Université Libanaise
Faculté des Sciences 1

# I2206 E

## Data Structures

**Dr. Zein Al Abidin IBRAHIM &**
**Dr. Ali Jaber**

**Final 2023 - 2024**
**Friday 7 June 2024**
**Duration 90 minutes**

## Question I: Applications [35 pts]

1. A hash table of **length 11** is shown below. The following keys were inserted in the order from left to right [54, 77, 94, 89, 14, 45, 35, 76] into the table. Fill in the keys in the appropriate index if Linear Probing method is used. Use the modulo as a hash function.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |

2. A hash table of **length 13** is shown below. The following keys were inserted in the order from left to right [54, 77, 94, 89, 14, 45, 35, 76, 22, 48, 17] into the table. Fill the keys in the appropriate index if Quadratic Probing method is used. Use the modulo as a hash function.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | |

3. A hash table of **length 15** is shown below. The following keys were inserted in the order from left to right [10, 25, 20, 35, 5, 30, 40, 45] into the table. Use the modulo as the primary hash function and h2(k) = 1+(k mod 14) as the secondary hash function.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | |

4. Fill in the table below a max-based heap (implemented as an array) after the following data is inserted into it (from left to right): [50, 30, 40, 20, 25, 35, 10, 5, 15, 3, 2, 1].

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |

5. Given the following min-heap tree represented by the following array [1,3,5,7,9,11,15,12,13,17,19,21,25], fill in the table below that represents the new heap after removing two elements from the tree.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |

6. Given the content of the array [25, 17, 31, 13, 2, 28, 10, 23, 45, 5, 19, 37]. Suppose we applied **one iteration** of the quicksort algorithm in which the pivot (key) is chosen as the first one, fill in the resulted array.

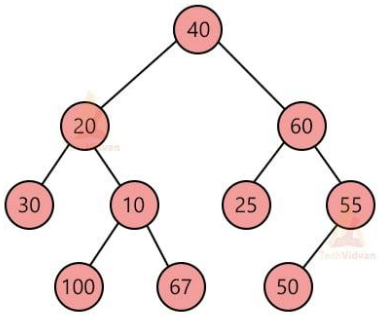
| **Iteration** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | | | | | | | | | | | | |

7. Given the content of the array [25, 17, 31, 13, 2, 28, 10, 23, 45, 5, 19, 37]. Suppose we applied three iterations of the selection sort algorithm, fill in the resulted array.

| **Iteration** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | | | | | | | | | | | | |
| **2** | | | | | | | | | | | | |
| **3** | | | | | | | | | | | | |

8. Given the following binary tree in which you are asked to fill the table on its left for the different types of traversals.

| Traversal | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| In-order | | | | | | | | | | |
| Post-order | | | | | | | | | | |
| Pre-order | | | | | | | | | | |
| Level-order | | | | | | | | | | |



Binary Tree

9. Given the content of the array [25, 17, 31, 13, 2, 28, 10, 23, 45, 5, 19, 37]. Draw below the **BST** that corresponds to the insertion of the elements of the array from left to right. **Then**, draw the new BST that results from removing the values 17 then 2 from the BST that you have drawn first.

| //Draw here the BST | //Draw here the same BST but after you remove from obtained BST the values 17 and then 2 |
|---|---|
| | |

10. Fill in the time complexities for each operation (Insert, Search, and Delete) concerning the provided data structures. Assume the following specifications:

- For array and linked lists: The insert is at the end and the delete is at the beginning.
- For binary tree and BST, consider that you are inserting and removing from balanced trees.
- For hash table, consider the worst-case scenario.

| Data Structure/Operation | Insert | Search | Delete |
|---|---|---|---|
| **Array** | | | |
| **Linked List** | | | |
| **Binary Tree** | | | |
| **Binary Search Tree** | | | |
| **Heap Tree** | | | |
| **Hash Table (Open Addressing)** | | | |
| **Hash Table (Separate Chaining)** | | | |

## Question II: Complexity [20 pts]

1. What is the complexity of the following piece of code? Prove that in the box below.

```
int fun(int n) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j < n; j += i) {
            printf("Hello");
        }
    }
}
```

| | |
|---|---|
| a. O(log(n)) | b. O(n) |
| c. O(n log(n)) | d. $O(n^2)$ |

2. What is the complexity of the following piece of code? Prove that in the box below.

```
for (int i = 1; i < n; i++) {
    i *= k;
}
```

| | |
|---|---|
| a. $O((\log_k(n))$ | b. O(n) |
| c. O(nlog(n)) | d. O(n/log(n)) |

