Student ID:

Student Name:

Course: Data Structures (CSE CS203A)

Assignment III: Linked List Selection Sort

Student Worksheet Companion

A1. Linked List Representation Drawing (5 pts)

a. (2 pts) Instructions: Draw a visual representation of a single node with next pointer that contains the initialized integer 10

[10 | •] →

b. (3 pts) Linked list representation with the given integers (Hint: For safety and clarity, include identifiable head and tail nodes)

Example: the input integers are (10, 20) and linked list representation will be [10 | •] → [20 | •] →

head → [60 | •] → [24 | •] → [15 | •] → [42 | •] → [20 | •] → [11 | •] → [90 | •] → [8 | NULL

↑
tail

A2. Populate with Integers (32 pts; 2 pts for each)

1

Fill the given integers (60, 24, 15, 42, 20, 11, 90, 8) into the above structures.

Annotate:

Node #	Value	Next Pointer
1	[60]	→ Node [2]
2	[24]	→ Node [3]
3	[15]	→ Node [4]
4	[42]	→ Node [5]
5	[20]	→ Node [6]
6	[11]	→ Node [7]
7	[90]	→ Node [8]

Student ID:	Student Name:
8 [8]	→ [NULL]

A3. Selection Sort – First Three Steps (45 pts; 15 pts for each step)

Step Trace Table (Linked list):

Step 1 is the example to help you to complete step 2 to 4.

Step 1 (i = head = 60): Traverse list to find minimum value 8 → call swap function Yes; swap (60, 8).

head → [8|•] → [24|•] → [15|•] → [42|•] → [20|•] → [11|•] → [90|•] → [60|NULL]

Step 2 (i = 24): Minimum value [11] → call swap function [✓ Yes] / No; swap ([24], [11]).

head → [8|•] → [11 |•] → [15 |•] → [42 |•] → [20 |•] → [24 |•] → [90 |•] → [60 |NULL]

Step 3 (i = 15): Minimum value [15] → call swap function Yes / [✓ No]; swap ([15], [15]).

head → [8|•] → [11 |•] → [15 |•] → [42 |•] → [20 |•] → [24 |•] → [90 |•] → [60 |NULL]

Step 4 (i = 42): Minimum value [20] → call swap function [✓ Yes] / No; swap ([42], [20]).

head → [8|•] → [11 |•] → [15 |•] → [20 |•] → [42 |•] → [24 |•] → [90 |•] → [60 |NULL]

Student ID: Student Name:

(1)	O(1)	(2)	O(n)
(3)	O(n)	(4)	O(n)
(5)	O(1)	(6)	O(1)
(7)	O(1)	(8)	O(1) ~~or O(n)~~
(9)	O(n²) ✓	(10)	O(n²)
(11)	O(1)	(12)	O(1)
(13)	Low	(14)	Moderate

1

Student ID: Student Name:

Characteristics (54 pts, 3 pts for each)

Aspect	Array	Linked List
Storage	(1)	(2)
Access	(3)	(4)
Extra Variables	(5)	(6)
Traversal	(7)	(8)
Overhead	(9)	(10)
Visualization	(11)	(12)
Swaps	(13)	(14)
Flexibility	(15)	(16)
Overall	(17)	(18)

(1)

