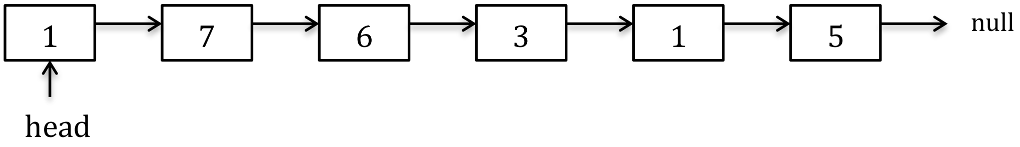
C S 272/463 Introduction to Data Structures

Q1. (20 pts) **(Linked list)** Given the *SNode* class as follows.

```
public class SNode <E>{


    public E data;
    public SNode<E> next = null;
    public SNode()      {; }

    public double func() {
        IntNode cursor;
        int num1 = 0;
        int num2 = 0;

        for (cursor = this; cursor != null; cursor = cursor.link) {
                if(cursor.data%2==1)
                        num1+=cursor.data;
                else
                        num2 +=cursor.data;
        }
        return ((num1)*1.0/num2);
    }

}
```

A. (10 pts) Given the above function ***func()***, what is the returned result of running head.*func()* on the following list.



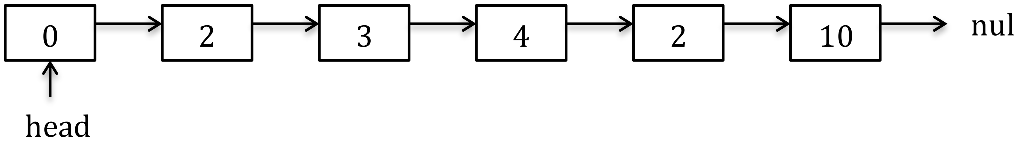
Result:

B. (10 pts) Given the following function in the above *SNode* class.

```
public SNode func(SNode head, E x){
    SNode dummyn = new SNode();
    dummyn.next = head;

    SNode cursorPrev = dummyn;
    SNode cursor = cursorPrev.next;
    while(cursor!=null){
            if(cursor.data.equals(x))
                    cursorPrev.next = cursor.next;
            else
                    cursorPrev = cursorPrev.next;
            cursor = cursor.next;
    }
    head = dummyn.next;

    return head;
}
```

Let *n* be the total number of nodes in the linked list starting from *head*. What is the worst-case complexity of the above **func()** method in Big-O ______________

Given the above function *func()*, show the result of running *func(head,2)* on the following list.



Result:

Q2. (10 pts) **[Stack]** Utilizing the *SNode* class given above, implement an **O(1)** ***push*** method for the class *LinkStack*, this method needs to match the given pop() method.

```
public class LinkStack<E> {
    public SNode<E> top;
    public LinkStack()          {top = null;}

    public void push(E e) {//Insert data to the stack



    }

    public E pop() {
        if(top==null) throw new EmptyStackException();
        E answer = top.data;
        top = top.next;
        return answer;
    }
}
```

Q3. (10 pts) **[Queue]** Given the class *LinkedQueue*.

```
public class LinkedQueue<E> {

    public SNode<E> rear = null;                 //the rear of a queue
    public SNode<E> front = null;                //the front of a queue
    public LinkedQueue(){; }

    public E func1() {
        if(front==null){ return null;}
        else{
                E answer = front.data;
                front = front.next;
                if(front==null) rear = null;
                return answer;
        }
    }

    public void func2(E e) {
        SNode<E> newNode = new SNode<E>();
        newNode.data = e;

        if(rear==null) {
                front = rear = newNode;
        }else{
                rear.next = newNode;
                rear = newNode;
        }
    }
}
```

What will the queue *qu* look like after running the following several lines of code? You need to clearly denote (1) which nodes that the *front* and the *rear* of *qu* point to, and (2) how the nodes in the queue link to each other.

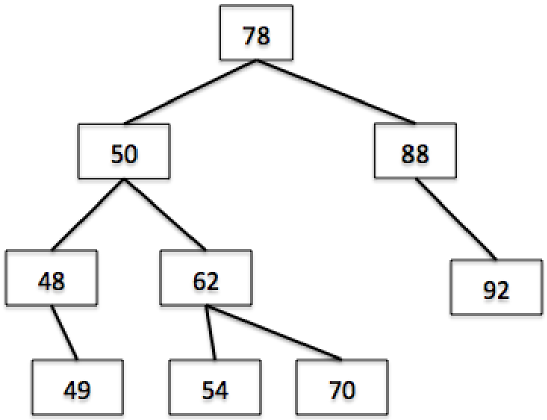
```
LinkedQueue<Integer> qu = new LinkedQueue<Integer>();
qu.func1();
qu.func2(1);
qu.func2(2);
qu.func1();
qu.func2(3);
```

