Course: Data Structures (CSE CS203A, 114-1)
Midterm
October 28, 2025, 15:10~17:00

Student ID: Student Name:

Q1: (14 pts, 2pts each) **Comprehensive Multiple-Choice**

1. In a linked list-based selection sort, compared to an array version, which operation becomes more expensive?

A. Swapping two values B. Traversing to find the minimum node
C. Comparison of node values **D. Reassigning pointers**

2. If we apply Binary Search on a linked list, its time complexity becomes:

A. O(n) B. O(log n) C. O(1) D. $O(n^2)$

3. At each step, Binary Search divides the search range by:

A. 1 **B. 2** C. log n D. n

4. Which of the following describes the order of a stack?

A. FIFO **B. LIFO** C. Random Access D. Circular

5. If the array length doubles (`n → 2n`), the runtime of Bubble Sort roughly becomes:

A. 2× longer **B. 4× longer** C. 8× longer D. Unchanged

6. Which statement best describes “swapping by value” vs “swapping by pointer”?

A. Value swap changes links; pointer swap changes data.
B. Pointer swap modifies structure; value swap changes node contents.
C. Both are identical.
D. Value swap is faster in all cases.

7. In a singly linked list, to insert a node after a specific target node, we must modify:

A. Only the target node’s `next` pointer.
B. Both the target and the next node’s `next`.
C. All node pointers from head to tail.
D. None, since linked lists are automatic.

Q2: (20 pts, 5 pts each) **Time complexity**

In class, we implemented **Bubble Sort** and **Selection Sort** using arrays.
To analyze their time complexity, we count the number of **basic operations** as the array size (n) increases.
The time complexity of Bubble Sort in the worst case (array completely reversed) is **$O(n^2)$**.
The time complexity of Selection Sort in the best case (array already sorted) is **$O(n^2)$**.
The worst-case time complexity of binary search is **O(logn)**.

If the array length is n, the maximum number of comparisons in binary search is approximately **$\lfloor \log_2(n) \rfloor + 1$** (Note: consider the floor or ceiling of $\log_2(n)$ as appropriate.).

Q3: (36 pts, 3 pts each) **Selection Sort - Array Implementation**

Task: Represent data using an array and apply the **selection sort** algorithm step by step. During the process, the array is divided into two parts: *sorted* and *unsorted*.

Given Input integers: 24, 9, 100, 2, 5.

Pseudocode:

```
SelectionSortArray(A[1..n]):
    for i from 1 to n-1:
        minIndex = i
        for j from i+1 to n:
            if A[j] < A[minIndex]:
                minIndex = j
        swap A[i] and A[minIndex]
```

Step Trace Table (Array):

Step 1 is completed as an example. Complete the remaining steps.

Subarray scanned is the unsorted part.

Step i	Subarray scanned (A[i..n])	minIndex found (value)	Swap A[i] ↔ A[minIndex]?	Array after this step
1	[24, 9, 100, 2, 5]	4 (2)	Yes	[2, 9, 100, 24, 5]
2	① **[9, 100, 24, 5]**	② **5 (5)**	③ **Yes**	④ **[2, 5, 100, 24, 9]**
3	⑤ **[100, 24, 9]**	⑥ **5 (9)**	⑦ **Yes**	⑧ **[2, 5, 9, 24, 100]**
4	⑨ **[24, 100]**	⑩ **4 (24)**	⑪ **No (self-swap)**	⑫ **[2, 5, 9, 24, 100]**

Q4: (30 pts, 5pts each) **Real-World Applications and Data Structures**

Match each real-world application (Column A) to the most suitable data structure (Column B). Each application should match one best answer. Provide a short explanation (1–2 sentences) describing *why* that structure fits. (Hints: Some real-world applications may combine multiple data structure.)

