訂正

Course: Data Structures (CSE CS203A, 114-1)

Quiz I: Introduction to C Programming and Data Structures

September 30, 2025, 16:30~17:00

Student ID: 1131527 Student Name: 周品岑

65 +20

Q1: (20 pts; 5 pts for each) **Complete the C Code**

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    ____int①____ *array;
    int n = 10;

    // Allocate memory for n integers
    array = (int *) malloc(n * __sizeof②(int)__);

    // Initialize array with values 1, 2, 3, ..., 10
    for(int i = 0; i < n; i++) {
        array[i] = i + 1;
    }

    // Print the original array
    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");

    // Double the array size
    n = n * 2;
    array = (int *) __real③oc____ (array, n * sizeof(int));

    // Initialize new elements (second half)
    for (int i = n/2; i < n; i++) {
        array[i] = i + 1;
    }

    // Print the resized array
    printf("Resized array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");

    // Clean up memory
    ____④____ free(array);
    array = NULL;

    return 0;
}
```

A1:

① int

② sizeof(int)

③ realloc

④ free(array);

+20.

Q2: (20 pts) **Memory Management Code Review**

You are conducting a code review for a junior developer who submitted the following C code for a production system that will handle user data processing. The code dynamically allocates memory for an integer array, processes the data, and then expands the array size as needed.

```
double *array;
int n = 10;

array = (double *) malloc(n * sizeof(double));
if (array == NULL){
// ... processing code ...
fprintf(stderr,"Error: Failed to allocate memory for %d doubles\n",n);
return 1;
}
n = n * 2;
array = (double *) realloc(array, n * sizeof(double));
if(temp == NULL){
// ... more processing ...
fprintf(stderr,"Error: Failed to reallocate memory for %d doubles\n",n);
free (array);
free(array);
return1;
}
array = temp;
```

As a senior developer responsible for code quality and system reliability, you notice several critical memory management issues that could lead to:

- Memory leaks
- Segmentation faults
- System crashes in production

- Data corruption
- Undefined behavior

Task: Identify the specific memory management issues and provide solutions to ensure safe memory management.

A2: 少了 #include <stdio.h> 主函式會無法編輯

```
#include <stdlib.h>
int main(){
    :
  return 0;
}
```

1. 缺少 malloc() error 的確認
2. 不安全的 realloc() usage
3. 沒有 error 處理策略

Q3: (40 pts) **Time Complexity Analysis**

Fill in the blanks with the appropriate Big O notation: O(1), O(log n), O(n), O(n log n), $O(n^2)$, $O(n^3)$, O(n!).

Q3-1: (5pts) If binary search is O(log n) and we perform it n times, the overall time complexity is ~~O(n)~~. O(nlogn)

答案為錯位置

