Mid-Term - ECE 242 - Fall 2015
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 (18pts)</td>
<td></td>
</tr>
<tr>
<td>2- Big-O (18pts)</td>
<td></td>
</tr>
<tr>
<td>3- Stack-RPN (14pts)</td>
<td></td>
</tr>
<tr>
<td>4- Queue/Stack (12pts)</td>
<td></td>
</tr>
<tr>
<td>5- Selection sort+recursion (12pts)</td>
<td></td>
</tr>
<tr>
<td>6- Linked-List (10pts)</td>
<td></td>
</tr>
<tr>
<td>7- Partitioning and quickSort (16pts)</td>
<td></td>
</tr>
<tr>
<td>TOTAL (100pts)</td>
<td></td>
</tr>
</tbody>
</table>
1- General questions [18pts] (brief answers)

a- What is the main advantage and the main disadvantage of using an ordered array vs unordered array and why?

Advantage: Binary search → \( O(\log N) \) vs \( O(N) \) for unordered array

Disadvantage: Insertion → \( O(N) \) vs \( O(1) \) for unordered array

b- Cite two main advantages of using linked-list vs unordered array? Cite one disadvantage.

Advantages:
- Can allocate memory dynamically (\( \text{nil waste} \))
- No need for (shuffling) after insert/remove

Disadvantage: Loss of data locality

c- How can we improve insertion sort? Cite three “similar” algorithms (based on the idea of insertion) and explain why they are expected to be more efficient?

- Enhanced insertion sort, use binary search to reduce \# comparisons
- List insertion sort, use less copies
- Shell sort, use increment \( > 1 \) to reduce both comparisons and copies

d- MergeSort vs Quicksort: cite two main similarities and two main differences.

Similarity
- Divide and conquer algo (can be implemented via recursion)
- \( O(N \log N) \)

Differences
- Merge sort does not work in place
- Quicksort use partitions rather than simple dividing
- Merge sort worst case is worse than simple quicksort case
2- Big-O [18pts]

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

```
for (int i=0; i<N; i=i+2) {
    //do something
}
O(N)
```

```
for (int i=0; i<100; i++) {
    //do something
}
O(1)
```

```
for (int i=0; i<N; i++) {
    for (int j=0; j<i; j++) {
        for (int k=0; k<j; k++)
        { //do something
        }
    }
}
O(N^3)
```

```
for (int i=0; i<N; i=i*5) {
    //do something
}
O(log N)
```

```
//here we consider two
//variables input \( N \) and \( M \)
for (int i=0; i<N; i=i*2) {
    for (int j=0; j<M; j++)
    { //do something
    }
}
O(N \log N)
```

b- Asymptotic analysis: Find the Big-O of the following functions

- \( \frac{N}{2} \) \( O(N) \)
- \( N + 2 \log(N) \) \( O(N) \)
- \( (N+N \log(N))^2 \) \( O(N^2 \log^2 N) \)
- \( 1+2+3+4+...+N \) \( O(N^2) \)
- \( 1+5+25+125+...+N \) \( O(N^2) \)

\( \log \) each

\( +1 \) if all correct
3- Stack-RPN [14pts]
Using the class Stack.java (with interface methods: pop, push, isEmpty, add, multiply, subtract, divide), we can implement a simple RPN calculator.
a- Complete the following code that solves: "(4+2)*8-3" in postfix notation:

```java
class StackApp {
    public static void main(String[] args) {
        Stack mystack = new Stack(5);
        // To complete: Solve "(4+2)*8-3"
        mystack.push(4);
        mystack.push(2);
        mystack.push(8);
        mystack.multiply();
        mystack.push(3);
        mystack.subtract();
        System.out.println("Result is " + mystack.peek()); // 1
    }
}
```

b- Let assume you use the instruction
System.out.println(mystack.peek());
after each new instructions in you code above. Write the successive outputs below:

```
4
2
6
8
48
3
45
```

