CSP2348 Data Structures

Workshop Test 2: 17APR15 1600-1800

Martin Ponce: Student 10371381

Question 1

1.1

A binary tree has 600 nodes. What is the maximum possible depth of the tree? And what is the minimum possible depth of the tree?

Answer

Minimum depth

$d = \text{floor}(\log_2 600)$
$\text{floor}(\log_2 512) = 9$
$\text{floor}(\log_2 1024) = 10$
$d = 9$

Maximum depth

$d_{max} = n - 1$
$d_{max} = 600 - 1$
$d_{max} = 599$

1.2

List the key properties (or features) of a binary search tree.

Answer

- For any node in the binary tree, if that node contains element *elem*, then:
  - Its left subtree (if not empty) contains only elements less than *elem*
  - Its right subtree (if not empty) contains only elements greater than *elem*

Question 2

The two elementary array sorting algorithms, Selection and Insertion, have the same time complexity of $O(n^2)$ on average case. However, their performances, in terms of number of comparisons, can be quite different in some special cases even on average cases. Compare the two algorithms and determine:

2.1

If A[] is already sorted, which of the above-mentioned sorting algorithms may achieve an O(n) time complexity to complete the sorting procedure? Explain your answer using an example.

Answer

Selection sort

Selection sort would achieve $O(n^2)$ since it would still need to execute its inner loop.

Insertion sort

Insertion sort would achieve O(n) since a sorted array would stop it from executing its inner while loop.

Example:

Index	Value
0	1
1	2
2	3
3	4
4	5

```
public static void insertionSort(Comparable[] a, int left, int right) {

    for (int r = left + 1; r <= right; r++) {
        Comparable val = a[r];
        int p = r;

        while (p > left && val.compareTo(a[p - 1]) < 0) {
            a[p] = a[p - 1];
            p--;
        }

        a[p] = val;
    }
}
```

- First iteration:
  - Outer loop:
    - `r` = 1
    - `val` = 2
    - `p` = 1
  - Inner loop
    - `p > left` = true
    - `val.compareTo(a[0] < 0)` = false
  - Exit inner loop
    - `a[p] = val` = 2
    - value 2 remains at index 1

The condition to enter the inner while loop is never satisfied, therefore the time complexity for insertion sort on a sorted list is O(n).

2.2

Which algorithm is better to sort A[0...n-1], in general case, in terms of the number of comparisons? Why?

Answer

Selection sort

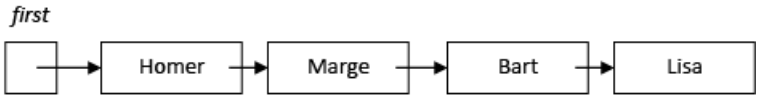
Takes the current element and swaps it with the smallest value on the right side of the current element. Time complexity is **always** n(n - 1) / 2 because the algorithm always executes its inner loop.

Insertion sort

Takes the current element and inserts it at the correct position in the list. Time complexity is n(n - 1) / 2 for **worst case**, which is sorting an unsorted array. The best case is O(n), when the given array is sorted due to the algorithm not executing its inner loop.

Question 3

Given the following linked list, write one Java-like sentence/s to insert a node containing element "Maggie" in between nodes containing "Homer" and "Marge".



Answer

```
public class MyLinkedList<E> {

    private Node<E> head;
    private Node<E> cursorCurrent;

    // ... constructor + other methods here

    /**
     * SLL: Insert after target.
     */
    public void insertAfterSLL(E data, E target) {

        cursorCurrent = head;

        for(MyLinkedList list = this; cursorCurrent != null; cursorCurrent = cursorCurrent.getNext()) {

            if(cursorCurrent.getData().equals(target)) {

                Node<E> insert = new Node<E>(data, cursorCurrent.getNext());

                cursorCurrent.setNext(insert);
                length++;

                System.out.println("[" + data + "]" + " inserted AFTER " +
                                   "[" + target + "]: \n" + this);
                System.out.println("Head: " + "[" + head.getData() + "]");
                System.out.println("Length: " + length + "\n");

                return;
            }
        }

        System.out.println("[" + data + "]" + " NOT inserted, " + "[" + target + "]" +
                           " was not found in the list!\n" + this);
        System.out.println("Head: " + "[" + head.getData() + "]");
        System.out.println("Length: " + length + "\n");
    }

    /**
     * Node subclass.
     *
     * @param <E>
     */
    class Node<E> {

        private E data;
        private Node<E> next;

        // ... constructor + other methods here

        public E getData() {
```

```
            return data;
        }

        public E getNext() {
            return next;
        }

        public E setNext(E next) {
            this.next = next;
        }
    }

}
```

