Student Name _____ Student Number _____

1. The time complexity of traditional bubble sort is $O(n^2)$. Please design an optimization to reduce unnecessary comparison operations in a partially sorted array and provide the optimized pseudocode.

The main change is the addition of a flag: In the optimized version, a `swapped` variable is introduced to track whether a swap occurred during each pass. If no swap occurs, it indicates that the array is already sorted, and the program will exit early.

Additionally, early termination: If no swap occurs during a particular pass, the optimized algorithm will use `break` to end the sorting early, thus avoiding unnecessary multiple passes.

```
function optimizedBubbleSort(arr):
    n = length of arr
    for i from 0 to n-1:
        swapped = false   // Position flag to check if there is a swap

        for j from 0 to n-i-2:
            if arr[j] > arr[j+1]:
                swap arr[j] with arr[j+1]
                swapped = true

       // If no exchange occurs, exit early
        if not swapped:
            break
```

original one:

```
function bubbleSort(arr):
    n = length of arr
    for i from 0 to n-1:
        for j from 0 to n-i-2:
            if arr[j] > arr[j+1]:
                swap arr[j] with arr[j+1]
```

2. Given an integer array of length $n = 8$, `[64,34,25,12,22,11,90,42]`, please simulate the process of selection sort, write out the corresponding array state changes.

Initial array: `[64,34,25,12,22,11,90,42]`

e.g., **Array after pass 1** (`i == 0`): `[11,34,25,12,22,64,90,42]`

Array after pass 2: `[11, 12, 25, 34, 22, 64, 90, 42]`

Array after pass 3: `[11, 12, 22, 34, 25, 64, 90, 42]`

Array after pass 4: `[11, 12, 22, 25, 34, 64, 90, 42]`

Array after pass 5: `[11, 12, 22, 25, 34, 64, 90, 42]`

Array after pass 6: `[11, 12, 22, 25, 34, 42, 90, 64]`

Array after pass 7: `[11, 12, 22, 25, 34, 42, 64, 90]`

- 一句话概括：从**未排序的子序列**中找到**最小（本题）的元素**，将它与**未排序子序列的第一个元素**进行交换。

3. Please explain the core idea of insertion sort in your own words (2-3 sentences) and analyze efficiency in following two scenarios:

(1) The array is completely unsorted.
(2) The array is nearly sorted.

The Core: (e.g.,)

- Insertion sort maintains a sorted portion of the array by inserting each element one by one. During each insertion, larger elements are shifted to find the correct position for the new element.

Key: Find insertion position, element movement

关键点：找到插入的位置、元素的移动

(1) Each element must be compared to all previously sorted elements, requiring n^2 comparisons and shifts for an array of size n. The time complexity $O(n^2)$, because every element is placed at the beginning of the array, (leading to the maximum number of shifts)

(2) The time complexity is $O(n)$. Because the algorithm only performs minimal shifts and comparisons for each element already close to its correct position.

4. Merge sort is a sorting algorithm based on the divide-and-conquer paradigm. Please explain the meaning of "divide-and-conquer" (1-2 sentences) , and fill in the gaps in the code.

(a) Your description:

e.g,.

It is a problem-solving approach where a problem is divided into smaller subproblems, each of which is solved independently (the "divide" step). Then, the solutions to these subproblems are combined to solve the original problem (the "conquer" step).

(b) The code:

```
def merge_sort(arr):
    if ____(1)_____:   # Base case
        return arr
    mid = len(arr) // 2
    left = ______(2)_______  # Recursively call merge_sort on the left half
    right = merge_sort(arr[mid:])  # Recursively call merge_sort on the right half
    return ____(3)_____


def merge(left, right):
    result = []
    i = j = 0
    while _______(4)_______:  # Loop condition
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])  # Add remaining elements from the left array
    result.extend(right[j:])  # Add remaining elements from the right array
    return result
```

```
23
```

(1) len(arr) <= 1

(2) merge_sort(arr[:mid])

(3) merge(left, right)

(4) i < len(left) and j < len(right)

5. When using a non-recursive method for quick sorting, one stack is usually used to remember the two endpoints of the interval to be sorted. Can queues be used to implement this stack? Why?

Queues can be used instead of stacks. During the quick sorting process, one-time partitioning allows you to divide an interval to be sorted into two subintervals, which are then divided equally. The purpose of a stack is to preserve the upper and lower bounds of another subinterval (new left and right subintervals may be generated during sorting) when one subinterval is processed, and then remove the boundary of another subinterval from the stack to process it. This functionality is also possible with queues, except that the order of processing subintervals has changed.

Tips：Use two queues to simulate the behavior of the stack to achieve non-recursive quick sorting. Although it is not efficient, it tests the ability to be flexible.

6. Please summarize what a hash collision is (in 1-2 sentences). Then list two common methods for resolving hash collisions (summarize them briefly) and compare their advantages and disadvantages.

A hash collision occurs when different inputs produce the same hash value.

Two common methods:separate chaining and open addressing.

- Separate chaining uses linked lists at each hash table slot to handle collisions, while open addressing finds the next available slot in the table for colliding elements.
- Separate chaining is simple and easy to implement but requires extra memory for lists. Open addressing saves space but can slow down insertion and search times, especially when the hash table is nearly full.

7. The Fibonacci sequence is defined as follows: $F(0) = 0, F(1) = 1$, For $n \geq 2, F(n) = F(n-1) + F(n-2)$

Given a non-negative integer n, please design a recursive function to calculate the n_{th} term of the Fibonacci sequence (using memoization).
You can use Python, C++, or C to implement the solution.

```python
def fibonacci(n, memo={}):
    if n in memo:
        return memo[n]
    if n == 0:
        return 0
    if n == 1:
        return 1
    memo[n] = fibonacci(n - 1, memo) + fibonacci(n - 2, memo)
    return memo[n]
```