Lab 4: Recursion

ELEC 278: Fundamentals of Information Structures

Learning Goals

- Implement common recursive algorithms (e.g., factorial, sum of array elements) and identify the key components that make up a recursive function.
- Learn to calculate and compare the time complexity of recursive and iterative solutions for the same problem.

Instructions

Download the zipped folder `lab4.zip` from OnQ and unzip it. Next, open the lab4 folder that you extracted in your code editor. You will complete the following exercises by completing the functions in the `main.c` file.

To receive full marks, your code must be able to correctly handle corner cases. Comment your code and be ready to discuss how your code handles corner cases with your TA. Generate a list a cases that you will test your code against with your TA.

Grading (fraction of exercise marks)

0	No code or lack of understanding of the code.
1/2	Functioning code, but fails corner cases, if any, or not readable/not well-explained.
1	Functioning code that handles corner cases

Exercise 1. (1 mark) Write a function `int factorial_iterative(int n)` to calculate and return the the factorial of n, i.e., $n!$ using an iterative approach.

For example, $4! = 4 \times 3 \times 2 \times 1 = 24$. Note that $0! = 1$

Exercise 2. (1 mark) Write a function `int factorial_recursive(int n)` that calculates and returns the the factorial of n, i.e., $n!$ using a recursive approach.

What is your base case to end the recursion?

Exercise 3. (0.5 mark) Compare the run-time complexities of `factorial_iterative` and `factorial_recursive`. In your code, comment next to each line or code section its run-time complexity, and explain how you derived the run-time complexity of the entire algorithm.

Exercise 4. (1 mark) Write a function `int sumArray_iterative(int arr[], int size)` to calculate the sum of elements in an array using a loop.

For example, the sum of `arr` $[1, 2, 3, 4]$ with `size` 4 is 10

Exercise 5. (1 mark) Write the function `int sumArray_recursive(int arr[], int size)` to calculate the sum of elements in an array using recursion.

What is your base case to end the recursion?

Exercise 6. (0.5 mark) Compare the run-time complexities of `sumArray_iterative` and `sumArray_recursive`. In your code, comment next to each line or code section its run-time complexity, and explain how you derived the run-time complexity of the entire algorithm.

Exercise 7. (2 mark) You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists, i.e., do not allocate new memory to new nodes.

Return the head of the merged linked list.

For example, given list1: $1 \to 2 \to 4$ and list2: $1 \to 3 \to 4$, return $1 \to 1 \to 2 \to 3 \to 4 \to 4$

Complete the functions `struct Node* mergeLists_iterative(struct Node* head1, struct Node* head2)` and `struct Node* mergeLists_recursive(struct Node* head1, struct Node* head2)` with iterative and recursive algorithms, respectively.

Exercise 8. (4 mark)

In an operating system, tasks are scheduled using a singly linked list, where each task is represented by a node. The tasks are typically executed first-in first-out, but higher priority tasks are moved ahead of lower priority tasks. Two tasks of the same priority are executed first-in first-out

For example, adding tasks in the following order:

```
newTask(&queue, 1, 3);
newTask(&queue, 2, 4);
newTask(&queue, 3, 2);
newTask(&queue, 4, 3);
newTask(&queue, 5, 5);
newTask(&queue, 6, 2);
```

creates the following task schedule:

```
Task 5 with Priority 5
-> Task 2 with Priority 4
-> Task 1 with Priority 3
-> Task 4 with Priority 3
-> Task 3 with Priority 2
-> Task 6 with Priority 2
```

- Complete a **recursive function** for the following operation: `void newTask(Schedule* queue1, int task_id, int task_priority)` that adds a new task to the task schedule and places it in the queue per its priority.
- Complete a function: `Task* execute(Schedule* queue1)` that returns and removes the task at the front of the queue.