Name: __________________________
Student ID: _____________________

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (16)</td>
<td></td>
</tr>
<tr>
<td>2 (10)</td>
<td></td>
</tr>
<tr>
<td>3 (24)</td>
<td></td>
</tr>
<tr>
<td>4 (20)</td>
<td></td>
</tr>
<tr>
<td>5 (15)</td>
<td></td>
</tr>
<tr>
<td>6 (15)</td>
<td></td>
</tr>
</tbody>
</table>

NOTE: Any questions on writing code must be answered in Java using Data Structures topics covered in lectures 1-10
1. [16 pts]

a) For the following methods, please identify the big-O notation computation complexity, given the size of the linked list or array is n. Please assume that these methods are implemented using efficient techniques. Write a short description for each answer. [8 pts]

i) Print out the node in a singly linked list with a value which matches a specific key value.

A: O(n), since you have to traverse the list to find out the node with this key value.

ii) Swap the middle value and the first value of an array, given the size of the array.

A: O(1), since you need to access the middle node first, which is O(1). Swapping is O(1).

iii) Print out the node with the largest value that is located within the first 10 nodes of a singly linked list. Assume the size of the list is greater than 10.

A: O(1), since this operation needs constant time.

iv) Calculate the average value of all the nodes in an array.

A: O(n), since you need to access all the nodes for at least once.
b) Use big-O notation to summarize the asymptotic behavior of the following functions (logn means log₂n) [8 pts]:

i) \( T_a(n) = n^2 + 10n \log n + 100n + 1000 \)

A: Here \( n^2 \) is the most dominant term. Hence, the \( T_a(n) \) is \( O(n^2) \).

ii) \( T_b(n) = n^{7/2} + 7n^3 \log n + n^2 \)

A: Here \( n^{7/2} \) is the most dominant term. Hence, the \( T_b(n) \) is \( O(n^{7/2}) \).

iii) \( T_c(n) = 4^{(\log(n/2))} + n^{15/8} + 10n \)

A: \( T_c(n) = 4^{(\log(n/2))} + n^{15/8} + 10n = (n/2)^2 + n^{15/8} + 10n \)  
Here \( (n/2)^2 \) is the most dominant term. Hence, the \( T_c(n) \) is \( O(n^2) \).

iv) \( T_d(n) = 6^{n+1} + 6(n+1)! + 24n^{42} \)

A: Here \( 6(n+1)! \) is the most dominant term. Hence, the \( T_d(n) \) is \( O((n+1)!) \).
2. [10 pts]

Write a method `find2C` which accepts an integer array of size n and an integer x. `find2C` returns true if the sum of any two consecutive numbers in the array equals x. Please also identify the complexity of your `find2C` method in big-O notation.

```java
boolean find2C(int[] myArray, int x) {
    for (int i=0; i<myArray.length-1; i++) {
        if (myArray[i]+myArray[i+1] == x) {
            return true;
        }
    }
    return false;
}
```

The method `find2C` has one loop in which i is incremented by 1 in each iteration. So the complexity of the method `find2C` is $O(n)$. 

3. [24 pts]

<table>
<thead>
<tr>
<th>Index</th>
<th>Key</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>10</td>
<td>[6]</td>
</tr>
<tr>
<td>[1]</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>[3]</td>
<td>9</td>
<td>NULL(-1)</td>
</tr>
<tr>
<td>[7]</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

A singly linked list is stored in an array as shown above. Each array element has 2 fields, which are the key value and the next node index (N/A means no value). **The “Next” value indicates the index of the next node (in the list) stored in this array.** The head of the list is shown above at index [0]. Please answer the following questions.

a) What is the index of the tail node of this singly linked list? [2 pts]

[3], since its “Next” is NULL(-1).

b) What is the index of the 3\(^{rd}\) node in this list? [2 pts]

[5], as indicated by [6].

c) A new node is inserted between the 3\(^{rd}\) and the 4\(^{th}\) node of this list and the content is stored in the [7] index of the array (the key value is 999). Please list **ALL** changed parts of this array. [4 pts]

