Bismillahir Rahmanir Rahim

International Islamic University Chittagong

Department of Computer Science & Engineering

Mid Term Examination, Spring 2023

CSE-2321 Data Structures

[Solution]

1(a)	**(i)** ● **Entity**: Each book is an entity ● **Entity Set**: Collection of books ● **Attributes**: Name of Authors, Title of Book, Edition, Publisher's Name, Year, ISBN **(ii)** ISBN can serve as the primary key.

Nayeem Mahmood / C161026

1(b)

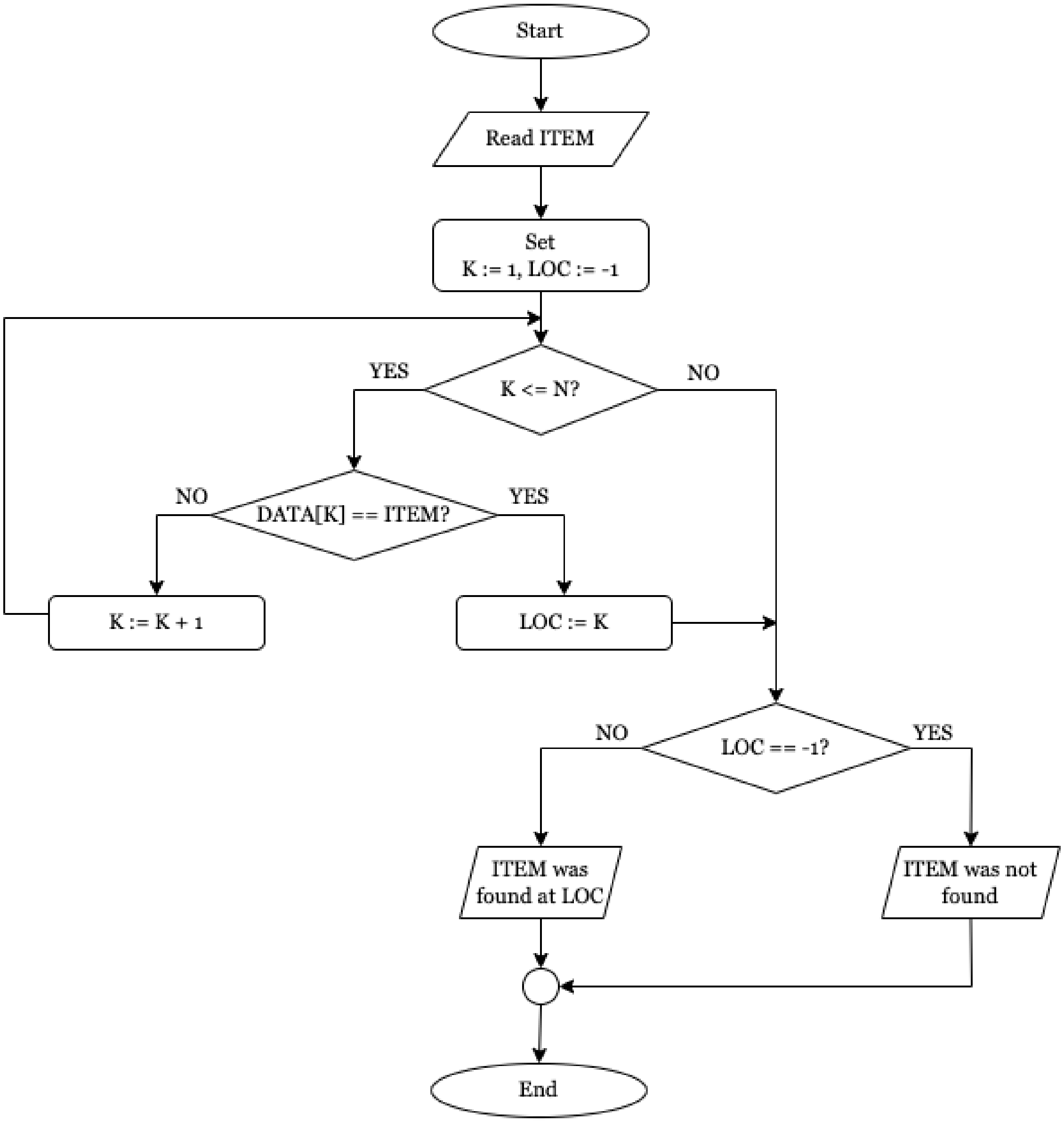

Figure: Flow chart of Linear Search Algorithm

Nayeem Mahmood / C161026

1(b) OR

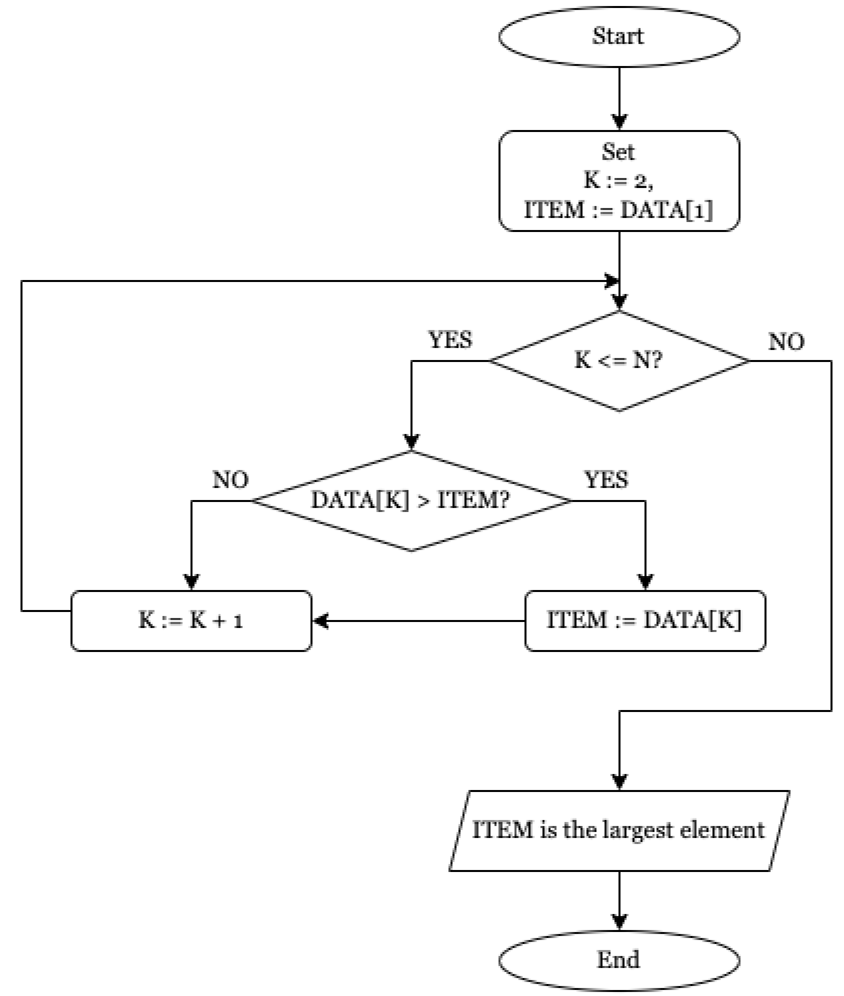

Figure: Flow chart of Finding the Largest Element in Array

1(c)

(i) O(1). Each statement will be executed once.
(ii) O(n). The loop will be executed n/2 times.
(iii) O($\log_2 n$). The loop will be executed $\log_2 n$ times.
(iv) O(n^2). The program will be executed n + (n - 1) + (n - 2) + … + 3 + 2 + 1 times.