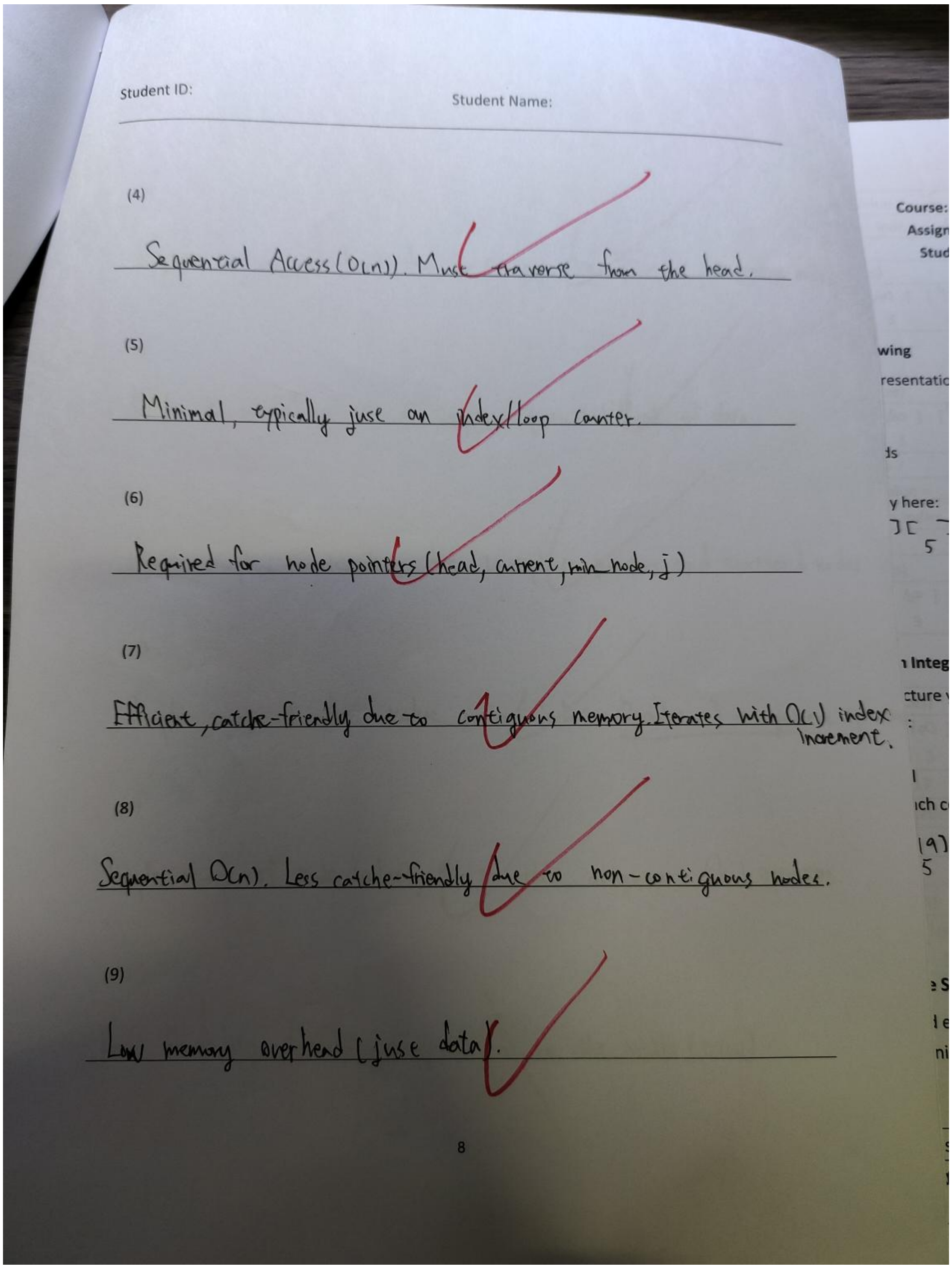
Contiguous memory block. Fixed size.

(2)

Non-Contiguous memory blocks (nodes). Dynamic size.

(3)

Random Access (O(1)). Direct Indexing



(4)

Sequential Access (O(n)). Must traverse from the head. ✓

(5)

Minimal, typically just an index/loop counter. ✓

(6)

Required for node pointers (head, current, min_node, j) ✓

(7)

Efficient, catche-friendly due to contiguous memory. Iterates with O(1) index increment. ✓

(8)

Sequential O(n). Less catche-friendly due to non-contiguous nodes. ✓

(9)

Low memory overhead (just data). ✓

Student ID: Student Name:

(10)

Higher memory overhead due to storage for pointers in every nodes.

(11)

Easier to visualize as a simple, sequential block of data.

(12)

More complex to visualize due to explicit pointers and scattered nodes.