3. What is the complexity of the following piece of code? Prove that in the box below.
   Given the following recurrence relation that we want to derive the complexity of such problem:

$$T(n) = \begin{cases} 16T(n/4) + n & if\ n > 0 \\ 1 & otherwise \end{cases}$$

| | |
|---|---|
| a. O(log(n)) | b. O(n log(n)) |
| c. $O(n^2)$ | d. $O(n^2 \log(n))$ |

4. What is the complexity of the following piece of code? Prove that in the box below.

```
void fun(){
      int i, j;
      for (i = 1; i <= n; i++)
            for (j = 1; j <= log(i); j++)
                  printf("GeeksforGeeks");
}
```

| | |
|---|---|
| a. O(log(n)) | b. O(n) |
| c. O(nlog(n)) | d. $O(n^2)$ |

5. What is the complexity of the following piece of code? Prove that in the box below.

```
void fun(int n) {
    if (n == 0) {
        return;
    }
    printf("%d\n", n);
    for (int i = 0; i < n; i++) {
        fun(n - 1);
    }
}
```

| | |
|---|---|
| a. O(n) | b. $O(n^2)$ |
| c. O(n!) | d. $O(n^n)$ |

## Question III: Algorithms [45 pts]

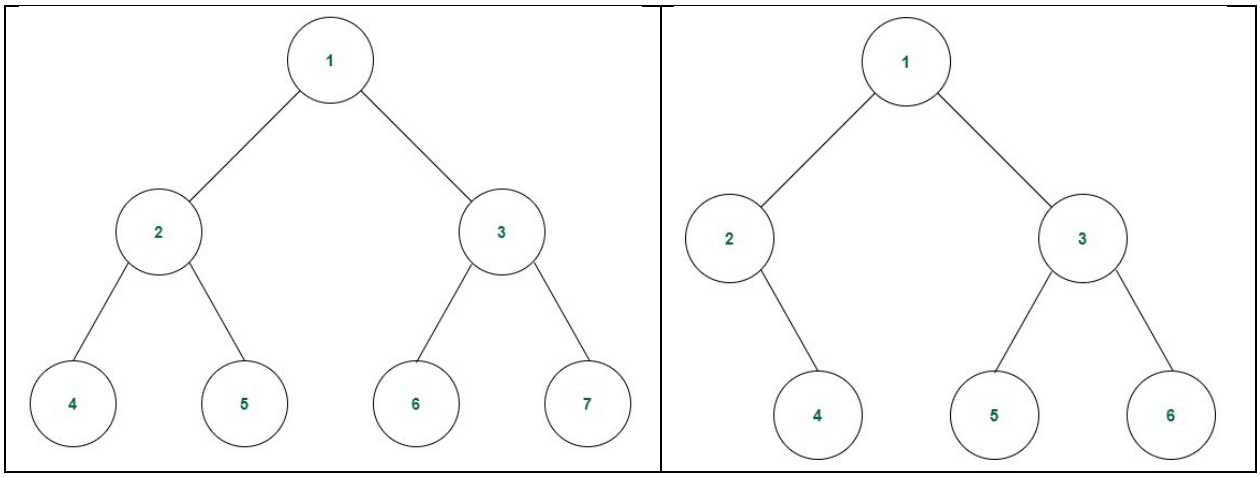
In the following questions, use the below representations for a binary tree node and a linked list node:

```
typedef struct ListNode{
     element data;
     struct ListNode *next;
} ListNode;
```