<table>
<thead>
<tr>
<th>Index</th>
<th>Key</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>10</td>
<td>[6]</td>
</tr>
<tr>
<td>[1]</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>[3]</td>
<td>9</td>
<td>NULL(-1)</td>
</tr>
</tbody>
</table>
d) This list is used to implement a stack where nodes are pushed and popped from the beginning of the list. Please write the push() and pop() methods for this stack. Consider the definitions of the Node and the List below in creating your methods. The head of the list is specified as an array index. The index of the new node waiting to be pushed into the stack is also given. [16 pts]

```java
public class Node
{
    int key;
    int next;
}

public static Node[] my_array = new Node[10];

public class List
{
    int head;
}

List myList = new List();

// position indicates the index of a free array location
public void push(Node nn, int position)
{
    if(myList.head == -1)
    {
        myList.head = nn_position;
        my_array[nn_position] = nn;
        nn.next = -1;
    }
    else
    {
        my_array[nn_position].next = myList.head;
        myList.head = nn_position;
        my_array[nn_position] = nn;
    }
}
```
public void pop()
{
    if (my_array[myList.head].next == -1)
    {
        int last_node = myList.head;
        myList.head = -1;
        return last_node;
    }
    else
    {
        int pop_node = myList.head;
        myList.head = my_array[myList.head].next;
        return pop_node;
    }
}
Consider the following question related to a singly linked list SList. Write a method removeSmallest that takes SList mylist as the input and deletes and returns the node with the smallest key (you can assume for this problem that the list contains at least one node and only one node has the smallest key). The class definitions for SNode and SList are provided below.

```java
public class SNode {
    public SNode next;
    public int key;
}

public class SList {
    public SNode head;
}

public SNode removeSmallest (SList mylist) {
    SNode fromHead = mylist.head;
    SNode prevNode = null;
    int tempkey = fromHead.key;
    while (fromHead != null) {
        if (fromHead.key < tempkey) {
            tempkey = fromHead.key;
        }
        fromHead = fromHead.next;
    }
    fromHead = mylist.head;
    while (fromHead != null) {
        if (fromHead.key == tempkey) {
            break;
        } else {
            prevNode = fromHead;
            fromHead = fromHead.next;
        }
    }
    if (fromHead == mylist.head) {
        mylist.head = fromHead.next;
    } else {
        prevNode.next = fromHead.next;
    }
    return fromHead;
}
```
5. [15 pts]

Consider a singly linked list Slist in which the total number of nodes is even. Write a method to eliminate the first half of the list. The modified list should only contain nodes from the second half of the original list. Your method should return the number of nodes in the new list.

```java
public class SNode {
    public SNode next;
    public Object element;
}

public class SList {
    private int list_size;
    public SNode head;

    public int ListSize() {
        return list_size;
    }
}

public int removeHalf(SList myList) {
    SNode secondHalfHead = myList.head;
    for (int i = 0; i < myList.ListSize() / 2; i++) {
        secondHalfHead = secondHalfHead.next;
    }
    myList.head = secondHalfHead;

    return myList.ListSize() / 2;
}
```
6. [15 pts]

You are given a predefined stack and a queue. The stack and the queue contain the same element type (int) and hold the same maximum number of elements. The following functions are available for use:

```java
public class Stack{
    public boolean empty();
    public void push(int n); 
    public int pop(); 
}

public class Queue{
    public boolean empty();
    public void enQueue(int n); 
    public int deQueue();
}
```

a) Write a method to move all the nodes from a stack to a queue. You can do the methods in Stack and Queue classes

```java
public void moveStackToQueue(Stack s, Queue q)
{
    while(q.empty() == false)
    {
        s.push(q.deQueue());
    }
}
```
b) Write a method which will return the number of items in the queue. There is no “size” function provided as part of the class. You are only allowed to use the functions given above.

```java
public void queueSize(Stack s) {
    int queue_size = 0;
    while (!q.empty()) {
        q.deQueue();
        queue_size++;
    }
    return queue_size;
}
```