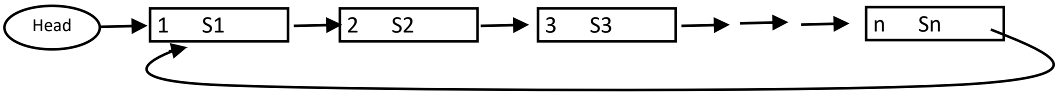
CS210 Data Structure & Algorithms (162) Major Exam 1 Name_________Solution___________ ID___________
Weight: 15% Time: 60 minutes

Important Instructions:
+Use of Mobile phones or any other electronic gadget is prohibited.
+After time is called, hand-in your work. Failure to do so within 3 minutes will result in 20% penalty.
+Time is of essence, use it wisely.

Solve all Questions. Questions 1-2 refer to the following node class.

```
public class node {
  int id;
  String name;
  node next;
  public node(){id=0;name="";}
}
```

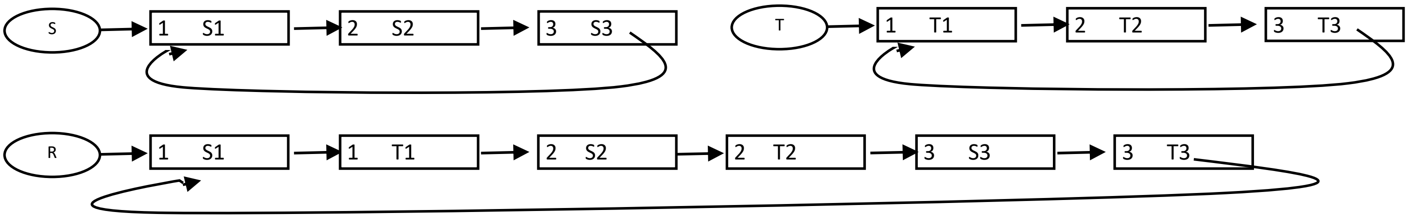
Q1. [3 points] Write a method **public Node buildStructure(int n)** that creates the following structure and returns a reference of type node. The variable ***id*** takes values from 1 up to the parameter ***n***. If n≤0 your method should return null.



```
public static class Node
    {
        int id;
        String name;
        Node next;
        public Node(){id=0;next=null;}
        public Node(int id, String s){this.id = id; this.name=s; next=null;}
    }

    public static Node buildStructure(int n)
    {
        if (n<=0) return null;
        Node temp = new Node(1, "S"+1);
        Node Head = temp;
        for(int i=2;i<=n;i++)
        {
            temp.next=new Node(i, "S"+i);
            temp = temp.next;
        }
        temp.next = Head;
        return Head;
    }
```

Q2. [4 points] Your method public Node MergeLists(node S, node T) is used to merge two lists (using node class given above) in a special way. The method returns a reference ***R*** to a new list by taking first node of List S followed by first node of list T; then second node of List S followed by second node of List T; and so on. <u>Your method checks if the size of both lists is the same; if it is not the same, the method returns null</u>.



```
public static Node mergeLists(Node S, Node T)
    {
        if(size(S)!=size(T))return null;
        Node R = S;
        Node moveS=S, moveT=T;
        int size = size(S);
        for(int i=0;i<size*2;i++)
        {
            if(i%2==0){
                moveS=moveS.next;
                R.next = moveT;

            }
            else{
                moveT=moveT.next;
                R.next = moveS;
            }
            R=R.next;
        }
        return R;
    }
    public static int size(Node X)
    {
        if(X==null) return 0;
        int i=1;
        Node temp = X;
        while(temp.next!=X)
        {
            temp = temp.next;
            i++;
        }
        return i;
    }
```

Q3. [2 points] Trace the output of the following code fragment. Show the contents of the stack object **`myStack`** at each step.

	Preformed Function	**Output**	**State of** `myStack`
1	**`myStack.push(10)`**		**10**
2	**`myStack.push(20)`**		**20,10**
3	**`myStack.pop()`**	**20**	**10**
4	**`myStack.push(2 * myStack.top())`**		**20,10**
5	**`myStack.push(20 - myStack.top())`**		**0,20,10**
6	**`myStack.size()`**	**3**	**0,20,10**
7	**`myStack.pop()`**	**0**	**20,10**
8	**`myStack.isEmpty()`**	**false**	**20,10**

Q4. [3 points] Consider the following code fragments/algorithm in the table below. For each, state the runtime of the algorithm in **big-Oh notation**.

No.	Algorithm	Runtime expressed in big-Oh

1	`//N is a large number`
	`int sum = 0;`
	`for (int n = N; n > 0; n -= 2)`
	`    for(int i = 0; i < n; i++)`
	`          sum++;`
		O(n^2)

2	`//N is a large number`
	`int sum = 0;`
	`for (int i = 1; i < N; i ++)`
	`     for (int j = 0; j < 10; j++)`
	`           sum++;`
		O(n)

3	**Algorithm** Algo (k)
	Input: k , a positive integer
	Output: k-th even natural number (the first even being 0)
	
	if (k = 1) **then return** 0
	else
	   return Algo (k-1) + 2
		O(k) or is k<=n then O(n)

Q5. [3 points] Write a recursive method public **int SumPow(int x, int n)** that computes and returns the sum of all powers ***p*** of ***x*** where *0 < p ≤ n*. Give trace and estimate number of operations with a Big-Oh notation for the run-time.

Example: SumPow(2, 3) would give $2^0 + 2^1 + 2^2 + 2^3 = 15$.

```
public static int sumPow(int x, int n)
    {
        if(n==0)
            return 1;
        else
            return sumPow(x,n-1)+(int)Math.pow(x, n);
    }
```

Example: Lets say x =2 and n=3 then

For n	call
	`sumPow(2, 3)`
3	`sumPow(2,2)+(int)Math.pow(2,3);`
2	`sumPow(2,1)+(int)Math.pow(2,2);`
1	`sumPow(2,0)+(int)Math.pow(2,1);`
0	1

In each call number of operations is 1 (if stmt) + 2 for Math.pow(): total is 3. Overall there would be n calls so 3n + c which is O(n).

-End of Exam-