; i<N; i++)
    for (int j=0; j<i; j++)
        for (int k=0; k<10; k++)
            // do something
    \[ O(N^2) \]

for (int i=N; i>0; i--)
    // do something
    \[ O(N^2) \]

for (int i=N; i>0; i=i/2)
    for (int j=1; j<N; j=j+2)
        // do something
    \[ O(10N^2) \]
```

b- Analysis

```java
public void surprise()
{
    Random rnd = new Random();
    int out, index;
    double temp;
    for(out=N-1; out>0; out--)
    {
        index = rnd.nextInt(out+1);
        temp = array[index];
        array[index] = array[out];
        array[out] = temp;
    }
}
```

What is the Big-O of method `surprise`?
\[ O(N) \]

What does it do?
Shuffle an array of double

What would happen if we replace `index=rnd.nextInt(out+1);` by `index=out;`?
nothing
3- Sorting [25pts]
[10pts=3+4+3] Insertion sort. Complete the code. Apply it to the array of Integer A.

```java
public void insertionSort()
{
    int in, out;
    int temp;          // temp variable
    for(out=1; out<N; out++) // outer loop - select key
    {
        temp = array[out]; // save in memory select key item
        System.out.println("pivot=\"+temp);
        in = out;          // start shifting at out
        while(in>0 && array[in-1]>=temp) // To COMPLETE
        {
            array[in] = array[in-1]; //
            in--;                    
        } // end while loop
        array.display(); // display array at each iteration
    } // end outer loop
} // end insertionSort()
```

Initial array A:

```
5 3 8 0 6 2 9 1 4 7
```

```
| 0.5 |
| 3 5 |
| 3 5 |
| 0 3 |
| 0 3 |
| 0 2 |
| 0 1 |
| 0 4 |
```

How many comparisons are used in sorting this sample array A? 1+1+3+2+5+1+7+5+3 = 29

What is the Big-O running time complexity of insertion sort? O(N^2)
[10pts=4+2+2+2]- Let us suppose your prefer instead using ShellSort to sort the array A with an increment of 3 to start with. What would be the resulting array just after an increment of 3 is used?

\[ 5 1 3 8 0 6 2 9 7 4 7 \]

What can you say about this set of data compared to the original one?

It looks more sorted

If you now decide to use ShellSort with increment of 1 (i.e. insertion sort) on this new array, how many comparisons would you get?

\[ 12 \]

What is the Big-O running time complexity of Shellsort (in average)? Comment also on the optimal big-O for insertion sort.

in average \( O(N \log N) \)

If array is almost sorted, case where sort will run optimally with \( O(N) \)

[5pts]- Quicksort (as seen in class, pivot at the right end). Apply it to array A.

**Initial array**

| 5 | 3 | 8 | 0 | 6 | 2 | 9 | 1 | 4 | 7 |

**pivot = 7**

| 5 | 3 | 4 | 0 | 6 | 2 | 7 | 8 | 9 | 5 |

**pivot = 1**

| 1 | 4 | 5 | 6 | 2 | 3 | 7 | 8 | 9 |

**pivot = 3**

| 0 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 |

**pivot = 5**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
4- Queue [8pts]
We consider first a priority queue implemented using an array. We know that removing an item from the front of queue (dequeue) is O(1) while inserting a new item is O(N) (enqueue). Why enqueue is so slow?

Because we first need to perform a linear search to find where to insert the item.  

Can you think on a way to improve a bit the insertion time (which is still O(N))? 

Since the queue is ordered, we could use a binary search, we could create an enhanced version (we cannot improve the shift).  

We now consider the following code:

```java
public static void main(String[] args) {
    Queue[] queue = new Queue[2];
    for (int i=0;i<queue.length;i++)
        queue[i]=new Queue(10);
    for (int i=1;i<10;i++)
        if (i%2==0)
            queue[0].enqueue(i);
        else
            queue[1].enqueue(i);
    for (int i=1;i<10;i++)
        if (!queue[0].isEmpty())
            System.out.println(queue[0].dequeue());
        else
            System.out.println(queue[1].dequeue());
}
```

What is the output of this code?
5- Stack [8pts]

We consider the following code that is making use of the class StackCalc.java (containing the methods: pop, push, size and isEmpty).

class StackCalcApp
{
    public static void main(String[] args)
    {
        StackCalc mystack = new StackCalc(100); //100 max capacity
        EasyIn easy = new EasyIn();
        int n=easy.readInt();
        for (int i=0;i<n;i++) mystack.push(i);
        boolean opadd=true;
        while (mystack.size()>1)
        {
            if (opadd)
            {
                mystack.push((int) mystack.pop() + (int) mystack.pop());
                opadd=false;
            }
            else
            {
                mystack.push((int) mystack.pop() * (int) mystack.pop());
                opadd=true;
            }
        }
        System.out.println("Result is "+mystack.peek());
    }
}

What does this code do (be precise)?

It is pushing n first natural number on the stack in reverse order.
It is then alternating between addition and multiplication
((1+2)*3+4)*5+6)*7...n
the operation act on two popped numbers and put the result back on the stack.

What would be the result after each run if the user choose successively n=1,2,3,4,5, or 6?

n=1 --- 1
n=2 --- 3
n=3 --- 9
n=4 --- 13
n=5 --- 65
n=6 --- 71

\[ \frac{1}{3} \]
6- Recursion [10pts]
We consider the Tribonacci series: 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, ...
And the recurrence relation \((n>2)\): \(T_n = T_{n-1} + T_{n-2} + T_{n-3}\) with \(T_0 = 1; T_1 = 1; T_2 = 2\)

Complete the recursive trib function below

```c
int trib(int n)
{
    // base case
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else if (n == 2) return 2;
    else return (trib(n-1) + trib(n-2) + trib(n-3)); // recursion step
}
```

For the case of \(n=5\), plot the (divide and conquer) structure of the recurrence including the various calls to trib and the intermediate summation. As a reminder here is what we obtained for Fibonacci.
7- Linked-List [7 pts]

We propose to use a doubly double-ended linked list to store a series of natural positive (integer) number: 1,2,3,4,..,n
The link class contains only one item integer named value. Example for n=4

```
1  2  3  4
first  next  next  next  next
      prev  prev  prev
      Null  Null  Null
Last
```

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

a- Write the constructor for the DoublyLinkedList class that accepts “n,” the length of the list, as an argument and initializes the linked-list.

```java
public DoublyLinkedList (int n)
{
    first = null; //optional
    last = null; //optional
    for (int i = 0; i < n; i++)
        insertLast(i);  
}
```

b- We propose to implement a method that will compute the sum (1+2+3+...+n) and return the result. Requirement: the list must be traversed forward.

```java
public int sumForward()
{
    Link current = first;
    int sum = 0;
    while (current != null)
    {
        sum = sum + current.value;
        current = current.next;
    }
    return sum;
}
```
c. We propose to implement a method that will compute the product \((1*2*3*...*n)\) which is \text{fact}(n) and return the result. Requirement: the list must be traversed backward.

```java
public int factBackward()
{
    Link cur = last;
    int fact = 1;
    while (cur != null)
    {
        fact = fact * cur.value;
        cur = cur.prev;
        return fact;
    }
}
```

d. We propose to implement the method "factBackward" above using a recursion. The new method factBackward will call a recursive method named recfactBackward. Write the two methods below.

```java
public int factBackward()
{
    if (!is Empty()) return (recFactBackward(last));
    else return 0;
}
```

```java
public int recFactBackward(Link current)
{
    if (current == null) return 1;
    else return (current.data * recFactBackward(current.prev));
}
```