```
for(int i = 0; i < n; i++) {
    // Binary search operation on sorted array
    binarySearch(sortedArray, target, n);
}
```

logn

Q3-2: (5 pts)

Accessing an element in an array by index (e.g., array[5]) has a time complexity of O(1).

Q3-3: (15 pts; 5 pts for each)

Finding the maximum value in an unsorted array by checking every element has a time complexity of O(n).

Traversing through all elements in an array of size n has a time complexity of O(n).

Do these two operations have the same time complexity? Yes (Yes/No).

Q3-4: (5 pts)

Bubble sort algorithm for sorting an array of n elements has a time complexity of $O(n^2)$

Q3-5: (10 pts) A: $O(1) < O(logn) < O(n) < O(nlogn) < O(n^2) < O(n^3) < O(n!)$ -5

Order the following Big O notations from fastest (most efficient) to slowest (least efficient):

Given: O(n!), O(1), $O(n^2)$, O(log n), O(n log n), O(n), $O(n^3)$

$logn < 1 < n < nlogn < n! < n^2 < n^3$

答案為這裡

2

A3-1: ~~O(logn)O(1), $O(n^2)$, O(n!), O(nlogn), O(n), $O(n^3)$~~

A3-2: ~~O(logn).O(1)$O(n^2)$. O(n!). O(nlogn). O(n). $O(n^3)$~~

A3-3: ~~O(logn). O(1). O(n). O(n!).O(nlogn). $O(n^2)$. $O(n^3)$~~

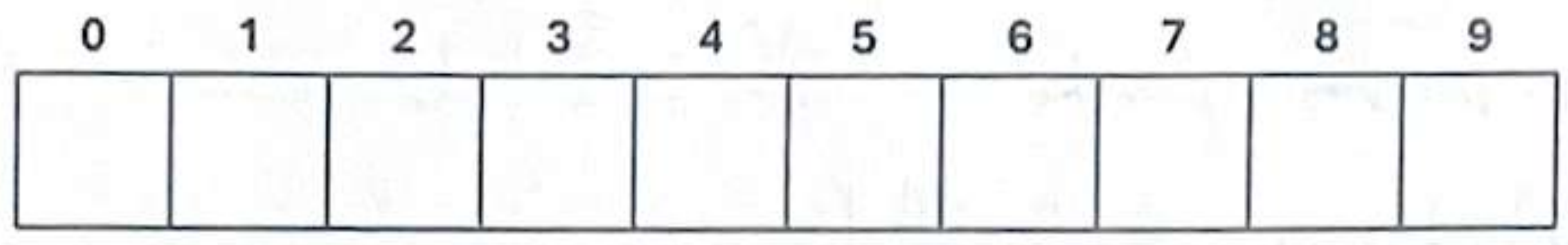
Course: Data Structures (CSE CS203A, 114-1)
Quiz II: Array, Linked List, Stack and Queue
October 21, 2025, 16:30~17:00

60