1(d)

This color denotes a match, and this color denotes a mis-match.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
T:	**a**	**b**	**a**	**b**	**b**	**a**	**b**	**b**	**b**	**a**	**b**	**a**	**a**	**a**	**a**	**b**	**b**	**b**	**C**
P1:	**a**	**a**	**b**																**2**
P2:		**a**	**a**	**b**															**1**

P3:			**a**	**a**	**b**														**2**
P4:				**a**	**a**	**b**													**1**
P5:					**a**	**a**	**b**												**1**
P6:						**a**	**a**	**b**											**2**
P7:							**a**	**a**	**b**										**1**
P8:								**a**	**a**	**b**									**1**
P9:									**a**	**a**	**b**								**1**
P10:										**a**	**a**	**b**							**2**
P11:											**a**	**a**	**b**						**1**
P12:												**a**	**a**	**b**					**3**
P13:													**a**	**a**	**b**				**3**
P14:														**a**	**a**	**b**			**3**

Total comparisons, C = 24. INDEX (P, T) = 14.

2(a)

(i)

- **Step 1**: Start
- **Step 2**: Set K := 23001, N := 23099
- **Step 3**: Repeat Step 4 to Step 5 while K <= N
- **Step 4**: If STUCGPA[K] > 3.90
  Print K
- **Step 5**: Set K := K + 1
- **Step 6**: End

(ii)

- **Step 1**: Start
- **Step 2**: Set K := 23001, N := 23099, COUNT := 0
- **Step 3**: Repeat Step 4 to Step 5 while K <= N
- **Step 4**: If STUCGPA[K] > 3.30
  Set COUNT := COUNT + 1
- **Step 5**: Set K := K + 1
- **Step 6**: Print COUNT
- **Step 7**: End

2(b)

(i)

Given, A(3:10, 1:10, -3:10).

- Length of 1st dimension, $L_1 = 10 - 3 + 1 = 8$.
- Length of 2nd dimension, $L_2 = 10 - 1 + 1 = 10$.
- Length of 3rd dimension, $L_3 = 10 - (-3) + 1 = 14$.
- Total number of elements $= L_1 \times L_2 \times L_3 = 8 \times 10 \times 14 = 1120$.

(ii)
Given, BASE (A) = 200, W = 4. We've to find the effective indices and address of A[8, 6, 4].

The effective indices of the subscripts are respectively,
- $E_1 = 8 - 3 = 5$
- $E_2 = 6 - 1 = 5$
- $E_3 = 4 - (-3) = 7$

Now,
- $E_1L_2 = 5 \times 10 = 50$
- $E_1L_2 + E_2 = 50 + 5 = 55$
- $(E_1L_2 + E_2)L_3 = 55 \times 14 = 770$
- $(E_1L_2 + E_2)L_3 + E_3 = 770 + 7 = 777$

So, $\text{ADDRESS}(A[8, 6, 4]) = \text{BASE}(A) + W((E_1L_2 + E_2)L_3 + E_3)$
$= 200 + 4(777)$
$= 3308$.

2(c) Comparisons are shown using **this color**, and interchanges are shown using **this color**.

- **Pass 1**:

C	H	I	T	T	A	G	O	N	G
C	H	I	T	T	A	G	O	N	G
C	H	I	T	T	A	G	O	N	G
C	H	I	T	T	A	G	O	N	G
C	H	I	T	T	A	G	O	N	G
C	H	I	T	A	T	G	O	N	G
C	H	I	T	A	G	T	O	N	G
C	H	I	T	A	G	O	T	N	G
C	H	I	T	A	G	O	N	T	G

At the end of Pass 1, we've got CHITAGONGT.

- **Pass 2**:

C	H	I	T	A	G	O	N	G	T
C	H	I	T	A	G	O	N	G	T
C	H	I	T	A	G	O	N	G	T
C	H	I	T	A	G	O	N	G	T
C	H	I	A	T	G	O	N	G	T
C	H	I	A	G	T	O	N	G	T
C	H	I	A	G	O	T	N	G	T
C	H	I	A	G	O	N	T	G	T

At the end of Pass 2, we've got CHIAGONGTT.

