Student ID: 1131428 Student Name: 徐妍雯

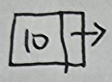
Course: Data Structures (CSE CS203A)

Assignment III: Linked List Selection Sort

Student Worksheet Companion

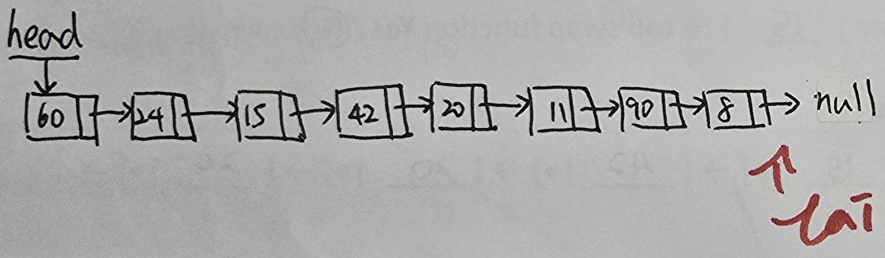
A1. Linked List Representation Drawing (5 pts)

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 | •] → [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 |•] → [24 |•] → [90 |•] → [60 |NULL]

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

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? 4?
- How expensive is traversal for arrays vs. linked lists? On On
- What memory/overhead differences do you see?
- Which representation is easier to visualize? array
- Which would you choose for implementing selection sort and why? array

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

Aspect / Operation	Array	Linked List	Explanation
Access Element	(1) O_1	(2) O_n	Array allows direct indexing; linked list needs traversal.
Find Minimum	(3) O_n	(4) O_{n^2}	Both must scan all remaining elements/nodes.
Swap Operation	(5) O_1	(6) O_n 1	In array, swap by indices; in linked list, swap node values.
Traversal Between Elements	(7) O_n	(8) O_n	Linked list traversal requires pointer navigation.
Overall Time Complexity (Selection Sort)	(9) ~~O_n~~ O_{n^2}	(10) O_{n^2}	Both involve nested traversal to find minima; linked list adds traversal overhead.
Space Complexity	(11) ~~O_n~~ O_1	(12) ~~O_n~~ 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 Name:

Student ID:

(1)	O_1	(2)	O_n
(3)	O_n	(4)	O_n
(5)	O_1	(6)	O_1
(7)	~~O_n~~	(8)	O_n
(9)	O_{n^2}	(10)	O_{n^2}
(11)	O_1.	(12)	O_1
(13)	Low	(14)	moderate

-1

△寫法不對，請照O( )寫，不要自創.

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 額外成本(兩次後)	(9)	(10)
Visualization	(11)	(12)
Swaps	(13)	(14)
Flexibility	(15)	(16)
Overall	(17)	(18)

(1)

Array put each elements side by side, and each memory space is one element

ex: [1][2][3][4]

(2)

Linked list's elements are made by a space and a pointer, not always side by side.

ex: [1]→[2]→[3]→null

(3)

Array access element by calling it index numbers ex: array[i]

Student ID: Student Name:

(4)

Linked list need to travel from head to find the elements we want.

(5)

Array seldom to use extra variables to record elements or something else.

(6)

Linked list must use extra variables (pointer) to record elements to swap, or just mark the head.

(7)

Array use its index number to traversal all elements. We can also traversal from the end to front.

(8)

Linked list use the next pointer to find next elements, it's impossible to traversal from end to front if there aren't previous pointer.

(9)

Array has lower overhead because we can easily find element by index number.

ex: `int x = array[1]`

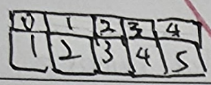
Student ID:

Student Name:

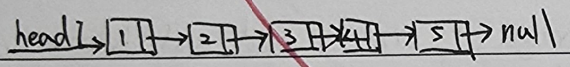
(10)

Linked list has more overhead because there is no way to directly find a elements. we need to started at a node that has pointer.

(11)

Array : 

(12)

Linked list 

(13)

swap(0,1); int tmp = Array[0] ; Array[0]= Array[1]; Array[1] = tmp ;

(14)

swap(0,1): Find linked list[0] , use pointer1 record it ; Find linked list[1] , use pointer2 record.
int tmp= pointer1->value ; pointer1->value = pointer2-> value; pointer2->value = tmp;

(15)

Array is not so flexible to insert , delete elements . Both must move the whole array.

Student ID: Student Name:

(16)

Linked list has more flexible to insert and delete because each node only connect with the previous one and the next one.

(17)

Array is easy to find a sigle element with its index number, has less overhead. but: not so flexible to add and delete element because it fixed space.

(18)

Linked list is flexible to change it size, add and delete element. However it has more overhead to find element because each element has no fixed position.

-35.