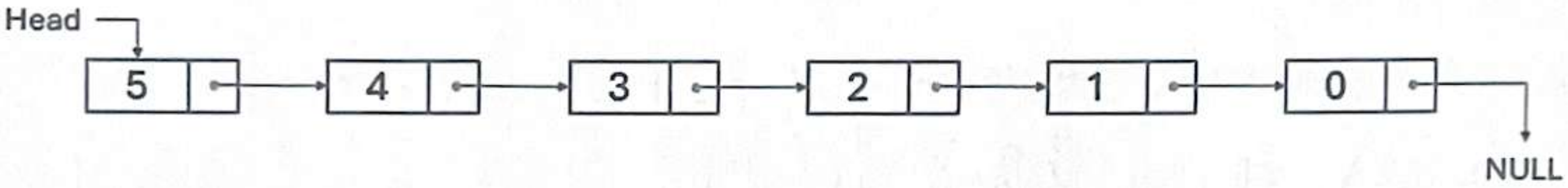
Student ID: 1131527 Student Name: 周品岑

Data Structures: Visualization

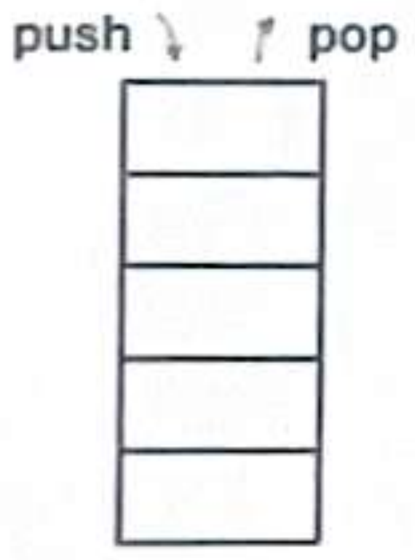
(1) Array



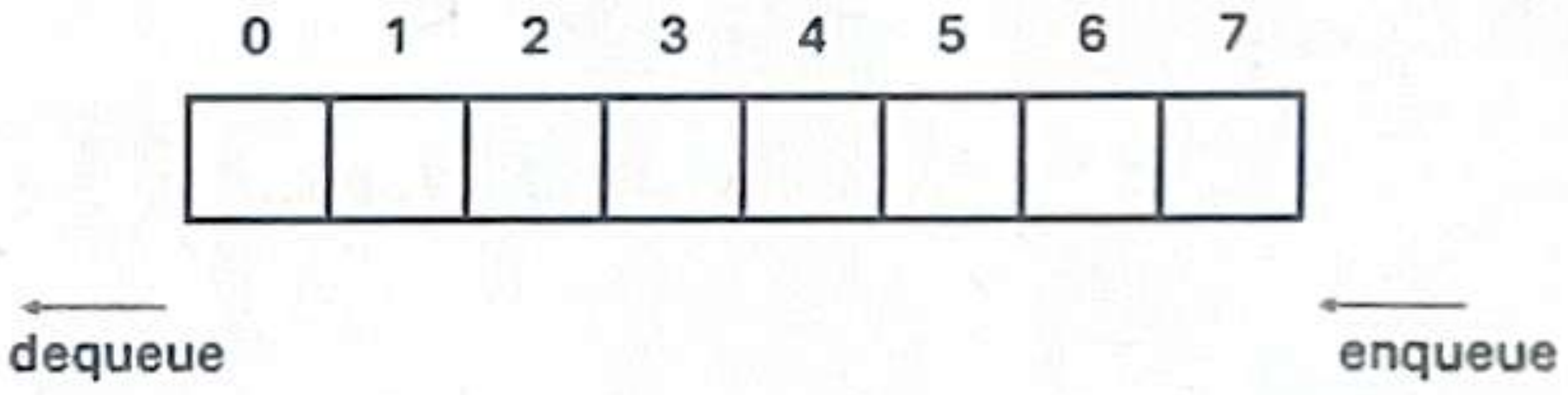
(2) Linked List



(3) Stack



(4) Queue



—20

Q1: (30 pts; 10 pts for each) **Describe the mechanism of the function**

MoveTo(node *head, node *target, node*destination)

A1: Write a short paragraph explaining how the **MoveTo** function works (you may answer in English or Mandarin).

↳ target移到 destination後面

額外參數

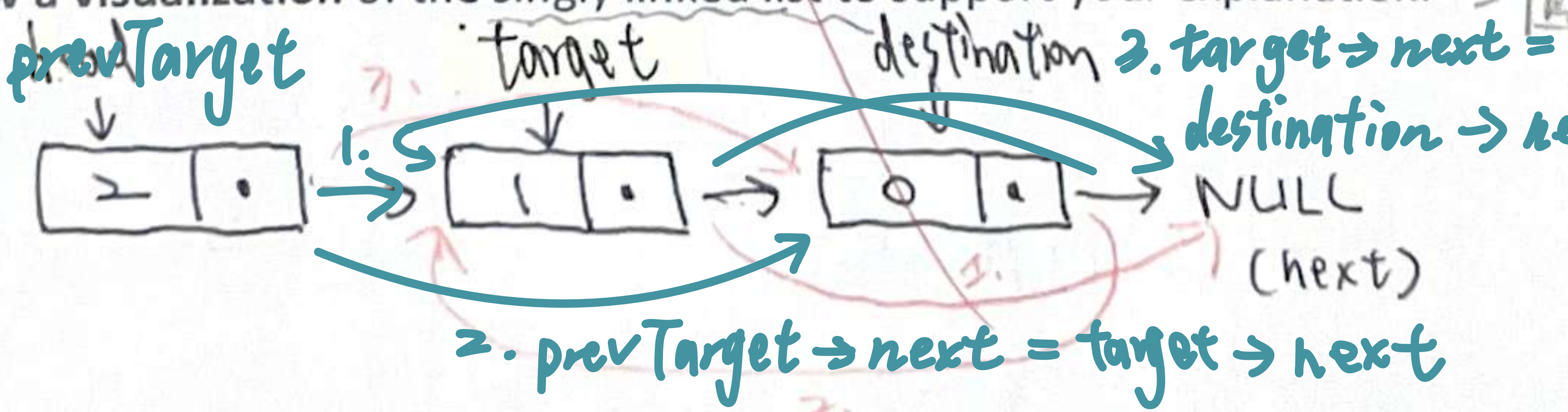
① Are there any **additional variables** required? If so, explain why they are necessary.

~~不需要。只要變換pointer的指向即可~~ 需要，prevTarget & destination

~~將 node*target → NULL (next)、node*destination → target、node*head → destination~~

移動透過 pointer manipulation 而非 data swapping

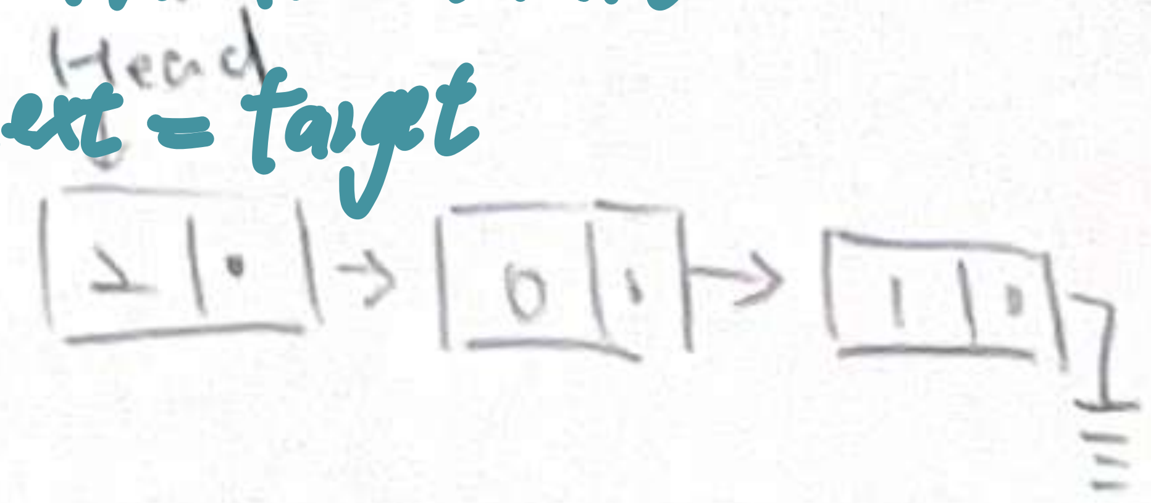
② **Draw a visualization** of the singly linked list to support your explanation. → 圖解



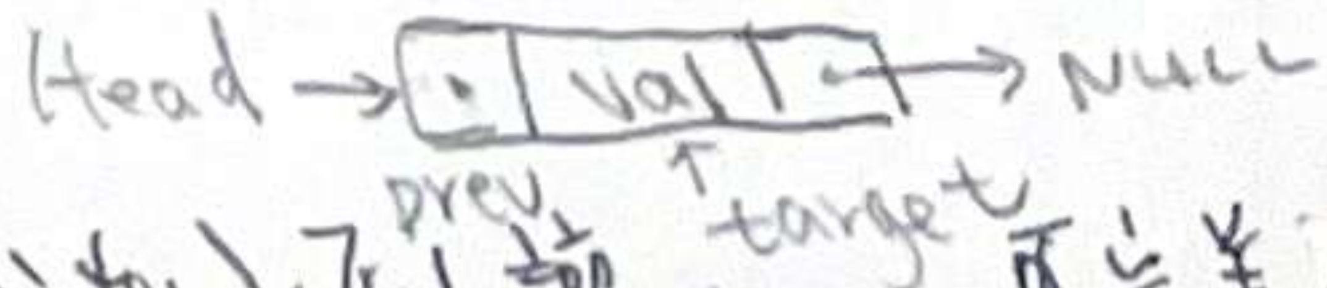
3. target → next = destination → next

destination → next = target

2. prevTarget → next = target → next



③ Is there any **variation of a linked list** (e.g., doubly linked list or circular linked list) that can simplify or improve this operation?



improve but not simplify

doubly linked list 有prev (pointer)，可以讓MoveTo更完善但步驟會變複雜

