No:

Date:

Question 01

(a) Recursion involves solving a problem by breaking it down into smaller sub-problems. and solving each ~~problem~~ subproblem in a recursive manner. It relies on the function calling itself to perform the repetitive task. Iteration involves using loops to repeatedly excute a block of code until a certain condition is met.

Recursion uses more memory compared to iteration. Iteration uses fixed amount of memory to store loop variables more efficiently while recursion call adds new frames to stack.

(b)
```
for (int i=0; i<count; i++) {          (count → arr length)
    int minLoc = i;
    for(int j=i+1; i<count; j++){
        if (~~cont~~ arr[j] < arr[minLoc]) {
            minLoc = j;
        }
    }
    int temp = arr[i];
    arr[i] = arr[minLoc];
    arr[minLoc] = temp;
}
```

No:

Date:

(c) Advantages:-

1. Simplicity:- Selection sort is realatively simple to understand and implement.
2. Space Efficiency:- It operates directly on the input array without requiring additional data structure.

Disadvantage:-

Inefficient: It has time complexity $O(n^2)$ - As a result, it become inefficient for large data sets.

(d)

Original Array: 8 2 5 9 7 8

8 2 5 9 7 8

2 8 5 9 7 8

2 5 8 9 7 8

2 5 7 9 8 8

2 5 7 8 9 8

2 5 7 8 8 9

Sorted Array : 2 5 7 8 8 9.

No:.....

Date:.....

⑧ 1 : Starting from 0 th index, compare the first and the second elements.

2 : In the first element is greater than the second element, they are swapped

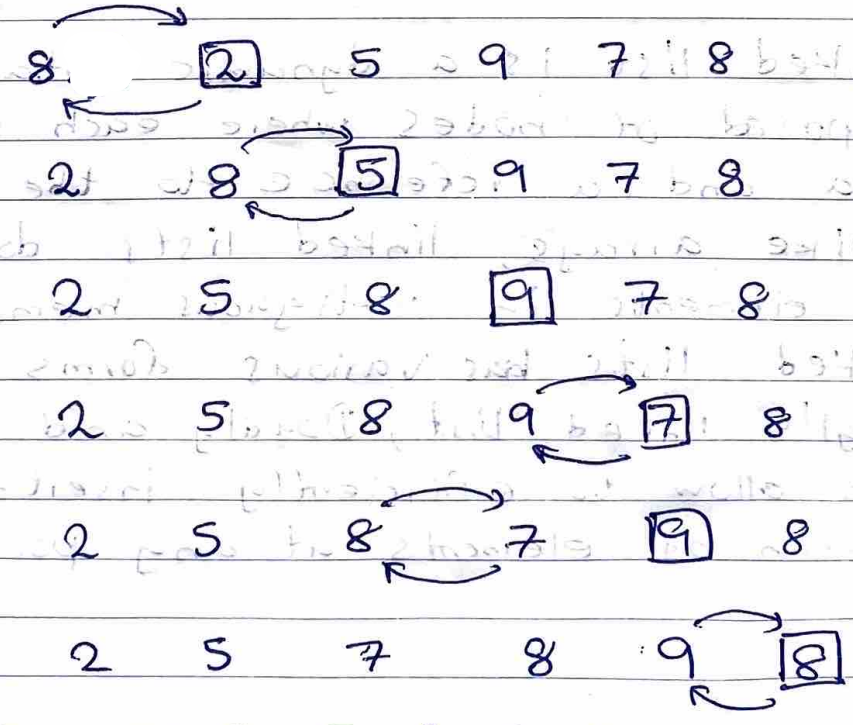
3 : Now, compare the second and the third elements, swap them if they are not in order.

4 : The above process goes on until the last elements.

⑦

	Best	Worst	Average
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$

⑨ original Array : 8 2 5 9 7 8



Sorted : 2 5 7 8 8 9

No:.....

Date:.....

Question 02

(a) (i) Stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. where the last item added is the first one to be removed. It has two main operators - 'Push' to add element to the top ~~and~~ of stack and 'pop' to remove the topmost element

(ii) Queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. It is similar to a queue of people waiting in a line where the first person to join the queue is the first one to be served. Has two primary operations 'Enqueue' adds an elements to the end and 'Dequeue' removes the elements from the front.

(iii) Linked list is a dynamic data structure composed of nodes where each node contains data and a reference to the next node. Unlike arrays, linked lists does not store its elements in contiguous memory locations. Linked lists has various forms such as singly linked list, Doubly and circular. This allow to efficiently insertion and deletion of elements at any position.

