

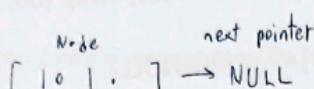
Course: Data Structures (CSE CS203A)

## Assignment III: Linked List Selection Sort

## Student Worksheet Companion

**A1. Linked List Representation Drawing (5 pts)**

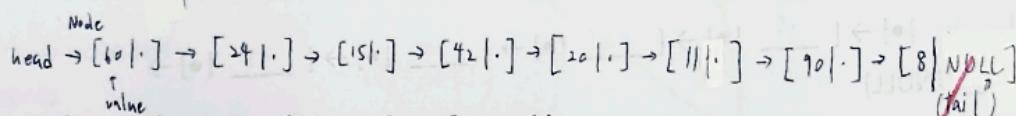
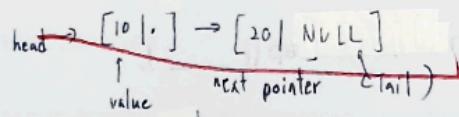
- a. (2 pts) Instructions: Draw a visual representation of a single node with next pointer that contains the initialized integer 10



- 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  $[10 | \bullet] \rightarrow [20 | \cdot]$

$\bullet | \cdot \rightarrow$

**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	<u>60</u>	$\rightarrow \text{Node } [2]$
2	<u>24</u>	$\rightarrow \text{Node } [3]$
3	<u>15</u>	$\rightarrow \text{Node } [4]$
4	<u>42</u>	$\rightarrow \text{Node } [5]$
5	<u>20</u>	$\rightarrow \text{Node } [6]$
6	<u>11</u>	$\rightarrow \text{Node } [7]$
7	<u>90</u>	$\rightarrow \text{Node } [8]$

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 ( $i = \text{head} = 60$ ): 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 ([ \_\_\_\_ ], [ \_\_\_\_ ]).  
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]

Unit 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) $\mathcal{O}(1)$	(2) $\mathcal{O}(n)$	Array allows direct indexing; linked list needs traversal.
Find Minimum	(3) $\mathcal{O}(n)$	(4) $\mathcal{O}(n)$	Both must scan all remaining elements/nodes.
Swap Operation	(5) $\mathcal{O}(1)$	(6) $\mathcal{O}(1)$	In array, swap by indices; in linked list, swap node values.
Traversal Between Elements	(7) $\mathcal{O}(1)$	(8) $\mathcal{O}(n)$	Linked list traversal requires pointer navigation.
Overall Time Complexity (Selection Sort)	(9) $\mathcal{O}(n^2)$	(10) $\mathcal{O}(n^2)$	Both involve nested traversal to find minima; linked list adds traversal overhead.
Space Complexity	(11) $\mathcal{O}(1)$	(12) $\mathcal{O}(1)$	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

## Characteristics (54 pts, 3 pts for each)

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

(1) 記憶體空間會連續，所以有限制，不然就不能用

(2) 分散的記憶體空間，用 pointer 連接在一起，空間利用率較佳

(3) 可以直接用 index 存取，所以  $O(1)$

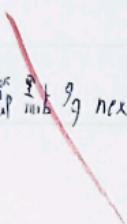
(4) 要從頭跑一遍



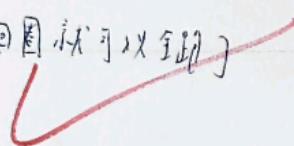
(5) 只要暫存變數，但要先確定空間是否足夠



(6) 利用 pointer，所以 ~~必須~~ next pointer 就行了



(7) 可以用 array [1], array [2] ... 用 for 迴圈就可以全部



(8) 要透過指標一步一步走



(9) 記憶體連續，不用額外 pointer，CPU cache 友善，可使用 index

結構較簡單，但非連續的記憶體，較不用額外資料

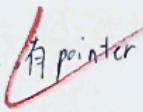
外

(10) 記憶體不連續，要在上一個pointer（有值，有下一個，也就是current），現在的pointer只有next pointer，要有很多資訊才能用  
~~還要注意NULL，邊界的狀況，cache不友善，overhead~~  
整體負擔更重

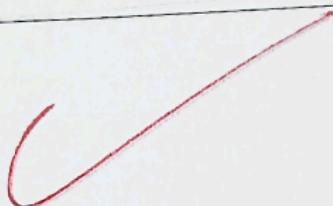
(11) 結構直觀，好理解，好處理



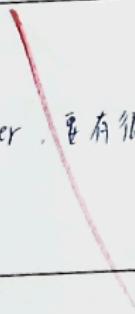
(12) 要畫箭頭指標，較抽象，有值



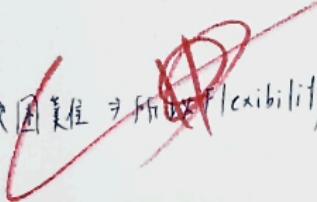
(13) 直接交換值很方便



(14) 有swap by node, swap by pointer，要有很多額外資訊才能進行



(15) 有固定的連續空間，擴充彈性差，flexibility下降



(16)

可以動態的增加，刪除值，因為用pointer，所以很靈活

(17)

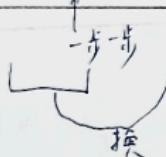
結構簡單，速度因此也較快，可以直接用index

Access / find minimum / swap / traverse  
 $O(1)$        $O(n)$        $O(1)$        $O(n)$   $\Rightarrow O(n^2)$

(18)

彈性高，但 traversal 速度慢（要一步一步來）（要靠 pointer）

Access / find / swap / traverse  
難操作       $O(n)$       一步步（要靠）



- 23