CS 61B
Spring 2021

Linked Lists & Arrays

Topical Review Section 2: February 7, 2021

1 Linked List Practice

Here's a basic `SLList` class we've defined. Assume the `SLList` constructor is properly implemented and creates a sentinel with a placeholder value. Use `SLList` to answer the following parts.

```
public class SLList {
    private class IntNode {
        public int item;
        public IntNode next;
        public IntNode(int item, IntNode next) {
            this.item = item;
            this.next = next;
        }
    }

    private IntNode sentinel;
    private int size;

    public void addFirst(int x) {
        this.sentinel.next = new IntNode(x, this.sentinel.next);
        this.size += 1;
    }
}
```

(a) Implement `addLast(int x)`, a method of `SLList` that creates a new `IntNode` and adds it to the back of our `SLList`

```
public void addLast(int x) {








}
```

(b) Notice that this is quite slow for long `SLLists`, why? How can we change `SLList` to make this faster?

(c) Let's create a Doubly Linked List class. The DLList should be able to support a fast insertion at both the front and back of the list. Assume the DLList constructor is already implemented and creates a sentinel node with a placeholder value properly. Also assume sentinel.next points to the first node in the list, and sentinel.prev points to the last node. Fill in the blanks below:

```
public class DLList {
    private class IntNode {
        public int item;



        public IntNode(int item, IntNode next, IntNode previous) {




        }
    }

    private IntNode sentinel;
    private int size;

    public void addFirst(int x) {
        this.size += 1;
        IntNode oldFront = this.sentinel.next;
        IntNode newNode =



    }

    public void addLast(int x) {
        this.size += 1;
        IntNode oldBack = this.sentinel.prev;
        IntNode newNode =



    }

}
```

(d) Implement `destructiveReverse`, a method of `DLList` that destructively reverses the values of our `DLList`. For example, if our list is 1 ↔ 3 ↔ 5 ↔ 7, then `destructiveReverse` should modify the list to be 7 ↔ 5 ↔ 3 ↔ 1. `destructiveReverse` should modify **values only**, not pointers.

```
public void destructiveReverse() {
    if (this.size == 0) {
        return;
    }
    IntNode lPointer =
    IntNode rPointer =
    int lIndex = 0;
    int rIndex = this.size - 1;
    while (___________________________________________________) {
        int temp =








    }
}
```

2 ArrayLists

Use the following class structure to answer the following parts below.

```java
public class AList {
  private int[] items;
  private int size;
  private int FACTOR = 2;

  public AList() {
    items = new int[100];
    size = 0;
  }

  public int getLast() {
    return items[size - 1];
  }

  public int get(int i) {
    return items[i];
  }

  public int size() {
    return size;
  }

  private void resize(int capacity) {
    int[] a = new int[capacity];
    System.arraycopy(items, 0,
                     a, 0, size);
    items = a;
  }
}
```

(a) Implentation the `removeLast(int x)` method that ”removes” and returns the int value at the end of the `AList` by setting it to null. You do not have to resize down in this implementation.

```java
public int removeLast() {






}
```

(b) Finish the implentation of the `addLast(int x)` method that adds an int at the end of the `AList` (index=size). The method should take into account the case when items has no more space available and increase the capacity of items by a factor of FACTOR. Feel free to use any helper methods available in the code above.

```
public void addLast(int x) {
    if (__________________________________) {



    }




}
```

(c) Your friend would love to use your AList class for Proj0 of his SC16p class at UCLA. However, he needs your AList class to have a method that allows him to remove and return values at specific indices. Since you go to the Number 1 public university in the United States, he requests you to implement `remove (int index)` which removes and returns the element at the index. Assume index is in [0, size) and that the method in part a works as intended.

```
public int remove(int index) {



    for (__________________________________) {


    }





}
```

3 ArrayLists vs LinkedLists

Consider the following scenarios. Choose between a LinkedList or an ArrayList implementation, and explain your reasoning.

(a) Keeping a list of the current stock of products in a supermarket where each stock item is numbered.

(b) Managing a list of unprocessed orders at a fast food restaurant.

(c) Keeping track of the grades you have for each class as you progress through the semester.