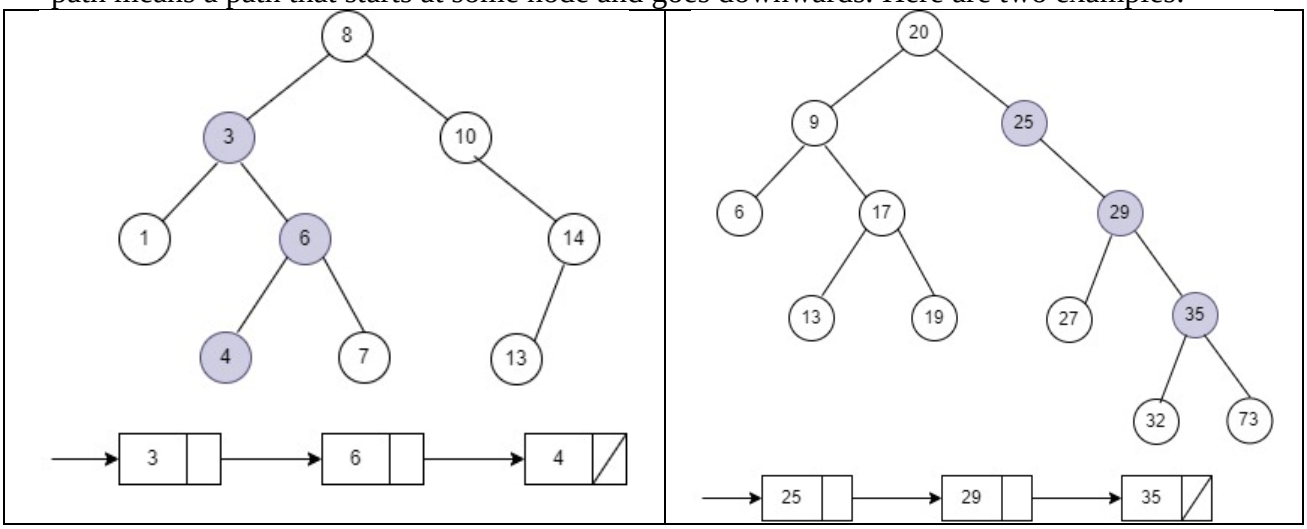
```
typedef struct BTNode{
     element data;
     struct BTNode *left, *right;
} BTNode;
```

1. A perfect binary tree is a special type of binary tree in which all the leaf nodes are at the same depth, and all non-leaf nodes have two children. In simple terms, this means that all leaf nodes are at the maximum depth of the tree, and the tree is completely filled with no gaps. In the following figure, the left figure is a perfect binary tree while the right is not. In this question, you are asked to write a function that checks if a binary tree is a perfect one or no. You are allowed to write the method recursively or iteratively and use the ADT stack or queue if you need.

   **Hint**: write a function to calculate the depth of the tree. Then all leaf nodes should be at the same depth. Recall that at level 0 (the root), we have $2^0$ nodes, at level 1 we have $2^1$ nodes, at level i we have $2^i$ nodes.

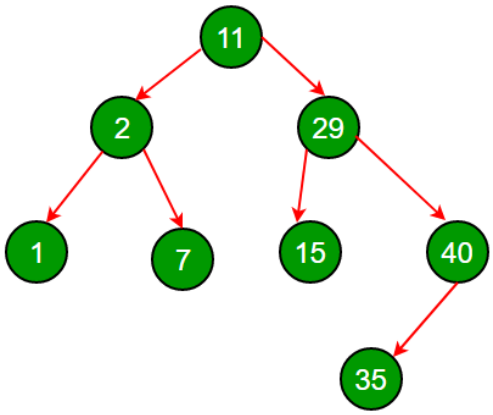


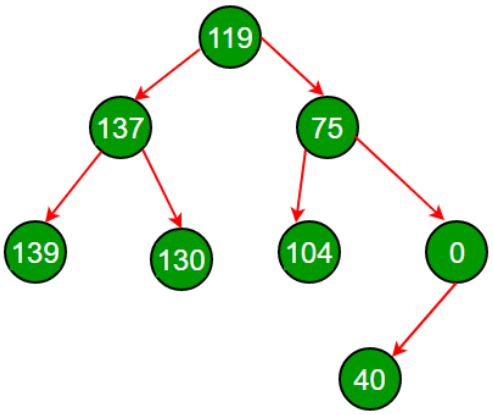
2. Given a **binary search tree** root and a linked list with head as the first node. We want to write the function that returns 1 if all the elements in the linked list starting from the head correspond to some *downward path* connected in the binary tree otherwise return 0. In this context downward path means a path that starts at some node and goes downwards. Here are two examples:



**Hint:** The given problem can be solved with the help of the DFS Traversal of the Binary tree. During the DFS traversal, if any node of the Binary Tree is equal to the head of the linked list, a recursive or iterative function can be called to check whether the other elements of the linked list also exist as a path from that node. If the complete linked list has been traversed, there exists a valid required path, hence return **1**. Otherwise, return **0**.

3. Given a **BST**, transform it into a greater sum tree where each node contains the sum of all nodes greater than that node. Your solution should not exceed the complexity O(n).
   Hint: Think to the reverse order of the in-order traversal to do that.



Scratch

**Pay attention, you should keep all the scratch papers stapled in the booklet exam.**

Scratch

Scratch

Scratch