—10

Q2: (40 pts, 10 pts for each) **Definition of Data Structures**

Define the following data structures and list their fundamental operations.

A2:

① Definition of "Stack"

~~operations：pop、push．peek~~ ．＊LIFO（last-in-first-out）

define：有 top 和 bottom，最先排序進去的元素會放在 bottom 滿至 top，增加/減少都只變動 top 的元素（top>0 ~ top== max_size-1）

② Definition of "Queue"

~~operations：dequeue，enqueue~~ ＊FIFO（first-in-first-out）

define：有 近處 和 rear（遠處），最先排序進去的元素會放在近⇒遠，增加會從 rear 進去，減少會從近處出去

③ Preliminary operations of "Stack"

operations：pop、push．peek．isEmpty（是否為空）．isFull（是否為滿）

（拿出）（放入）（看top的元素）

④ Preliminary operations of "Queues"

delete　addQ

operations：dequeue，enqueue．front（看front的元素）．isEmpty．isFull

（移出）（移入）

Q3: (30 pts) **AI Copilot Application**

Choose **up to two** data structures from the visualization list above. prompt 怎麼給

Compose a **single prompt (within 300 words)** that you would use with an **AI Copilot** to explore or learn advanced concepts related to your chosen data structures.

150

A3: 請先定義 array 和 linked list 是什麼並說明其 ADT 和在 C++ 中將會如何呈現，將不同的動作（如 acess、travesal、delete、insert 等）都列出 Big-O 並解釋（為什麼是 O(1) 而不是 O(n) 等）delete 可延伸出 top、n、end 的時間複雜度差異，若有多種可能性（O(1) 和 O(n) 都可能）也要逐一講解比對。並做表格呈現 data structures 的特色，使用情境，使用差異，好處，flexibilty 和綜合評價，且詳細說明哪種適用於哪個情況較好，可附圖表便利了解。

—10

