Student ID: 1130824 Student Name: 何宥安

**Course: Data Structures (CSE CS203A)**

**Assignment III: Linked List Selection Sort**

**Student Worksheet Companion**

42

## 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)

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: 113084 Student Name: 何宥孚

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 ([ ___ ], [ ___ ]).

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: 1130824 Student Name: 何宥安

**A4. Discussion (68 pts)**

Guiding Questions:

- How many swaps/exchanges are performed? 5 in A1 in A3
- How expensive is traversal for arrays vs. linked lists? array = O(n) = linked lists = O(n)
- What memory/overhead differences do you see? array = contiguous linked lists = non-contiguous
  ↳ require extra memory for pointers
- Which representation is easier to visualize? array
- Which would you choose for implementing selection sort and why?
  array, Because more complex due to pointer handing,

**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: 113084 Student Name: 何宥萍

| | | | |
|---|---|---|---|
| (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

Student ID: 1130824 Student Name: 何宥萍

**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)

Elements stored in contiguous memory locations. Elements arranged by index.

(2)

Elements are stored in sequence, but not in contiguous memory locations. Each node contains data and pointer,

(3)

Direct acess with index => O(1)

Student ID: 113087# Student Name: 何易洋

(4)

Sequential acess only O(n). Accessing elements requires traversing from the head node.

(5)

Very few extra variables needed, usually just an index or temporary variable.

(6)

Each node requires an extra pointer to link to the next node.

(7)

Visit all elements. => O(n)

(8)

Traverse via pointers. => O(n)

(9)

low extra overhead.

Student ID: 113087H Student Name: 何宥安

(10)

Requires extra memory for pointers.

(11)

Easy to represent, element positions are clear.

(12)

Requires nodes and pointer connections to visualize.

(13)

Swap values via index.

(14)

By exchanging node values.

(15)

Fixed size, resizing requires reallocation.

Student ID: 113084 Student Name: 何雨芊

(16)

Linked lists can grow or shrink during runtime. → Dynamic

(17)

Easier to implement and use and fast and acess.
But less flexible, resizing requires reallocation.

(18)

More flexible, but implement is more complex, requires careful pointer and NULL handing.

-6.