COMS W 3134 - Problem Set #2

Xiaoli Sun (xs2338) - xs2338@columbia.edu

March 3, 2020

Problem 1

```
public static void printLots(List L, List<Integer> P) {
    if(P.isEmpty() || L.isEmpty()) {
        System.out.println("Nothing in the list.");
    } else {
        for(int i=0; i<P.size(); i++) {
            Integer index_object = P.get(i);
            int index = index_object.intValue();
            System.out.println(L.get(index));
        }
    }
}
```

Problem 2

A

In the for loop, when list remove an item, its size will decrease by 1. If we don't save it before the for loop, it will keep changing while we go through the for loop.

B

Arraylist: To remove the i-th item, we need to move all items behind it one step forward.
runtime = O(n^2)

C

Linkedlist: Remove i-th item we need O(1) time.
runtime = O(n)

D

For Arraylist: NOT better.
For LinkedList: Better.

Problem 3

```
public class TwoStacks<T> {
    private T[] A;
    private int size1;
    private int size2;
    private int array_capacity;

    public TwoStacks(int capacity) {
        A = (T[]) new Object[capacity];
        array_capacity = capacity;
        size1 = 0;
        size2 = 0;
    }

    public T peek1() {
        return A[size1-1];
    }

    public T peek2() {
        return A[array_capacity-size2];
    }

    public T pop1() {
        if(size1==0) {
            NullPointerException err = new NullPointerException("Stack Under flow");
            throw err;
        }
        size1 = size1-1;
        T item = A[size1];
        return item;
    }

    public T pop2() {
        if(size2==0) {
            NullPointerException err = new NullPointerException("Stack Under flow");
            throw err;
        }
        T item=  A[array_capacity-size2];
        size2=size2-1;
        return item;
    }
    public void push1(T item) {
        if(size1+size2==array_capacity) {
            StackOverflowError exp = new StackOverflowError("Stack Overflow Error.");
            throw  exp;
        }
        A[size1] = item;
        size1 = size1+1;
    }

    public void push2(T item) {
        if(size1+size2==array_capacity) {
            StackOverflowError exp = new StackOverflowError("Stack Overflow Error.");
            throw  exp;
        }
        size2 = size2+1;
        A[array_capacity-size2] = item;
    }

    public int getSize1() {
        return size1;
    }

    public int getSize2() {
        return size2;
    }

    public boolean isEmpty1() {
        if(size1 == 0) {
            return true;
        } else {
            return false;
        }
    }

    public boolean isEmpty2() {
        if(size2 == 0) {
            return true;
        } else {
            return false;
        }
    }

}
```

Problem 4-1

O(n) time and any extra space

Traverse down the linked list from its head and use a stack to store the data in the node of the linked list. We need length(linked list) extra memory to store the data in the stack. Because stack is LIFO, we can pop and print the data in the stack until it's empty. Thus we print the list in reverse order.
From linked list to stack : O(n) time
Print the stack: O(n) time
total running time: O(n)
total extra space = the size of stack = the length of linked list = O(N)

```
public void printreverse(LinkedList list) {
    MyStack<T> S = new MyStack<T>();
    Node<T>  curNode = list.head;
    while(curNode!=null) {
        S.push(curNode.data);
        curNode = curNode.next;
    }
    while(S.isEmpty()==false) {
        System.out.println(S.pop());
    }
}
```

Problem 4-2

any runtime and O(1) extra space

In the i-th round, traverse from 0-th node (head of the list) to the (length-i-1)-th node, and print data in this node. Repeat this procedure n=length(list) times. We print the list in reverse order.

Extra space: integer n and Node pointer cueNode. space=2=O(1)
runtime: (n-1)+(n-2)+...+0 = O(n^2)

```
public void printL(LinkedList l) {
    int n = l.length();
    for(int i=0; i<n;i++) {
        Node curNode = l.head;
        for(int j=0; j<n-i-1; j++) {
            curNode = curNode.next;
        }
        System.out.println(curNode.data);
    }
}
```