CS 6301.502. Implementation of advanced data structures and algorithms

Fall 2017

Short Project 4: Recursion

Fri, Sep 15, 2017

Version 1.0: Initial description (Fri, Sep 15).

Due: 11:59 PM, Sun, Sep 24.

Solve as many problems as you wish. Maximum score: 50

1. [20 points]

Suppose you are given a doubly-linked list class with elements stored

(similar ro SLL) using an entry class as follows:

static class Entry<T> {

T element;

Entry<T> prev, next;

}

You realize that this is isomorphic to the entry class for binary trees:

static class Entry<T> {

T element;

Entry<T> left, right;

}

Write a recursive function to convert a doubly-linked list, in sorted order,

into a height-balanced binary search tree, if we interpreted prev as left,

and next as right. Assume that lists are implemented with dummy headers.

These are member functions of the list class, and do the work by rearranging

references (pointers), without allocating additional space for elements.

Write another function for the inverse problem: BST to sorted list.

Signatures:

// Precondition: list is sorted

void sortedListToBST() { ... }

// Precondition: data is arranged as a binary search tree

// using prev for left, and next for right

void BSTtoSortedList() { ... }

2. [20 points]

Write functions to compute the nth Fibonacci number and compare

their running times. Since the numbers grow fast, use BigInteger

class to represent the numbers.

// Do a simple linear scan algorithm: Fib[n] = Fin[n-1] + Fin[n-2];

// Since numbers are stored with BigInteger class, use add for "+"

static BigInteger linearFibonacci(int n) { ... }

// Implement O(log n) algorithm described in class (Sep 15)

static BigInteger logFibonacci(int n) { ... }

3. [30 points]

Comapre 4 different versions of mergesort discussed in class (Sep 15)

and evaluate the improvement in running times on int[] arrays.

(1) Merge sort as described in text books, where temp array is

allocated in each instance of merge.

(2) Keep one temp array that is passed as a parameter to merge.

(3) Improvement in (2) + use insertion sort for base case when

the size of array is below some threshold.

(4) Improvements in (2) and (3) + avoid copying to tmp array.

4. [20 points]

Binary search: in class we saw a version of binary search that returned

a boolean to indicate whether x occurs in the array or not.

Rewrite the function to return the index of the largest element that

is less than or equal to x.

// Preconditions: arr[0..n-1] is sorted, and arr[0] <= x < arr[n-1].

// Returns index i such that arr[i] <= x < arr[i+1].

public static<T extends Comparable<? super T>> int binarySearch(T[] arr, T x)

5. [30 points]

Reorder an int array A[] by moving negative elements to the front,

followed by its positive elements. The relative order of positive numbers

must be the same as in the given array. Similarly, the relative order of

its negative numbers should also be retained. Write an algorithm that

runs in O(nlogn), and uses only O(1) extra space (for variables),

but can use O(log n) space for recursion.

void rearrangeMinusPlus(int[] arr) { ... }

6. [20 points]

Given an array of n distinct integers, in sorted order, starting

at 1 and ending with n+k, find the k missing numbers in the sequence.

Your algorithm should run in O(k+log(n)) time. Note that a simple

linear scan of the array can find the answer, but it will not meet

the requirement on the running time.