

Student ID: 1133322

Student Name: 林晉遠

Course: Data Structures (CSE CS203A)

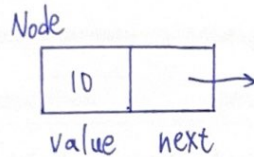
Assignment III: Linked List Selection Sort

Student Worksheet Companion

127

A1. Linked List Representation Drawing (5 pts)

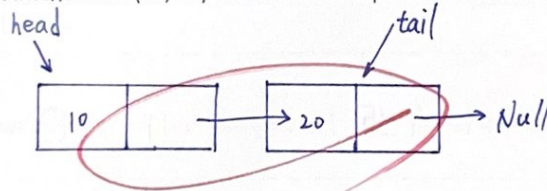
- a. (2 pts) Instructions: Draw a visual representation of a single node with next pointer that contains the initialized integer 10



- 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 |

•] →



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

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: 1133322

Student Name: 林晉廷

Course: Data Structures (CSE CS203A)

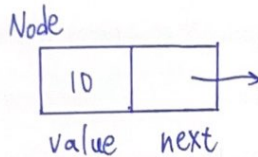
Assignment III: Linked List Selection Sort

Student Worksheet Companion

127

A1. Linked List Representation Drawing (5 pts)

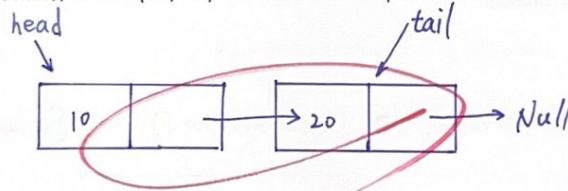
- a. (2 pts) Instructions: Draw a visual representation of a single node with next pointer that contains the initialized integer 10



- 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 |

•] →



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

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 |•] → [11 |•] → [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:

A4. Discussion (68 pts)

Guiding Questions:

- How many swaps/exchanges are performed?
- How expensive is traversal for arrays vs. linked lists?
- What memory/overhead differences do you see?
- Which representation is easier to visualize?
- Which would you choose for implementing selection sort and why?

Time complexity comparison (14 pts, 1pt for each)

Aspect / Operation	Array	Linked List	Explanation
Access Element	(1)	(2)	Array allows direct indexing; linked list needs traversal.
Find Minimum	(3)	(4)	Both must scan all remaining elements/nodes.
Swap Operation	(5)	(6)	In array, swap by indices; in linked list, swap node values.
Traversal Between Elements	(7)	(8)	Linked list traversal requires pointer navigation.
Overall Time Complexity (Selection Sort)	(9)	(10)	Both involve nested traversal to find minima; linked list adds traversal overhead.
Space Complexity	(11)	(12)	Both sorts are in-place if swapping values, not nodes.
Implementation Overhead	(13) Low or Moderate	(14) Low or Moderate	Linked list needs pointer operations and careful null checks.

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)	$O(n)$
(9)	$O(n^2)$	(10)	$O(n^2)$
(11)	$O(1)$	(12)	$O(1)$
(13)	Low	(14)	Moderate

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)

使用連續的記憶體儲存

(2)

使用非連續的記憶體儲存, 每個結點動態分配

(3)

可直接用 index 存取 $\rightarrow O(1)$

Student ID:

Student Name:

(4)

需遍歷節點 $\rightarrow O(n)$

(5)

沒有 extra variables

(6)

每個節點有指標變數

(7)

可用 index Traversal

(8)

需從 Head 逐一沿指標前進

(9)

低

Student ID:

Student Name:

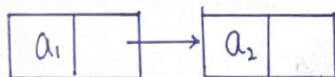
(10)

高, 每個節點需額外管理 next 指標

(11)

$[a_1, a_2, a_3, \dots]$ (線性排列)

(12)



(13)

透過索引交換值

(14)

交換節點的值並更新指標

(15)

大小固定

Student ID:

Student Name:

(16)

可以動態增加新節點

(17)

存取速度快，實作容易

(18)

靈活度高但存取速度慢，實作複雜

-20