University Ferhat Abess-Sétif 1
Faculty of sciences
Computer sciences Departement
Module: ASD3
Level: 2nd-year Computer Science Engineering Students **Dr. Lakehal Abderrahim**

TD/TP 2

Objectives: In this TD, we will explore the fundamentals of algorithm analysis and develop an understanding of time and space complexity using Big O notation through various algorithmic problems.

In this TP, we will implement the source code for each algorithm in this series and measure the exact execution time for different input sizes n.

Instructions:

TD: Analyze each algorithm and determine both its time and space complexity using Big O notation, providing a clear explanation for each case.

TP: Implement each algorithm from this series in C++, then measure and record its exact execution time using the `<chrono>` library for different input sizes n (e.g., 1000, 5000, 10000, ...). Compare the measured times and analyze how they grow with n.

A. Basic Complexity Estimation

For each code fragment, determine the time complexity (in Big O notation):

a)

```
for (int i = 0; i < n; i++)
    cout << i;
```

b)

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        cout << i + j;
```

c)

```
for (int i = 1; i < n; i *= 2)
    cout << i;
```

d)

```
for (int i = 0; i < n; i++)
    for (int j = i; j < n; j++)
        cout << i + j;
```

B. Binary Search Analysis

Code (iterative solution):

```
int binarySearch(int arr[], int n, int target) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target)
            return mid;
        else if (arr[mid] < target)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
```

1. Determine **time complexity** in Big O notation.
2. What is its **space complexity** (iterative vs recursive version)?

C. Tower of Hanoi

Code:

```
void hanoi(int n, char from, char aux, char to) {
    if (n == 1) {
        cout << "Move disk 1 from " << from << " to " << to << endl;
        return;
    }
    hanoi(n - 1, from, to, aux);
    cout << "Move disk " << n << " from " << from << " to " << to << endl;
    hanoi(n - 1, aux, from, to);
}
```

1. Write the **recurrence relation** of the problem.
2. Solve it to find the **closed form**.
3. Determine **time and space complexity**.
4. Discuss how many **moves** are required for n = 3, 4, 5

D. Merge Sort (Divide and Conquer)

Code (Recursive Version):

```
void merge(int arr[], int left, int mid, int right) {
    int sizeLeft = mid - left + 1;
    int sizeRight = right - mid;

    int leftArray[sizeLeft], rightArray[sizeRight];

    for (int i = 0; i < sizeLeft; i++)
        leftArray[i] = arr[left + i];
    for (int j = 0; j < sizeRight; j++)
        rightArray[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < sizeLeft && j < sizeRight) {
        if (leftArray[i] <= rightArray[j])
            arr[k++] = leftArray[i++];
        else
            arr[k++] = rightArray[j++];
    }

    while (i < sizeLeft) arr[k++] = leftArray[i++];
    while (j < sizeRight) arr[k++] = rightArray[j++];
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
```

1. Write the recurrence relation.
2. Determine **time complexity** and **space complexity**.

E. Home work (Fibonacci Sequence)

Code of Fibonacci function (Recursive Version):

```
int fibonacci(int n) {
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

1. Write the recurrence relation.
2. Solve it approximately.
3. Determine **time complexity** and **space complexity** (hint T(n-1) ≈ T(n−2)).