Result is 45.
c- Very often RPN calculator gives you the possibility to swap the first two items at the top of the stack (which provides more flexibility when entering the postfix notation and manipulating the order of operations). Complete the following code that solves our same expression but where the number 3 has been first pushed into the stack. You can use the new “swap” method when needed.

```java
class StackApp {
    public static void main(String[] args) {
        Stack mystack = new Stack(5);
        mystack.push(3);
        mystack.push(4);
        mystack.push(2);
        mystack.add();
        mystack.push(5);
        mystack.multiply();
        mystack.swap();
        mystack.subtract();
        System.out.println("Result is " + mystack.pop());
    }
}
```

d- Complete the swap method that will be located within the class Stack.java (we assume the item are of type double)

```java
public void swap() {
    double x = pop();
    double y = pop();
    push(x);
    push(y);
}
```
Lemmings follow each other and keep falling down the cliff. Suppose there is one lemming coming in every 1 second (new lemmings arrive at the end of the waiting line → queue). It takes 2 seconds for each lemming to fall down the cliff and stack themselves on the ground (forming then a stack of lemmings).

Complete the following code using the class Queue.java and the class Stack.java composed of integer. The goal is to display the content of the queue at the cliff and the stack on the ground after 12 seconds have passed. We identify each lemming by an integer (lemming 1,2,3, etc. by order of arrival).

The relevant interface methods for Stack are: push, pop, isEmpty
The relevant interface methods for Queue are: enqueue, dequeue, isEmpty

class LemmingsApp
{
    public static void main(String[] args)
    {
        Queue myqueue = new Queue(6);
        Stack mystack = new Stack(6);

        int T=12;  // initialization total time T to 12 seconds
        int time=0; // initialization of variable time
        // Find the content of the queue and stack after T seconds

        while (time < 12) {
            myqueue.enqueue(time+1); // time+1 is the lemming number
            if (time%2 == 0) mystack.push(myqueue.dequeue(1));
            // Note that time%2 can be replaced by (time/2)*2 == time
            time++;
        }
    }
}
// display the content of the queue after T seconds (on a single line)- from front to rear
System.out.println("Queue content :");
while (!myqueue.isEmpty()) System.out.print(myqueue.dequeue() + " ");

System.out.println(" ");
// display the content of the stack after T seconds (on a single line)- front top to bottom
System.out.println("Stack content :");
while (!mystack.isEmpty()) System.out.print(mystack.pop() + " ");

What is the output of this code:

Queue content: 7 8 9 10 11 12 /C
Stack content: 6 5 4 3 2 1 /C
5- Selection sort + Recursion [12pts]
a- The traditional selectionSort method is given below (we consider that “array” is a private array of integer of size N in the class). Complete the findMin and swap methods.

```java
public void selectionSort()
{
    for(int out=0; out<N-1; out++) // outer loop
    {
        // find the minimum item between [out+1,N-1]
        // and swap item “array[min]” with item “array[out]”
        swap(out,findMin(out,N-1)); // comment
    }
} // end selectionSort()
```

```java
public int findMin(int start,int end)
{
    // find the minimum item of “array” between [start,end]
    // To complete
    int min = start;
    for (int i = start+1; i <= end; i++)
    {
        if (A[i] < A[min]) min = i;
    }
    return min;
}
```

```java
public void swap(int i,int j)
{
    // swap item “array[i]” with item “array[j]”
    // To complete
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
```
b- Below you will write a recursive version of selectionSort (you can also use a call to the swap and findMin methods)

```java
public void recSelectionSort(int start, int end)
{
    // base step...to complete
    if (start == end)
        return; // optional.
    else
    {
        // recursive step...to complete
        swap(start, findMin(start, end));
        recSelectionSort(start + 1, end);
    }
}
```