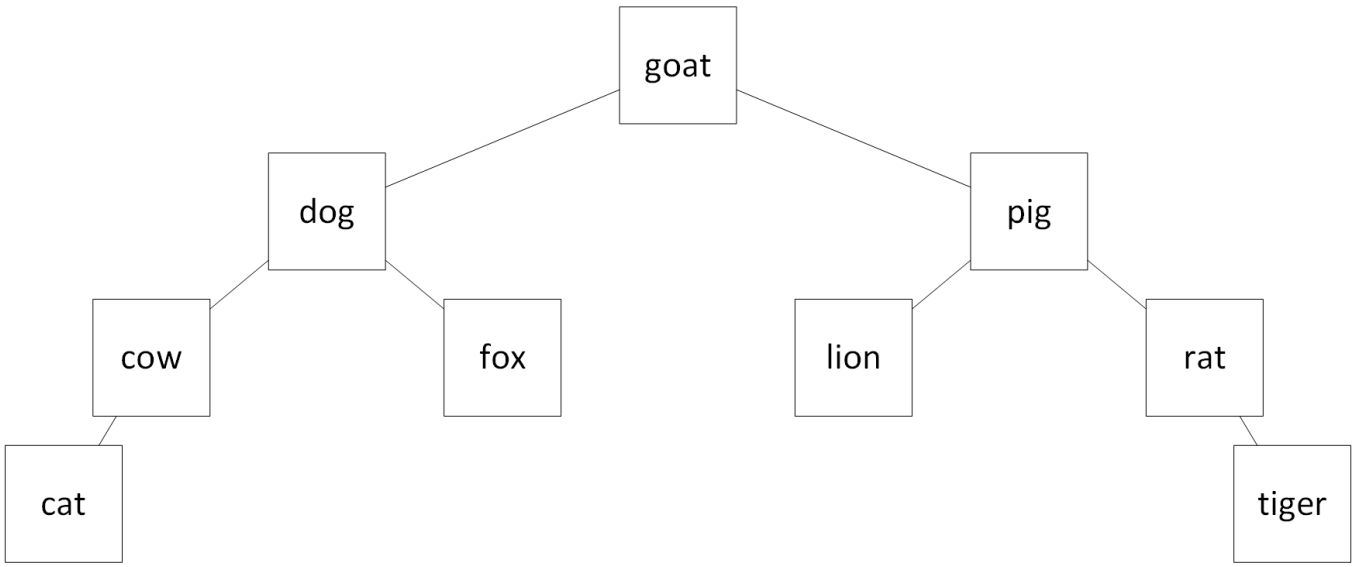
Question 4

4.1

Insert the following animals into an empty BST (you may show the final BST only):

> goat, pig, dog, cow, rat, cat, tiger, fox, lion

Answer



4.2

After the insertions, show the results of pre-order, in-order, and post-order traversals of the BST.

Answer

Pre-order (VLR)

1. goat
2. dog
3. cow
4. cat
5. fox
6. pig
7. lion
8. rat
9. tiger

In-order (LVR)

1. cat
2. cow
3. dog
4. fox
5. goat
6. lion
7. pig
8. rat
9. tiger

Post-order (LRV)

1. cat
2. cow
3. fox
4. dog
5. lion
6. tiger
7. rat
8. pig
9. goat

Question 5

An array A[0 ... n-1], where n > 10, stores the result of assignment 1 of this unit. Assume that all marks are integers, and the array is unsorted. By a preliminary rule, all students who achieved a mark on the top-10 mark-list of the class will be qualified to receive an award.

5.1

Write an algorithm, print_top10(A, n), that prints the 10 top marks of the class.

Answer

Using merge sort to sort array, then linear algorithm to print top 10 elements in array.

```
public void print_top10(int[] array) {
    mergeSortInit(array);

    // since list is in descending order, print from last to n - 10
    for(i = array.length - 1; i <= array.length - 10; i--) {
        System.out.println("Score is: " + array[i]);
    }
}

public void mergeSortInit(int[] array) {

    // assign data array and length
    dataArray = array;
    length = array.length;

    // create helper array, will store sorted array
    helperArray = new int[length];

    // call mergeSort()
    mergeSort(0, length - 1);
}

public void mergeSort(int low, int high) {
```

```
    // 1.0 If left < right
    if(low < high) {

        // 1.1 Let m be an integer about midway between left and right
        int middle = low + (high - low) / 2;

        // 1.2 Sort a[left...m] into ascending order
        mergeSort(low, middle);

        // 1.3 Sort a[m+1...right] into ascending order
        mergeSort(middle + 1, high);

        // 1.5 Merge a[left...m] and a[m+1...right] into auxiliary array b
        // call merge() which is O(n)
        merge(low, middle, high);
    }
}

private void merge(int low, int middle, int high) {

    // iterate from low through to high
    for(int i = low; i <= high; i++) {

        // copy each dataArray element into helperArray
        helperArray[i] = dataArray[i];
    }

    // 1.0 Set i = low, set j = middle + 1, set k = low
    int i = low;
    int j = middle + 1;
    int k = low;

    // 2.0 While i <= middle AND j <= high, repeat:
    while(i <= middle && j <= high) {

        // 2.1 If helperArray[i] <= helperArray[j],
        if(helperArray[i] <= helperArray[j]) {

            // 2.1.1 Copy helperArray[i] into datArray[k], then increment i and k
            dataArray[k] = helperArray[i];
            i++;

            // 2.2 If helperArray[i] > helperArray[j],
        } else {

            // 2.2.1 Copy helperArray[j] into dataArray[k], then increment j and k
            dataArray[k] = helperArray[j];
            j++;
        }
        k++;
    }

    // 3.0 While i <= middle,
    while(i <= middle) {

        // 3.1 Copy helperArray[i] into dataArray[k], then increment i and k
        dataArray[k] = helperArray[i];
        k++;
```

```
            i++;
        }
}
```

5.2

Analyse your algorithm using O-notation.

Answer

Sorting: O(n log n)
Printing: O(n)
Max: O(n)