(13)

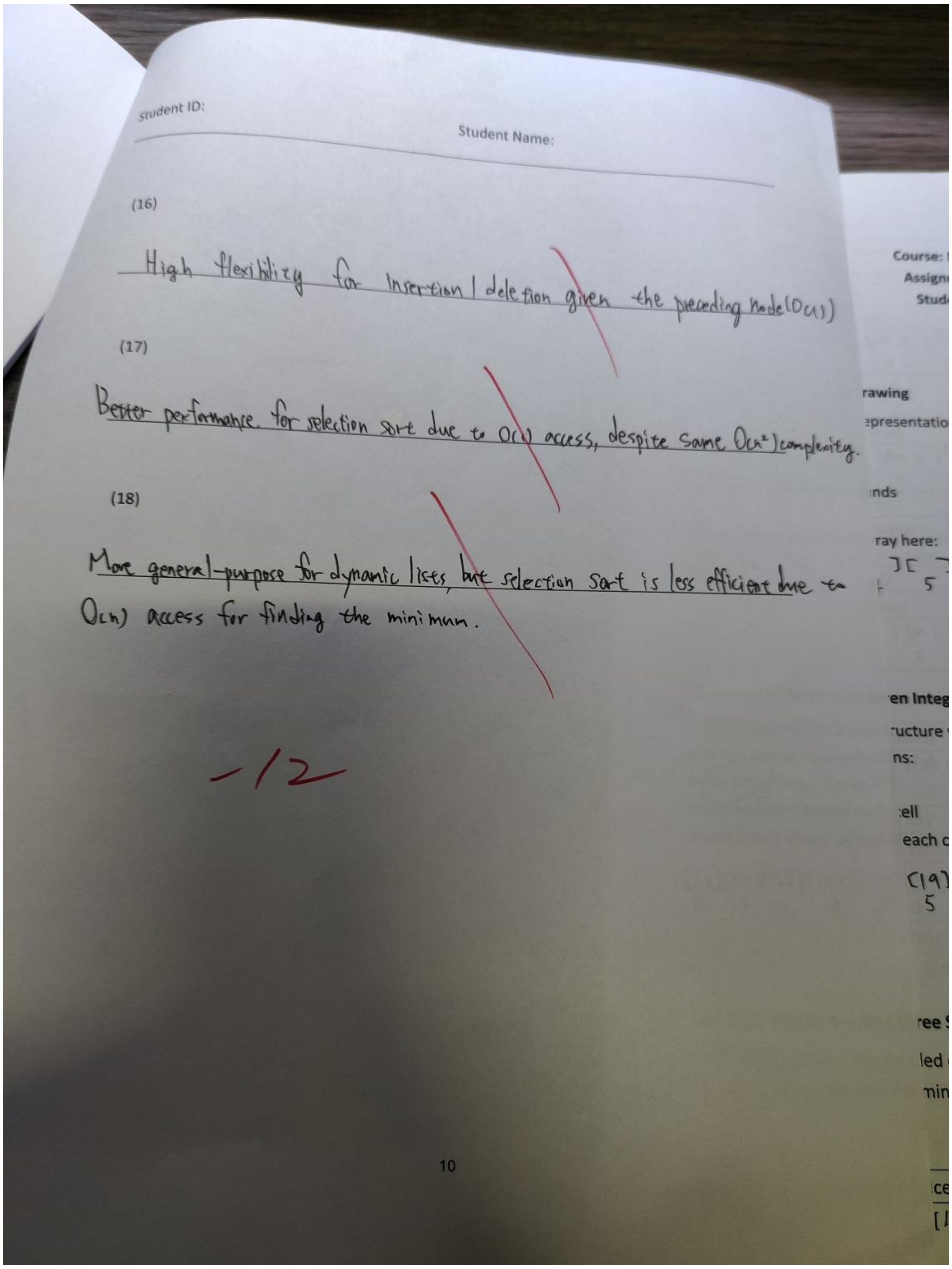
Simple swap of data value using indices. Require 3 assignment operations.

(14)

Simple swap of node values, as per this assignment. O(1) operation.

(15)

Poor flexibility for insertion/deletion in the middle (O(n)).



Student ID:

Student Name:

(16)

High flexibility for insertion / deletion given the preceding node ($O(1)$)

(17)

Better performance for selection sort due to $O(1)$ access, despite same $O(n^2)$ complexity.

(18)

More general-purpose for dynamic lists, but selection sort is less efficient due to $O(n)$ access for finding the minimum.

-1/2