- **Pass 3**:

C	H	I	A	G	O	N	G	T	T
C	H	I	A	G	O	N	G	T	T
C	H	I	A	G	O	N	G	T	T
C	H	A	I	G	O	N	G	T	T
C	H	A	G	I	O	N	G	T	T
C	H	A	G	I	O	N	G	T	T
C	H	A	G	I	N	O	G	T	T

At the end of Pass 3, we've got CHAGINGOTT.

- **Pass 4**:

C	H	A	G	I	N	G	O	T	T
C	H	A	G	I	N	G	O	T	T
C	A	H	G	I	N	G	O	T	T
C	A	G	H	I	N	G	O	T	T

C	A	G	H	I	N	G	O	T	T
C	A	G	H	I	N	G	O	T	T

At the end of Pass 4, we've got CAGHIGNOTT.

- **Pass 5**:

C	A	G	H	I	G	N	O	T	T
A	C	G	H	I	G	N	O	T	T
A	C	G	H	I	G	N	O	T	T
A	C	G	H	I	G	N	O	T	T
A	C	G	H	I	G	N	O	T	T

At the end of Pass 5, we've got ACGHGINOTT.

- **Pass 6**:

A	C	G	H	G	I	N	O	T	T
A	C	G	H	G	I	N	O	T	T
A	C	G	H	G	I	N	O	T	T
A	C	G	H	G	I	N	O	T	T

At the end of Pass 6, we've got ACGGHINOTT.

- **Pass 7**:

A	C	G	G	H	I	N	O	T	T
A	C	G	G	H	I	N	O	T	T
A	C	G	G	H	I	N	O	T	T

At the end of Pass 7, we've got ACGGHINOTT.

- **Pass 8**:

A	C	G	G	H	I	N	O	T	T
A	C	G	G	H	I	N	O	T	T

At the end of Pass 8, we've got ACGGHINOTT.

- **Pass 9**:

A	C	G	G	H	I	N	O	T	T

At the end of Pass 9, we've got ACGGHINOTT.

- Total number of Comparisons, C = 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1
  = **45**
- Total number of Interchanges, D = 5 + 5 + 4 + 3 + 2 + 1 + 0 + 0 + 0
  = **20**

2(c) OR

ModifiedBinarySearch (DATA, LB, UB, ITEM, LOC)

- **Step 0**: Start
- **Step 1**: Set BEG := LB, END := UB, MID := INT ((BEG + END) / 2).
- **Step 2**: Repeat Step 3 & 4 while BEG ≤ END and DATA[MID] ≠ ITEM
- **Step 3**: If ITEM < DATA[MID], then:
  Set END := MID - 1
  Else
  Set BEG := MID + 1
- **Step 4**: Set MID := INT ((BEG + END) / 2)
- **Step 5**: If ITEM < DATA[MID], then:
  Set LOC := MID
  Else
  Set LOC := MID + 1
- **Step 6**: Insert ITEM into DATA[LOC] using typical array insertion algorithm
- **Step 7**: Exit

2(d)

(i) <u>Advantage of Binary Search over Linear Search</u>: The time complexity of BS is O($\log_2 n$), which is way better than LS of O(n).

(ii) <u>Disadvantage of Binary Search over Linear Search</u>: To perform BS in any data structure, it needs to be sorted, and must have direct access to any datapoint. But, to perform LS in any data structures, those aren't mandatory at all.

3(a)

PUSH (STACK, TOP, MAX_SIZE, ITEM)
[This procedure pushes an ITEM onto the STACK]

- **Step 1**: If TOP = MAX_SIZE, then:
  Print Overflow, and Return
- **Step 2**: Set TOP := TOP + 1
- **Step 3**: Set STACK[TOP] := ITEM
- **Step 4**: Return

POP (STACK, TOP, ITEM)

[This procedure deletes the TOP item from STACK, and assigns it to ITEM]

- **Step 1**: If TOP = 0, then:
  Print Underflow, and Return
- **Step 2**: Set ITEM := STACK[TOP]
- **Step 3**: Set TOP := TOP - 1
- **Step 4**: Return