Course: Data Structures (CSE CS203A, 114-1)
Quiz III: Hash Table
November 25, 2025, 16:40~17:00

65

Student ID: 1131527 Student Name: 周品彥

+30

Part A: Hash Table Definitions (Conceptual Understanding)

Q1. Define "collision" in the context of hash tables.

A1: different keys have the same index. (hash address)

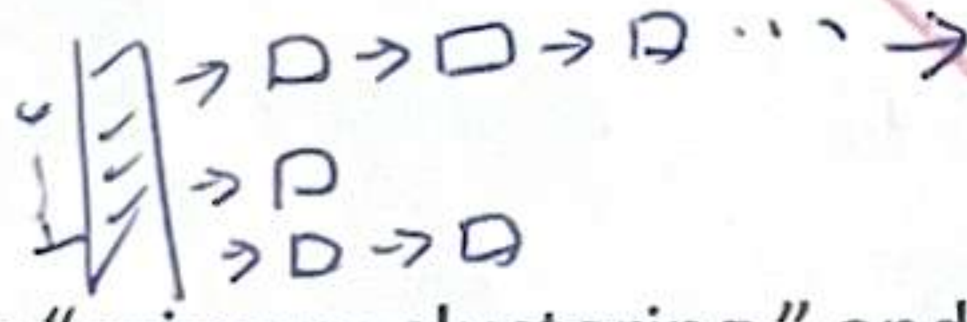
Q2. What is a "bucket" in a hash table?

A2: ~~like index，在經過 hash function 後用於做索引分門別類的索引合集~~

eg. →有一或多個 records at a table index (set of indexs)

Q3. Define "load factor (α)" and explain why it affects performance.

A3: like linked list，一個格子一個 loading，若數量多會降效率 α↑

eg.  → 降效率

$\alpha = \frac{n = \text{number of elements}}{m = \text{table size}}$

⇒ more collisions

⇒ longer probe chains

Q4. What is "primary clustering," and which probing method suffers from it?

A4: happened in linear probing，由於 index 不斷地+1 儲存下去而導致 cluster 的數量過多降效率.

consecutive

Q5. What is "secondary clustering," and how is it different from primary clustering?

A5: happened in quadratic probing，會重複產生不同 keys 位於相同 index 的情況而要做進一步的 hash function 去解決。相比 primary clustering 會使 cluster 的數量變少，index 跨的幅度較大

↓ 不連續但相同情況

Q6. Briefly explain the difference between:

- Open addressing
- Separate chaining

A6: Open addressing：透過 probing 的方式無限擴大 bucket eg.

separate chaining：透過 linked list 的方式 chaining value