Column A — Applications

No.	Real-World Application
1	Managing the browser back/forward history
2	Serving customers in a line at a ticket counter
3	Representing a playlist of songs that can be dynamically added or removed
4	Implementing undo/redo functionality in a text editor
5	Handling task scheduling in an operating system
6	Implementing the binary search algorithm for integers

Column B — Data Structures

Letter	Data Structure	Letter	Data Structure	Letter	Data Structure
A	Array	B	Linked List	C	Stack
D	Queue	E	Hash Table	F	Graph
G	Tree				

No.	Application	Correct Data Structure	Explanation
1	Browser back/forward history	**Stack**	**LIFO: last page visited → first undone.**
2	Ticket counter line	**Queue**	**FIFO: first customer served first.**
3	Playlist of songs (dynamic)	**Linked List**	**Easy insertion/deletion without shifting memory.**
4	Undo/redo in text editor	**Two Stacks**	**One stack for undo, one for redo.**
5	Task scheduling in OS	**Queue**	**Jobs handled in order received.**
6	Binary search on integers	**Array**	**Requires indexed, sorted data.**

Q5: (20 pts) Coding Questions

A. Linked List - Swap two nodes by pointer (10 pts, 5 pts each)

Implement a function that swaps two nodes in a singly linked list by pointer manipulation, not by swapping values. (Hints: Visualization can help you better understanding)

```
void swapNodes(Node** head, int x, int y) {
    if (x == y) return;

    Node dummy = {0, *head};
    Node *prevX = &dummy, *prevY = &dummy;
    Node *a = NULL, *b = NULL;

    for (Node* p = &dummy; p->next; p = p->next) {
        if (p->next->val == x) { prevX = p; a = p->next; }
        if (p->next->val == y) { prevY = p; b = p->next; }
    }
    if (!a || !b) return;

    if (a->next == b) { // adjacent a before b
        // ① insert your code
        a->next = b->next;
        b->next = a;
        prevX->next = b;
    } else if (b->next == a) { // adjacent b before a
```

```
            // ② insert your code here
            b->next = a->next;
            a->next = b;
            prevY->next = a;
        } else {
            Node* tmp = a->next;
            a->next = b->next;
            b->next = tmp;
            prevX->next = b;
            prevY->next = a;
        }

        *head = dummy.next;
}
```

B. Stack Overflow Check (5 pts)

Fill in the missing condition for stack overflow in an array-based stack implementation:

```
#define MAX 5
int stack[MAX];
int top = -1;

void push(int value) {
        if (top == MAX - 1 ) {
            printf("Stack Overflow!\n");
            return;
        }
        stack[++top] = value;
}
```

C. Queue Enqueue (5 pts)

Implement enqueue for a circular queue.

```
#define MAX 5
int queue[MAX];
int front = -1, rear = -1;

void enqueue(int item) {
        if ( (rear + 1) % MAX == front ) {
            printf("Queue Overflow!\n");
            return;
        }
        if (front == -1) front = 0;
        rear = (rear + 1) % MAX;
```

```
        queue[rear] = item;
}
```

Q6: (20 pts, 5 pts each) **Bonus – Hashing Concepts**

Definitions

- Hash Function - A hash function is a function that converts a key (such as a number or string) into an integer index. This index is then used to determine where the data will be stored in a hash table.
- Hash Table - A hash table is a data structure that stores key–value pairs. It uses a hash function to compute an index in a data structure (called a bucket) where the data will be placed.
- Collision Handling - A collision occurs when two or more keys are assigned to the same index by the hash function. Collision handling refers to the methods used to store and retrieve these multiple items that share the same index.

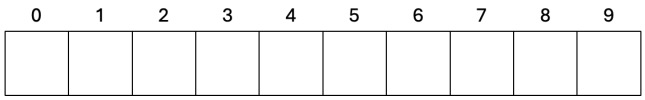
Task: Based on the definitions above and the data structures you have learned so far (array, linked list, stack, and queue), explain which data structures can be used to implement each of the following concepts and why. A visualization is also required.

Concept	Data Structure(s) Used	Explanation
Hash Function	Not a data structure	Produces an integer index used to select a bucket: index = h(key) % m.
Hash Table	**① Array**	**② Stores buckets indexed by hash value.**
Collision Handling	**③ Linked List**	**④Manages multiple keys per bucket (separate chaining).**

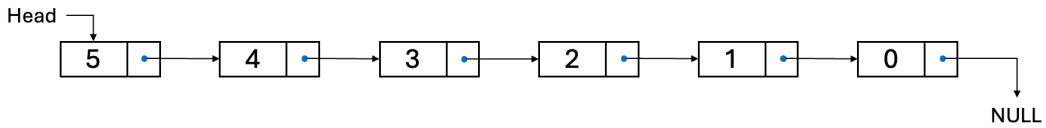
Supplementary

Data Structures: Visualization

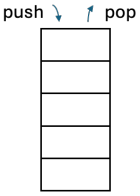
(1) Array



(2) Linked List



(3) Stack



(4) Queue