6- Linked-list [10pts]

a- We consider the following doubly and double-ended linked list where we also provide the methods:
displayForward, and insertFirst. Complete the deleteFirst and deleteLast methods.

```java
class Link {
    public int data; // data item
    public Link next; // reference to next link
    public Link prev; // reference to previous link

    // Constructor
    public Link(int data)
    {
        this.data = data;
        next = null; // optional
        prev = null; // optional
    }
}
```
class Doubly2EndedLinkList {

    private Link first; // Reference to the first link
    private Link last; // Reference to the last link

    // constructor
    public Doubly2EndedLinkList() { first = null; last = null; }

    // methods
    public void displayListForward() {
        Link current = first; // start probe at the beginning
        while (current != null) { // until the end of the list
            System.out.println(current.data);
            current = current.next; // move to next link
        }
    }

    public void insertFirst(int data)
    {
        Link newNode = new Link(data); // create link
        if (isEmpty())
            last = newNode; // special case
        else
            first.prev = newNode; // (step 1)
        newNode.next = first; // (step 2)
        first = newNode; // (step 3)
    }

    public Link deleteFirst()
    {
        Link temp = first; // save the link
        if (first == null) return null; // empty list
        if (first.next == null) // if only 1 elem.
            last = null; // special case
        else
            first.prev = null; // step (1)
        first = first.next; // step (2)
        return temp;
    }

    public Link deleteLast()
    {
        Link temp = last; // save the link
        if (last == null) return null; // empty list
        if (first.next == null) // if only 1 elem.
            first = null;
        else
            last.prev.next = null; // step (1)
        last = last.prev; // step (2)
        return temp;
    }
}
7- Partitioning and QuickSort [16pts]
Below are the methods recQuickSort and partition as seen in lecture and discussion (using the right end item as pivot). We also consider here an array of String of size N defined as private variable inside the corresponding class. We also suppose that the swap method is given.

```java
public void recQuickSort(int left, int right)
{
    // size 1- base step- sorted
    if (right-left<=0) return;
    // size 2 or larger- recursive step
    else{
        System.out.println("pivot= " + array[right]); // print pivot array[right] here
        // partition range using pivot at right and return pivotIndex
        int pivotIndex=partition(left, right, array[right]); // print pivot
        for (int i=0; i<N; i++) {System.out.println(array[i] + " ");} //print array content
        // sort left side
        recQuickSort(left, pivotIndex-1);
        // sort right side
        recQuickSort(pivotIndex+1, right);
    }
}

public int partition(int left, int right, double pivot) {
    // here the array of double (for example) is a private variable of the class
    int leftProbe = left-1;    // becomes left after ++leftProbe
    int rightProbe = right;    // becomes right-1 after --rightProbe
    while (true) {
        // search smaller item from left- No need of extra test
        while (array[++leftProbe].compareTo(pivot)<0) { ; }
        // search larger item from right- with extra test if it reaches the end
        while (rightProbe>=left && array[--rightProbe].compareTo(pivot)>0) { ; }
        if (leftProbe >= rightProbe) // cross-over- end of partitioning process
            break;
        else { // found misplaced items; swap
            swap(leftProbe, rightProbe);
        }
        swap(leftProbe, right); // insert pivot
        return leftProbe; // return pivot location
    }
```
We consider the array of Strings of size 9 [0-8]:

i  need  a  good  night  sleep  after  this  exam

You will:
a- Fill-up step by step the arrays below that are supposed to be printed at each time we exit the partition method
b- Indicate also the pivot used in the partition method

pivot= exam

after  a  exam  good  night  sleep  i  this  need 1/2

pivot=..a  /1

a  after  exam  good  night  sleep  i  this  need 1/2

pivot=..need 1/1

a  after  exam  good  i  need  night  sleep  this 1/2

pivot=..i  /1

a  after  exam  good  i  need  night  sleep  this 1/2

pivot=..sleep 1/1

a  after  exam  good  i  need  night  sleep  this 1/2

Final result

a  after  exam  good  i  need  night  sleep  this 1/2