eg. □→□→□→…

Part B: Hash Function Calculation (Collision & Pattern Observation)

Show your steps clearly.

Hash Function 1 — Division Method

$$h_1(k) = k mod 10$$

Hash Function 2 — Folding Method

Split key into two-digit chunks and sum the chunks.

$$h_2(k) = (sum\ of\ 2 - digit\ groups) mod \underline{\underline{11}}$$ ★ 看錯ㄌ

Example:

Key = 8429 → groups: 84 + 29 → 113 → 113 mod 11 = 3

Q7. (Compute using Hash Function 1)

Given keys: 27, 37, 47, 57, 67

Compute their hash values using:

$$h_1(k) = k mod 10$$

A7:

key	i	$h_1(k)$
27	0	27 mod 10 = 7
37	1	37 mod 10 = 7
47	2	47 mod 10 = 7
57	3	57 mod 10 = 7
67	4	67 mod 10 = 7

移上題

index[7] = 27
index[7] = 27 → 37
index[7] = 27 → 37 → 47
index[7] = 27 → 37 → 47 → 57
index[7] = 27 → 37 → 47 → 57 → 67 #

collision
↓

index	linear probing
7+0	index[7] = 27
7+1	index[8] = 37
7+2	index[9] = 47
7+3	index[0] = 57
7+4	index[1] = 67.

→ (超過10-1=9.)

Q8. (Identify collision pattern)

From your results in Q1:

- What pattern do you observe?
- Explain why these keys collide.

A8: key ⇒ hash function ⇒ index ⇒ value. 不同的key但有相同的index → 產生collision

bucket 空間不夠 (只有10格) 且 function 方式簡單 (只有一個步驟即 mod 10)

↓

value 可能值過多 → 但不夠放 ⇒ 不同value 擠在一起

Q9. (Compute using Hash Function 2)

Compute $h_2(k)$for: 1234, 9217, 4519, 9902

A9:

key	i	groups	$h_2(k)$	
1234	0	12+34=46	46 mod 11 = ~~6~~ 2	index[2] = 1234
9217	1	92+17=109	109 mod 11 = ~~9~~ 10	index[10] = 9217
4519	2	45+19=64	64 mod 11 = ~~4~~ 9	index[9] = 4519
9902	3	99+02=101	101 mod 11 = ~~1~~ 2	index[2] = 9902 #

Q10. (Compare distribution)

- Which hash function (h₁ or h₂) produced more collisions for the input set?
- Which seems to spread keys more evenly?
- Provide 1–2 sentences of explanation.

A10: (1) h1 (2) h1 ~~使用linear probing 時 [illegible] 每個index都填到~~ 發生較多collisions eg. 都在 index[7]

h2 ~~會跳著放入不同index 較無順序~~ 能更分散地儲存values eg. 在index[2].[9].[10]

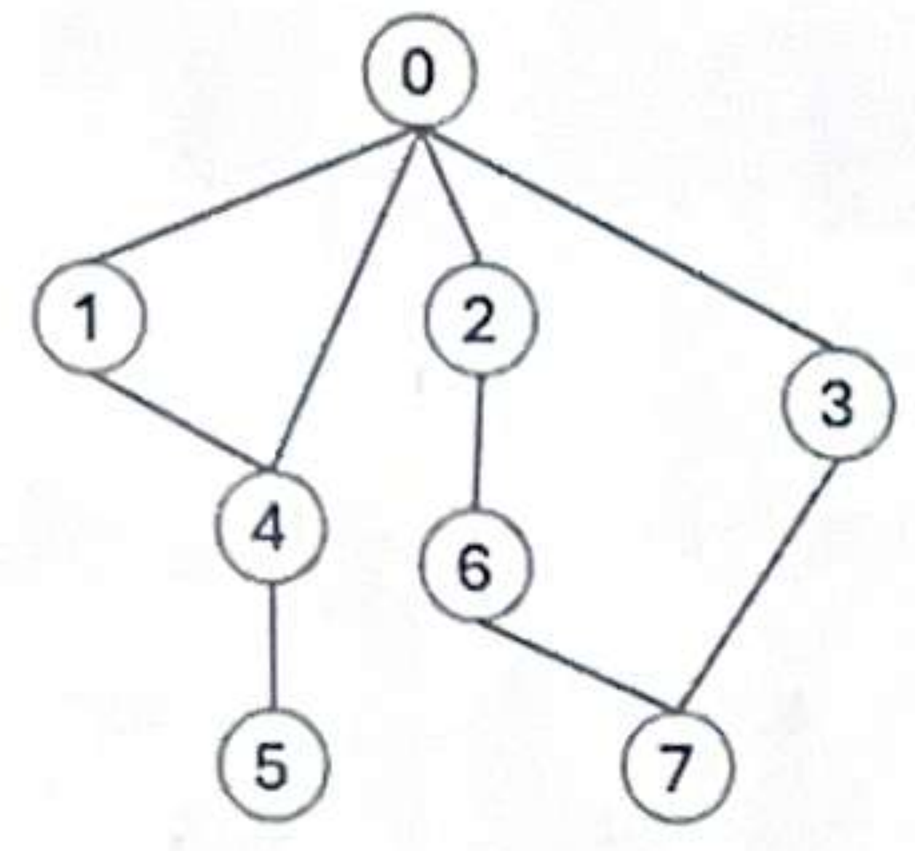
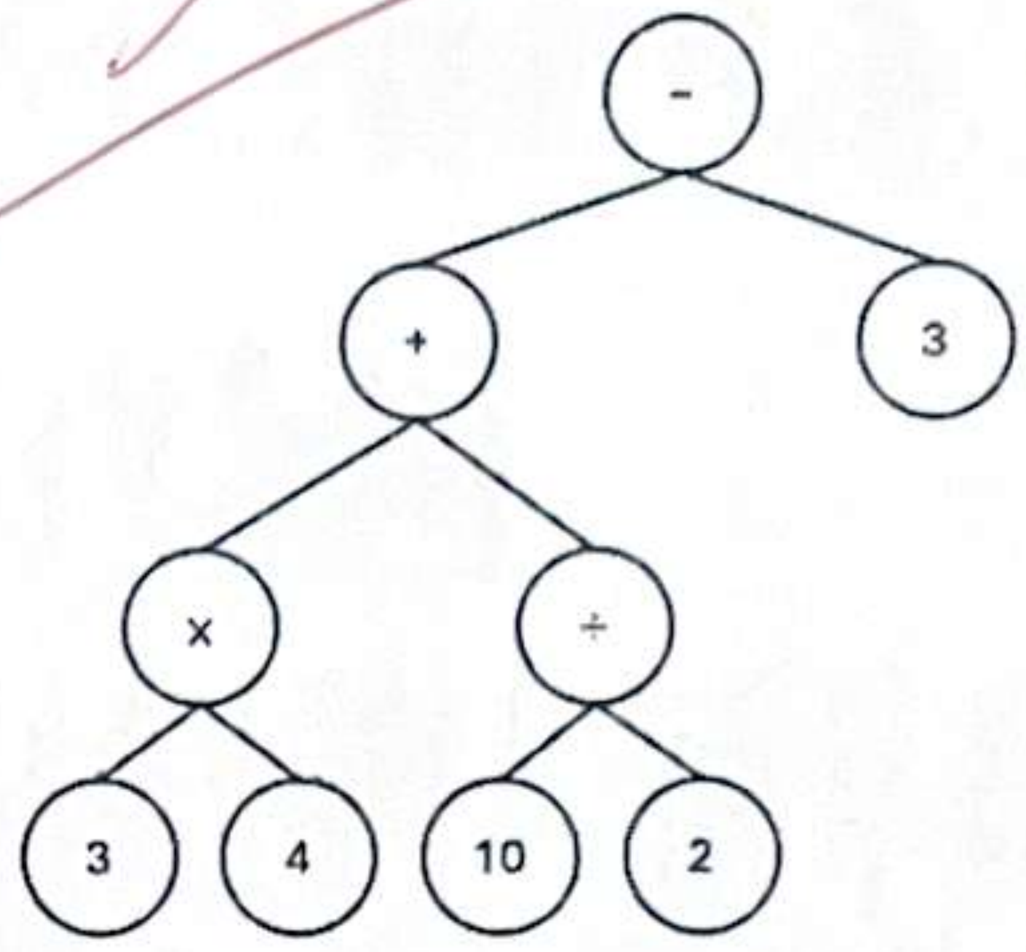
Course: Data Structures (CSE CS203A, 114-1)
Take-Home Quiz IV: Tree/Heap/Graph

94

Due: December 16, 2025, 17:00 (Room R1102)

Important Notice: You must print this take-home quiz sand **write your answers by hand with a pen.**

Student ID: 1131527　　　Student Name: 周品彥

Q1 Figure	Q2 Figure
	

Q1. (30 pts) Explain Breadth-First Search (BFS) on the graph and provide the BFS traversal order for the graph shown in Q1 Figure. 1.用 queue 2. 3. level by level

A1:
(1) BFS即廣度優先搜尋，從根節點(root)開始一層一層向下搜尋，先從左到右走完相鄰的節點，再往下一層繼續。 5.確保無 revisiting

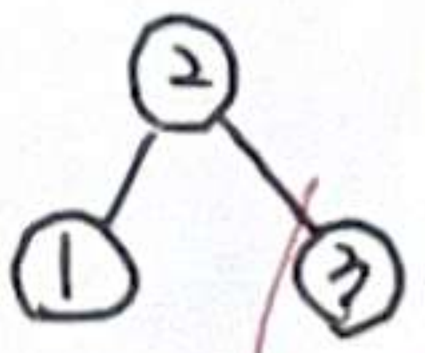
(2) 0 → 1 → 2 → 3 → 4 → 6 → 7 → 5
(root) (一) (一) (一) (二) (二) (三) (三)