3(b)

Given,
N = 8,
TOP = 4,
STACK = 11, 22, 33, 44, _, _, _, _

(i) POP (STACK, ITEM)
ITEM = 44
STACK = 11, 22, 33, _, _, _, _, _
TOP = 3

(ii) PUSH (STACK, 55)
STACK = 11, 22, 33, 55, _, _, _, _
TOP = 4

(iii) PUSH (STACK, 66)
STACK = 11, 22, 33, 55, 66, _, _, _
TOP = 5

(iv) POP (STACK, ITEM)
ITEM = 66
STACK = 11, 22, 33, 55, _, _, _, _
TOP = 4

3(c)

Given,

P: 3, 1, -, 2, ^, N, 4, /, 2, *, +, 5, -,)

Symbols	**Stack**	**Explanation**
(1) 3	3	
(2) 1	3, 1	
(3) -	2	3 - 1 = 2
(4) 2	2, 2	
(5) ^	4	2^2 = 4
(6) N	4, N	

(7) 4	4, N, 4	
(8) /	4, N/4	
(9) 2	4, N/4, 2	
(10) *	4, N/2	(N/4) * 2 = N/2
(11) +	4 + N/2	
(12) 5	4 + N/2, 5	
(13) -	N/2 - 1	4 + N/2 - 5 = N/2 - 1
(14))		End of evaluation

If the last two digits of my id be XY, then

N/2 - 1 = XY

=> N/2 = XY + 1

=> N = 2 * (XY + 1)

[For example, if XY be 26, N = 2 * (26 + 1) = 54.]

3(d) Given,

Q: (A * B ^ D) / (E - F) + G)

Symbols	**Stack**	**Postfix Exp. P**
(1) (	((	
(2) A	((	A
(3) *	((*	A
(4) B	((*	A B
(5) ^	((* ^	A B
(6) D	((* ^	A B D
(7))	(	A B D ^ *
(8) /	(/	A B D ^ *
(9) (	(/ (	A B D ^ *
(10) E	(/ (	A B D ^ * E

(11) -	(/ (-	A B D ^ * E
(12) F	(/ (-	A B D ^ * E F
(13))	(/	A B D ^ * E F -
(14) +	(+	A B D ^ * E F - /
(15) G	(+	A B D ^ * E F - / G
(16))		A B D ^ * E F - / G +

3(d) OR

We need a **STACK** to implement this algorithm. Let **SEQ** be the bracket sequence of size **N**. The following algorithm determines whether SEQ is a valid bracket sequence or not.

- **Step 0**: Start
- **Step 1**: Set K := 1, VALID := TRUE. STACK is empty.
- **Step 2**: Repeat Step 3 to Step 7 while K $\le$ N
- **Step 3**: If SEQ[K] = ‘(’ or ‘{’, or ‘[’, then:
  Push SEQ[K] to STACK, and go to Step 7
  Else If SEQ[K] = ‘)’, then:
  Go to Step 4
  Else If SEQ[K] = ‘}’, then:
  Go to Step 5
  Else If SEQ[K] = ‘]’, then:
  Go to Step 6
- **Step 4**: If STACK is non-empty and STACK[TOP] = ‘(’, then:
  Pop from STACK, and go to Step 7
  Else
  Set VALID := FALSE, and go to Step 8
- **Step 5**: If STACK is non-empty and STACK[TOP] = ‘{’, then:
  Pop from STACK, and go to Step 7
  Else
  Set VALID := FALSE, and go to Step 8
- **Step 6**: If STACK is non-empty and STACK[TOP] = ‘[’, then:
  Pop from STACK, and go to Step 7
  Else
  Set VALID := FALSE, and go to Step 8
- **Step 7**: Set K := K + 1
- **Step 8**: If VALID = TRUE, then:
  Print “Sequence is valid.”
  Else
  Print “Sequence is not valid.”
- **Step 9**: End