Seoul National University

M1522.000900 Data Structure

Spring 2025, Kang

Homework 3: Non-Binary Trees & Internal Sorting

(Chapters 6 & 7)

Due: **23:59, May 20th (Tuesday), 2025**

Reminders

- Lead TA: Seungho Kim (snuds.ta@gmail.com)
- The points of this homework add up to 100.
- All assignments must be done individually.
- **Type your answer in English** and submit your assignment **in PDF format**.
  - Answer written in Korean may get 0 points.
  - You will get a 10% deduction on your grade for this homework if your submission is either hand-written or non-PDF format.
- Name your file as “(studentID)-(name)-HW3.pdf”.
  (e.g., *202512345-GildongHong-HW3.pdf*)
- Whenever you are making an assumption, state it clearly.
- If you have any questions about the assignment, post them on eTL.

Submission

- Submit your assignment to **eTL**.
- The submission after the due date will be regarded as a late **submission, even if it is only one second late.** Late submissions are accepted **within only one week after the due date.**
- You do not need to specify whether to use the slip-days; they are automatically used.

Question 1 [10 points]

The following code prints the sequential representation of a general tree. The general tree is implemented using the ‘Node’ class. The ‘toSequential’ function takes the root of the tree as an argument and returns the sequential representation of that tree. In this representation, the end of each node’s subtree is marked with a special symbol ‘)’. Your tasks are as follows: (1) Fill in the blanks in the code so that it correctly generates the sequential representation of the tree. (2) Write the expected output when the program is executed.

```
class Node {
    String val;
    List<Node> children;
    Node(String val) {
        this.val = val;
        this.children = new ArrayList<>();}}
public class TreeSerializer {
    public static List<String> toSequential(Node root) {
        List<String> result = new ArrayList<>();
        dfs(root, result);
        return result;
    }
    private static void dfs(Node node, List<String> result) {
        if (node == null) return;
        result.add( [          ] );    // blank(1)
        for (Node child : node.children) {    // blank(2)
            [          ]
        }
        result.add( [          ] );    // blank(3)
    }
    public static void main(String[] args) {
        Node a = new Node("A");
        Node b = new Node("B");
        Node c = new Node("C");
        Node d = new Node("D");
        Node e = new Node("E");
        a.children.add(b);
        a.children.add(c);
        a.children.add(d);
        d.children.add(e);
        List<String> result = toSequential(a);
        System.out.println(String.join(" ", result));}}
```

Question 2 [15 points]

For this question only, handwriting is permitted. However, if the handwriting is unclear or illegible to the grader, points will be deducted.

(1) The following table represents a general tree using an array-based Leftmost Child / Right Sibling representation. Draw the corresponding general tree. ‘X’ indicates ‘null’. [7 points]

Index	Left	Val	Par	Right
0	1	R	X	X
1	4	A	0	2
2	X	B	0	3
3	6	C	0	X
4	X	D	1	5
5	X	E	1	X
6	X	F	3	X

(2) Convert the general tree constructed in Question 2-(1) into its corresponding binary tree using the Leftmost Child / Right Sibling representation. Draw the binary tree. [8 points]

Question 3 [15 points]

Using the weighted union rule and path compression, show the array for the parent pointer implementation that results from the following series of equivalences on a set of objects indexed by the values 0 through 15. Initially, each element in the set should be in a separate equivalence class. When two trees to be merged are the same size, make the root with greater index value be the child of the root with lesser index value.

(0,1) (2,3) (4,5) (6,5) (3,5) (8,9) (7,8) (2,7) (10,11) (12,15) (13,15) (14,13) (6,11) (0,14)

Node	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Parent																

Question 4 [10 points]

Analyze the space overhead of the “list of children” implementation, and the “left-child/right-sibling” implementation. Indicate the size of the data value as D, the size of point as P, and the size of an index as I.

Question 5 [10 points]

Given the following array, sort in ascending order using three iterations of the bubble sort algorithm. Use the last swapped optimization when necessary. After each iteration, rewrite the array on the following row. (Note: 3a and 3b both have the numeric value 3; the letter labels are only there so you can track whether the algorithm keeps equal elements in their original relative order.)

41	25	11	3a	75	12	3b	10	49	53

Question 6 [10 points]

Figure 1 shows an optimized merge sort implementation to sort an array of integers in Java. If there are $n = 2^m - 1$ calls to the merge sort function (mergeSort) to sort an array, how many calls will there be to insertion sort (insertionSort)?

```
public void mergeSort(int[] a, int[] temp, int l, int r){
    int mid = (l + r) / 2;
    if(l == r) return;
    // sort the left part
    if(mid - l > Threshold) mergeSort(a, temp, l, mid);
    else insertionSort(a, l, mid);
    // sort the right part
    if(r - mid > Threshold) mergeSort(a, temp, mid + 1, r);
    else insertionSort(a, mid + 1, r);
    // merge
    for(int i = l; i <= mid; i++) temp[i] = a[i];
    for(int i = 1; i <= r - mid; i++) temp[r - i + 1] = a[mid + i];
    for(int i = l, j = r, k = l; k <= r; k++){
        if(temp[i] < temp[j]){
            a[k] = temp[i]; i++;
        }
        else{
            a[k] = temp[j]; j--;
        }
    }
}
```

Figure 1. An optimized merge sort implementation in Java

Question 7 [20 points]

Write your answers to each of the questions below. Unless otherwise stated, assume the worst-case time complexity. However, make sure you choose the tightest Big-Oh upper bound possible for the operation. Do not use an amortized analysis for these operations unless otherwise specified.

(1) What is the best-case time complexity of running Merge Sort on an already sorted array? [5 points]

(2) What is the best-case time complexity of running Quick Sort when the minimum value is always selected as the pivot index? [5 points]

(3) Suppose we modify the deterministic version of the quick-sort algorithm so that, instead of selecting the last element in an n-element sequence as the pivot, we choose the element at index [n/2]. What is the running time of this version of quick-sort on a sequence that is already sorted? Answer in Big-Oh notation. [5 points]

(4) Daniel has a comparison-based sorting algorithm that sorts the first k elements of a sequence of size n in O(n) time. Give a Big-Oh characterization of the biggest that k can be? [5 points]

Question 8 [10 points]

Here is an array which has just been partitioned by the partition function shown in Figure 2:

33, 0, 21, 42, 59, 82, 77, 65, 91

Which of these elements could have been the pivot? If there are more than one possibilities, list them all.

```
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {    // Move bounds inward until they meet
        while (A[++l].compareTo(pivot) < 0) ;
        while ((r != 0) &&
                (A[--r].compareTo(pivot) > 0)) ;
        DSutil.swap(A, l, r);
    } while (l < r);
    DSutil.swap(A, l, r);
    return l;
}
```

Figure 2. 'partition' function of quick sort