University of May 8, 1945 - Guelma
Faculty of Mathematics and Computer Science
Department of Computer Science
2nd Year Engineering - Computer Science
2024/2025
Dr. Chemseddine Chohra

Exam: Algorithms and Data Structures (Duration: 2 hours)

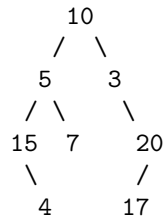
Questions: (6 pts)

Choose the correct answer (only one):

1. The time complexity of O(n^2) means that:
   - A. The algorithm performs a number of operations proportional to the square of the input size. **(Correct Answer)** **(1 pt)**
   - B. The algorithm performs a number of operations proportional to the input size.
   - C. The algorithm performs a constant number of operations independent of the input size.
2. In a tree of height `h`, the maximum number of nodes is:
   - B. 2^h - 1. **(Correct Answer)** **(1 pt)**
   - A. 2^h.
   - C. 2^h + 1.
3. The bubble sort algorithm has a time complexity of:
   - A. O(n).
   - B. O(n log n).
   - C. O(n^2). **(Correct Answer)** **(1 pt)**
4. Among the following sorting algorithms, which one has a space complexity of O(n)?
   - A. Insertion sort.
   - B. Selection sort.
   - C. Merge sort. **(Correct Answer)** **(1 pt)**
5. In a tree, we call a **leaf** a node that:
   - A. Has a null value.
   - B. Is not connected to the root.
   - C. Has no children. **(Correct Answer)** **(1 pt)**
6. To perform a search in a sorted array, the algorithm with the lowest time complexity is:
   - A. Linear search.
   - B. Binary search. **(Correct Answer)** **(1 pt)**
   - C. Manual search.

Exercise 1: (3 pts)

Consider the following binary tree:



1. Give the inorder traversal of the tree.
   Solution:
   Inorder traversal: $15, 4, 5, 7, 10, 3, 17, 20$ **(0.75 pts)**

2. Is the tree a binary search tree? Justify.
   Solution:
   No, **(0 pts)** the tree is not a binary search tree. In a binary search tree, the left subtree of a node must contain only nodes with values less than the node's value, and the right subtree must contain only nodes with values greater than the node's value. Here, the node with value 15 is in the left subtree of 10, which violates the BST property. **(0.75 pts)**

3. Find two nodes to swap so that the tree becomes a binary search tree.
   Solution:
   Swap the nodes with values 15 and 3. **(0.75 pts)**

4. Give the inorder traversal of the tree after the swap.
   Solution:
   Inorder traversal after swap: $3, 4, 5, 7, 10, 15, 17, 20$ **(0.75 pts)**

Exercise 2: (6 pts)

The symmetric difference of two sets A and B is the set of elements that belong exclusively to A or B, but not to both. Write a function **symmetricDifference** that takes as input two sorted integer arrays A and B and returns a sorted array containing the elements of the symmetric difference of A and B.

- **Assumptions**:
  - The arrays **A** and **B** are already sorted and contain no duplicates.
  - The array **C** is created outside the function with sufficient size to hold the elements of the symmetric difference.
- **Constraint**: The function must have linear time complexity (O(na + nb)).
- **Hint**: You can take inspiration from the merge function used in the merge sort algorithm.
- You are free to write your code in any programming language of your choice. However, you are not allowed to use predefined functions in high-level languages.

Solution:
Here is a possible implementation in C:

```c
int symmetricDifference(int A[], int na, int B[], int nb, int C[]) {
    int i = 0, j = 0, k = 0;          // 0.5 pts
    while (i < na && j < nb) {        // 0.5 pts
        if (A[i] < B[j]) {            // 1.5 pts
            C[k++] = A[i++];
        } else if (A[i] > B[j]) {     // 1.5 pts
            C[k++] = B[j++];
        } else {                      // 1.5 pts
            i++;
            j++;
        }
```

```
    }
    while (i < na) {                    // 0.25 pts
        C[k++] = A[i++];
    }
    while (j < nb) {                    // 0.25 pts
        C[k++] = B[j++];
    }
}
```

Exercise 3 - MI: (5 pts)

The following function is supposed to delete the minimum element from a binary search tree. However, it contains some errors.

```
typedef struct Node* Tree;
typedef struct Node {
    int data;
    Tree *left, *right;
};

Tree deleteMin(Tree root) {
    if (root == NULL) return 0;
    if (root->left == NULL) {
        free(root);
        return root->right;
    } else {
        root->left = deleteNode(root);
    }
    return root;
}
```

- Identify, explain, and correct the errors in the `deleteMin` function.
  Solution:
  Errors and corrections:

  1. **Error 1:** The function `deleteNode` is called instead of `deleteMin` in the recursive call. **(0.5 pts)**
     Correction: Replace by a recursive call to `deleteMin`. **(0.5 pts)**
  2. **Error 2:** The recursive call is made on `root`. **(0.5 pts)**
     Correction: The recursive call should be made on `root->left`. **(0.5 pts)**
  3. **Error 3:** The function returns `0` when `root` is `NULL`, which is incorrect. **(0.5 pts)**
     Correction: Return `NULL` instead of `0` when `root` is `NULL`. **(0.5 pts)**
  4. **Error 4:** The function does not handle the case where `root->left` is `NULL` correctly. **(0.5 pts)**
     Correction: Store `root->right` in a temporary variable before freeing `root` to avoid memory issues. **(1.5 pts)**

  Corrected Code:

```
Tree deleteMin(Tree root) {
    if (root == NULL) return NULL;
    if (root->left == NULL) {
        Tree temp = root->right;
        free(root);
        return temp;
    } else {
        root->left = deleteMin(root->left);
    }
    return root;
}
```