No:

Date:

IN2110 - October 2019

Question 01

(c) The for loop iterates over the 'myArray' from 0 to size - 1. Which is ~~the~~ n, the size of array. Therefore the method find2C has a time complexity of O(n).

Question 02

(a) → Elements of an array can be randomly accessed in a constant (O(1)) time.

→ An element can be insereted to the end of an array in a constant time (O(1)).

→ Arrays are simpler to understand and implement.

→ Arrays has good cache locality due their contagious memory allocation.

→ Arrays are more memory efficient compared to linked lists.

(b)
```
int Search (node **head, int value)
    node *current = *head
    While (current ~~current~~ != NULL)
        if (current -> data == value)
            return 1
        current = current -> next
    return 0
```

No:

Date:

(c)

	Index	key	Next
head -->	[0]	10	[6]
	[1]	N/A	N/A
	[2]	87	[8]
	[3]	9	NULL
	[4]	31	[3]
	[5]	10	[2]
	[6]	8	[5]
	[7]	N/A	N/A
	[8]	90	[4]

	Index	key	Next
head -->	[0]	10	[6]
	[6]	8	[5]
	[5]	10	[2]
	[2]	87	[8]
	[8]	90	[4]
	[4]	31	[3]
	[3]	9	NULL

(i) [3]

(ii) [5]

(iii)

	inder	key	Next
head -->	[0]	10	[6]
	[6]	8	[5]
	[5]	10	[7]
	[7]	999	[2]
	[2]	87	[8]
	[8]	90	[4]
	[4]	31	[3]
	[3]	9	NULL

	Index	key	Next
head -->	[0]	10	[6]
	[1]	N/A	N/A
	[2]	87	[8]
	[3]	9	NULL
	[4]	31	[3]
	[5]	10	[7]
	[6]	8	[5]
	[7]	999	[2]
	[8]	90	[4]

No:

Date:

Question 03

(a)

(i) TRUE

Using a stack is a one of the easiest ways to print sorted list in a reverse order. We just have to push the names in order to a stack and pop one by one. Then we can simply print the each poped element.

(ii) FALSE

Breadth-First Search should be implemented using a queue. BFS process naturally follows FIFO principle.

(b)

$Q_1 = [\ ]$

$Q_2 = [5, 5, 7, 7, 12, 12, 4, 4, 0, 0, 4, 4, 6, 6]$

(c)

(i)

```
Void   stackToQueue ( ) {
    while (! S.empty( )) {
        q. enQueue
```

```
Void   stackToQueue (Stack S, Queue q) {
    while ( ! S.empty( )) {
        q.enqueue(S.pop( ));
    }
}
```

No:.....

Date:.....

(ii)
```
int sizeOfQueue(Queue q) {
    int count = 0;
    while (!q.empty) {
        count++;
        q.dequeue();
    }
    return count;
}
```

No:

Date:

IN2110 - January 2019

Question 01

(a) Number of Comparisons - 21

Number of Swaps - 11

(b) Selection sort

In selection ~~process~~ sort, sorting process starts from the begining of the array. In the intial pass the smallest element in the array would come to the index 0. In the second pass the 2nd smallest element would come to the index 1. This process will go on for the other passes too.

Therefore even in the part way through in a selection sort, we can see ~~the~~ a sorted array at the begining of the array.

(c) → When input size of the array is small.

→ When the array is partially sorted.

Question 02

(a) Big O notation is used to describe the upper bound or worst-case time complexity of an algorithm. It represents the growth rate of the alogorithm's running time as the user input size increases.

$O(n^2)$ means algorithm's running time, in the worst case, is propotional to square of the input size.

(Algorithm's time complexity quadratically increases with the input size)

No:

Date:

(b)

(i) mult (5, 7)
mult (4, 7)
mult (3, 7)
mult (2, 7)
mult (1, 7)

(ii) 2000 times

(c)(i)
```
int F(int n) {
    if (n == 0)
        return 1;
    if (n == 1)
        return 1;
    return F(n-1) + F(n-2);
}
```

(ii) Question 03

(a) → Dynamic size
→ Reduced memomry waste since memory is allocated dynamically.
→ No need for contagious memory.

(b)
```
void insertAtMiddle (struct node** head_ref, struct node *new_node,
                     int pos) {
    struct node* prev = *head_ref;
    for (int i=0; i< pos-2; i++) {
        prev = prev -> next;
    }
    new_node -> next = prev -> next;
    prev_node -> next = new_node;
}
```

No:

Date:

(c)

(i) 3

(ii) 10

(iii)

Index	key	Next
[0]	10	[6]
[1]	N/A	N/A
[2]	87	[8]
[3]	9	NULL
[4]	31	[3]
[5]	10	[7]
[6]	8	[5]
[7]	999	[2]
[8]	90	[4]

Question 04

(a)

(i) c, e, d, b, a

(ii) push(a)
push(b)
pop() → b
push(c)
~~pop()~~ push(d)
pop() → d
pop() → c
pop() → a
push(e)
Pop() → e

No:

Date:

(b)

(i) S2. S2.Push(S1.Pop()) ⟶ A to S2

S2. push(S1. pop()) ⟶ B to S2

S3. push(S1. pop()) ⟶ C

S3. push(S1. pop()) ⟶ D

S3. push(S2. pop()) ⟶ B

S3 push (S2. pop()) — A

(ii) S2. push(S1. pop()) ⟶ A to S2

S2. push(S1. pop()) ⟶ B to S2

S3. push (S1. pop()) ⟶ C to

S1. push(S2. pop()) ⟶ B to S1

S3. push(S2. pop()) ⟶ A

S2. push(S1. pop()) ⟶ B to S2

S3 push(S1. pop()) ⟶ D

S3 push(S2. pop()) ⟶ B