Homework # 01

Guidelines:

- Attempt all questions by yourself before discussing with peers, this is practice to strengthen your concepts.
- For coding questions, try writing clean, readable code on paper.
- Since this homework is ungraded, focus on learning rather than using LLMs to generate codes and get answers.
- If you get stuck, you are encouraged to:
  - Post your doubts on the course Slack channel.
  - Visit the TAs during office hours for guidance.

Topics: Experimental Analysis, Big-Oh Analysis, Arrays, Linked Lists, Stacks, Queues and their Applications

Section 1: Experimental and Asymptotic Analysis

Q1a. Two developers, Alex and Ben, create algorithms for a data processing task.

- Alex's algorithm has a time complexity of **$O(n^2)$**.
- Ben's algorithm has a time complexity of **$O(2^n)$**.

They run their algorithms on small test datasets with n = 2, 3, and 4. They observe that for these values of n, Ben's $O(2^n)$ algorithm runs significantly faster than Alex's $O(n^2)$ algorithm. Ben claims that his algorithm is therefore superior and will always be a better choice.

Is Ben's statement correct? Explain why the experimental results might differ from what the Big-O notation suggests for these small inputs.

Q1b. Which algorithm (Alex's or Ben's) would be faster for a large input size, such as n = 500,000? Provide your reasoning based on their time complexities.

Q2. (a) Express the following functions in terms of Big-O notation (tightest upper bound):

a. $10n^4 + 50n^2 + 300$

b. $n * \log(n) + 3n + 500$

c. $n^2 + 2^n$

d. $n^3 + \log(n^4)$
e. sqrt(n) + log(n)

(b) State if each of the following is True or False.
a. $100n^2 + 2n + 5 \in O(n^3)$
b. $n \log n \in O(n)$
c. $5^n \in O(2^n)$
d. $n! \in O(n^n)$
e. $1000 \in O(1)$

Q3. Consider the following code snippet. Determine its best-case and worst-case time complexity in Big-O notation. Explain your reasoning.

```
 void processData(int arr[], int n, int key) {
  if (arr[0] == key) {
      cout << "Key found at the beginning!" << endl;
      return;
  }

  for (int i = 0; i < n; i++) {
      for (int j = 1; j < n; j = j * 2) {
          cout << "Processing item: " << arr[i] << " and " << j << endl;
      }
  }
}
```

Q4. (a) Determine the worst-case time complexity in Big-O notation for the complexFunction below.

```
 void complexFunction(int n) {
  for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            for (int k = 0; k < j; k++) {
                // some O(1) operation
            }
        }
    }
}
```

(b) Count the number of primitive operations in the countOperations function below and determine its worst-case time complexity in Big-O notation.

```
 int countOperations(int n) {
  int operations = 0;
  operations++; // for initialization
  int i = n;
  operations++; // for initialization
  while (i > 1) {
      operations++; // for the while check
      // some O(1) work
      operations++;
      i = i / 2;
      operations++; // for the division/assignment
  }
  operations++; // for the final while check
  return operations;
}
```

Section 2: Arrays and Linked Lists

Q5. You are tasked with developing the "Undo" feature for a text editor. This feature must support two main operations:

1. A new action (like typing a character or deleting a word) must be recorded.
2. The user must be able to "undo" the most recently recorded action, effectively reversing it.

The system should be highly efficient for these two operations, as they will be used frequently. Which data structure would you choose to store the user's actions: a **dynamic array** or a **stack**? Justify your choice by analyzing the time complexity of the required operations for each structure.

Q6. State if each of the following is True or False. If a statement is false, provide a brief justification. Assume the lists are standard implementations unless stated otherwise.

1. Accessing the element at index k in a singly linked list is an O(1) operation.
2. Inserting an element at the beginning of a dynamic array is an O(1) operation on average.
3. In a doubly linked list, deleting a given node (for which you have a direct pointer) is an O(1) operation.
4. A key advantage of a circular linked list is that it allows traversal from the last node to the first node in O(1) time.
5. If memory usage is the absolute top priority, a dynamic array is always more memory-efficient than a linked list.

Q7. You are designing a system to manage a web browser's history. The system needs to efficiently support the following operations:

1. **Visit a new page:** Add a new URL to the end of the history.
2. **Go back:** Move from the current page to the previously visited page.
3. **Go forward:** Move from the current page to the next page (only possible after going back).

You are deciding between a **singly linked list**, a **doubly linked list**, and a **dynamic array**. Which data structure would be the best choice? Analyze the time complexities of all three operations for each data structure to justify your answer.

Section 3: Stacks and Queues

Q8a. A stack processes the following sequence of operations: push('A'), push('B'), pop(), push('C'), push('D'), pop(), pop(), push('E'). What is the final content of the stack, from top to bottom?

Q8b. A queue initially contains the numbers 10, 20, 30 (with 10 at the front). The following sequence of operations is performed: enqueue(40), dequeue(), enqueue(50), enqueue(dequeue()). What is the final content of the queue, from front to rear?

Q9a. What would be the final contents of a **queue** after the following sequence of operations?
enqueue(5), enqueue(10), enqueue(15), dequeue(), enqueue(20), enqueue(dequeue()), dequeue(), enqueue(25)

Q9b. What would be the final contents of a **stack** after the following sequence of operations?
push(1), push(2), push(3), pop(), push(pop()), push(4), pop(), push(5)