Q4. (20 pts) [**Binary search tree**] Given the classes *BSTNode* and *BST* as follows. Assume duplication values are not allowed in the tree.

```
class BSTNode{
        public int data;             //the element value for this node
        public BST left;             //the left child of this node
        public BST right;            //the right child of this node
        public int height=1;         //height of the tree rooted at this node

        public BSTNode ()                    {data = 0; left = new BST(); right = new BST(); }
        public BSTNode (int initData)  {data = initData; left = new BST();right = new BST();}

}

public class BST {
        public BSTNode root; //instance variable to denote the root of the BST tree
        public BST()      {root = null;}

        public boolean isEmpty() {return (root==null);}

        //Function to find the node that contains e; if e does not exist in the tree, return null
        public BSTNode searchNonRecursion (int e){
                BST cursor = this;
                while ((cursor!=null)&&(cursor.root!=null)){
                        if(e==cursor.root.data){
                                return cursor.root;
                        }else if(e<cursor.root.data){
                                cursor = cursor.root.left;
                        }else{
                                cursor = cursor.root.right;
                        }
                }
                return null;
        }
}
```

A. (10 pts) Given the searchNonRecursion() method, its worst case running time complexity in Big-O is O(log n). Is this statement correct (yes/no)? ______________
If it is correct, please explain. If it is not correct, please give a concreate example to show that the statement is not correct.

Given the following BST tree **t1**, which is rooted at node with value 78.



Given the following function in BST class,

```
private int func(){
        int result=0;

        if(right.isEmpty()){
                result = root.data;
                root = left.root;
        }else{
                result = right.func();
        }
        return result;
}
```

B. (10 pts) After you call t1.left.func(), (1) what will t1 look like and (2) what is the returned value?

Q5. (10 pts) [**Recursive thinking, binary search**] Given the below *binarySearch* function,

```
  // Search e from array A[idxs,…., idxe]
  // If e exists in A[idxs,…., idxe], return its index in A; otherwise, return -1
  public int binarySearch (int[]A, int idxs, int idxe, int e){
        if(idxe<idxs) return (-1);
        int idx_middle = (idxe+idxs)/2;

        if(A[idx_middle]==e) return idx_middle;
        else if(e<A[idx_middle]) return binarySearch(A, idxs, idx_middle,e);
        else return binarySearch(A, idx_middle,idxe,e);
   }
```

Given an array A with content {1, 3, 6, 9, 10, 13}.
Draw the recursion trace of **binarySearch(A, 0, 5,10).**

Q6. (10 pts) **[Heap, Recursive thinking]** Given the following ***Heap*** class which utilizes an array to hold the elements. This heap needs to be a max heap.

```
public class Heap {
        private int[]     elements;
        private int       num;
        public Heap()  {elements=new int[100]}; num=0;}

        public void add(int e){
                elements[num++] =e;
                reheapUpward(elements, num-1);
        }
        public static void reheapUpward(int[] elements, int pos){
                if(pos<=0) return;
                int parentPos = pos/2;
                if(elements[parentPos]<elements[pos]){
                        int tmp = elements[parentPos];
                        elements[parentPos]= elements[pos];
                        elements[pos] = tmp;

                        reheapUpward(elements, parentPos);
                }
        }
}
```

Is there any bug in the provided **reheapUpward** method? If no, explain. If yes, fix the bug.

Q7. (10 pts) [**Open-address hashing**] Given the following *Table* class.

```
public class Table {
    private int num = 0;
    private Object[] keys = new Object[10];
    private Object[] data = new Object[10];
    private boolean[] used = new boolean[10];

    public Table()                  {for(int i=0;i<10;i++) {used[i]=false; keys[i]=data[i]=null;}}
    private int hash(Object key) {return Math.abs(key.hashCode())%data.length; }

    public void func(Object  _key, Object obj) throws Exception{
        if(num==data.length) throw new Exception("Table is full");

        int idx = hash(key);
        int count = 0;
        boolean found = false;
        while(count<data.length & used[idx]){
                if(key.equals(keys[idx])) {found = true; break;}
                else idx = ((idx+1)==data.length)?0:(idx+1);
                count ++;
        }
        if(found==false) idx = -1;

        if(idx!=-1)  data[idx] = obj;
        else{
                idx = hash(key);
                while(used[idx]) {idx = ((idx+1)==data.length)?0:(idx+1);}
                keys[idx] = key;
                data[idx] = obj;
                used[idx] = true;
                num++;
        }
    }
}
```

Assume that your run the following several lines of code:

```
        Table tb = new Table();
        tb.func(1, "o1");
        tb.func(10, "o10");
        tb.func(11, "o11");
        tb.func(5, "o5");
        tb.func(20, "o20");
```

What will be the content of keys, data, and used (Note that you also need to show clearly where the null is)?

Keys[0-9]: ____

Data[0-9]: ____

Used[0-9]: ____

Q8. (10 pts) [**Open question, only for GRADUATE students who take CS 463**] You are given an array of integers. Design an O(n log) algorithm to sort the elements in this array in ascending order by using a **MaxHeap**.

========================== END ==============================