

Student ID: 1137724

Student Name: 6/8/24

Course: Data Structures (CSE CS203A)

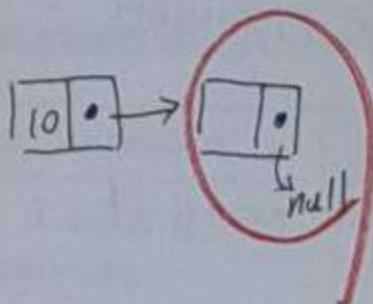
Assignment III: Linked List Selection Sort

Student Worksheet Companion

/ 22

A1. Linked List Representation Drawing (5 pts)

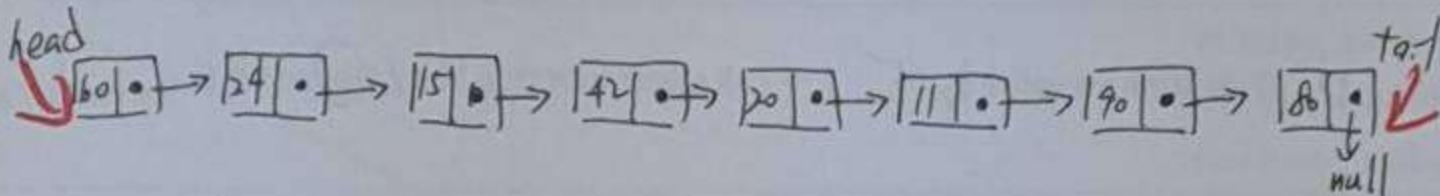
- a. (2 pts) Instructions: Draw a visual representation of a single node with next pointer that contains the initialized integer 10



1

- 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 | •] →



-2

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	[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:

8

[8]

Student Name:

→ [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 \rightarrow call swap function Yes; swap (60, 8).

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

Step 2 ($i = \underline{24}$): Minimum value [11] \rightarrow call swap function Yes / No; swap ([24], [11]).
head \rightarrow [8|•] \rightarrow [11 |•] \rightarrow [15 |•] \rightarrow [42 |•] \rightarrow [20 |•] \rightarrow [24 |•] \rightarrow [90 |•] \rightarrow [60 |NULL]

Step 3 ($i = \underline{15}$): Minimum value [11] \rightarrow call swap function Yes / No; swap ([11], [15]).
head \rightarrow [8|•] \rightarrow [11 |•] \rightarrow [15 |•] \rightarrow [42 |•] \rightarrow [20 |•] \rightarrow [24 |•] \rightarrow [90 |•] \rightarrow [60 |NULL]

Step 4 ($i = \underline{42}$): Minimum value [20] \rightarrow call swap function Yes / No; swap ([42], [20]).
head \rightarrow [8|•] \rightarrow [11 |•] \rightarrow [15 |•] \rightarrow [20 |•] \rightarrow [42 |•] \rightarrow [24 |•] \rightarrow [90 |•] \rightarrow [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)	$O(n)$
(13)	Slow	(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) store in contiguous block of memory

(2) store in Non-contiguous, each node allocated dynamically

(3) direct access with index (0(i))

Student ID:

Student Name:

(4)

Sequential access only pointers ($O(n)$)

(5)

No extra overhead

(6)

Requires extra memory for pointers

(7)

use for loop

(8)

use pointer chain

(9)

No extra overhead

(10)

requires extra memory for pointers

(11)

look like forms or boxes

(12)

have links between nodes.

(13)

easier to implement and use

(14)

more complex due to pointer handling

(15)

fixed sized, must be defined at declaration

(16)

Dynamic , can grow at runtime.

(17)

Fast access , best for random access

(18)

Flexible size , best for dynamic data

-24