queue　revisting

→ Traversal order: 0, [任意順序 1, 2, 3, 4], [2→6, 3→7, 4→5]

16　(有 1×4! = 24 種組合)

Q2. (30 pts) In tree traversal, one common method is inorder traversal. Please use inorder traversal to print the arithmetic expression represented by the expression tree in Q2 Figure, and then evaluate it to compute the final result.

A2:
(1) inorder traversal即中序遍歷，從左子節點開始，再拜訪父節點，最後拜訪右子節點  ①左子節點 ②父節點 ③右子節點

⇒ 3 → × → 4 → + → 10 → ÷ → 2 → − → 3 #

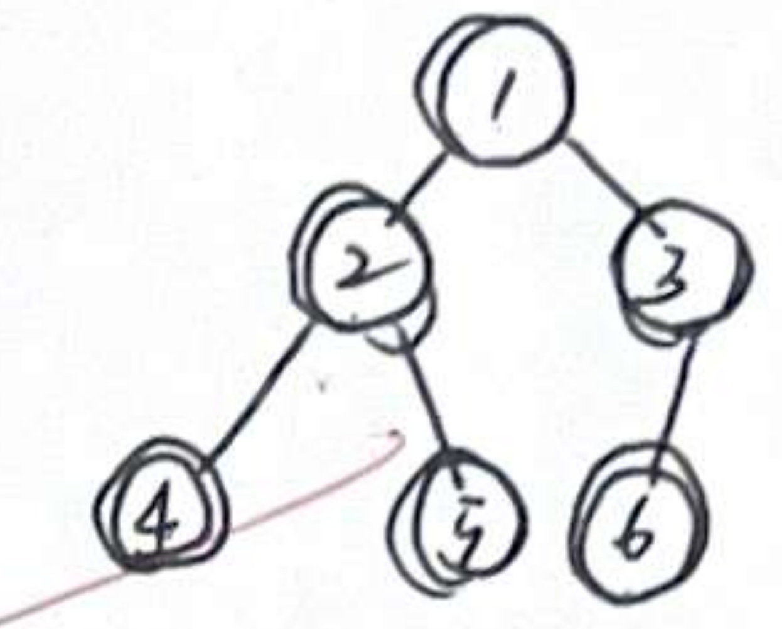
(2) $(3\times4)+(10\div2)-3 = 12+5-3$
$= 14$ #

Q3. (40 pts) A binary tree is a fascinating data structure with many variations, including binary search trees, AVL trees, red-black trees, complete binary trees, and max/min heaps. These variations can be classified as shape-based (structural constraints) or criteria-based (rules such as ordering). Choose one shape-based tree and one criteria-based tree, and provide a brief description of each.

A3:

eg. Full Binary Tree. Perfect Binary Tree

(1) shape-based: complete binary tree 完全二元樹



特性: ① 除了最後一層外, 每一層都要被填滿.

② 最後一層的節點則須由左到右依序排列填滿, 中間不能有空缺

eg. AVL Tree. Red-Black Tree. Heap (Max/Min)

(2) criteria-based: binary search tree 二元搜尋樹

特性: ① 若任意節點的左子樹不空, 則左子樹上所有節點 < 根節點

② 若任意節點的右子樹不空, 則右子樹上所有節點 > 根節點

③ 任意節點的左、右子樹也可以是 binary search tree

④ 沒有值相等的節點

⇒ 左子樹 < 父節點, 右子樹 > 父節點