No:

Date:

(b) (i) `#define MAX_SIZE 100`
`int top = -1;`

(ii) `int STACK_DATA[MAX_SIZE];`

(iii)
```
int isEmpty() {
    return top == -1;
}
```

(iv)
```
int isFull() {
    return top == MAX_SIZE - 1;
}
```

(v)
```
void push(int value) {
    if (isFull()) {
        printf("stack overflow");
        return;
    }
    STACK_DATA[++top] = value
}
```

(vi)
```
int pop() {
    if (isEmpty()) {
        printf("Stack Underflow");
        return;
    }
    return STACK_DATA[top--];
}
```

No:.....

Date:.....

(c)
```
void empty_stack() {
    if (top == -1)
        return;
    pop()
    empty_stack();
}.
```

Question 03

(a)

Array	Linked lists.
→ Size if fixed	→ Size is not fixed
→ Occupies less memory	→ Occupies more memory
→ Deletions is not possible	→ Deletion of elements is ~~Delete~~ possible
→ Memory allocation From stak	→ Memory Allocation From heap

(b) ~~A counter~~ A scenario where customers visit a service counter, such as a bank. It helps organize and serve customers in the order they arive, ensuring and minimize wait time.

No:

Date:

(c) (i)
```
int isEmpty() {
    if (front == NULL && rear == NULL) {
        return 1;
    } else {
        return 0;
    }
}
```

(ii)
```
int peek() {
    if (front == NULL) {
        printf("Empty Queue");
        return -1;
    }
    return front->data;
}
```

(iii)
```
void enqueue() {
    struct node *temp = (struct node*) malloc
                        - (sizeof(struct node));
    temp->data = data;
    temp->next = NULL;

    if (front == NULL && rear = NULL.) {
        front = rear = temp;
    } else {
        rear->next = temp;
        rear = temp;
    }
}
```

No:.....

Date:.....

d)

	S	Q
S.push(4)	4	
S.push(10)	4 10	
Q.enqueue(6)	4 10	6
Q.enqueue(8)	4 10	6 8
X = S.pop() (10)	4	6 8
S.push(2)	4 2	6 8
Q.enqueue(X)	4 2	6 8 10
Y = Q.dequeue() (6)	4 2	8 10
S.push(X)	4 2 10	8 10
S.push(Y)	4 2 10 6	8 10

No:

Date:

Question 4

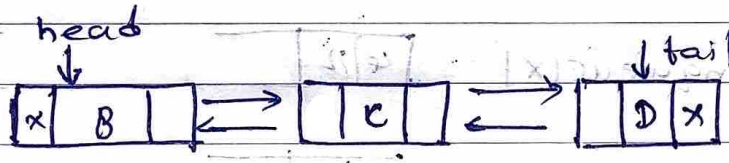
(a) Static Memory Allocation:

Memory is allocated for variables during the compilation time, and the allocation remains fixed for the lifetime of program. It suitable for variable with known sizes and lifetimes.

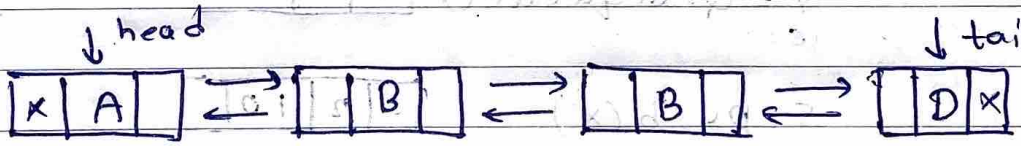
Dynamic Memory Allocation:

Memory is allocated at runtime using functions like 'malloc'. It allows for variable-sized data structures and flexible memory management.

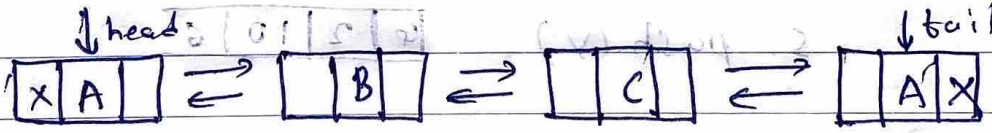
(b) (i)



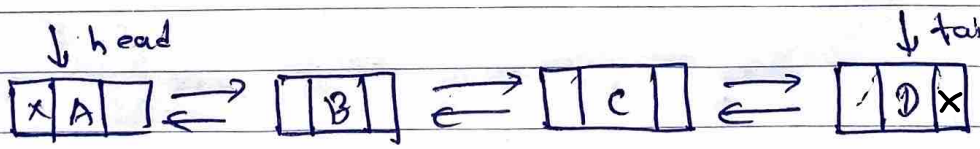
(ii)



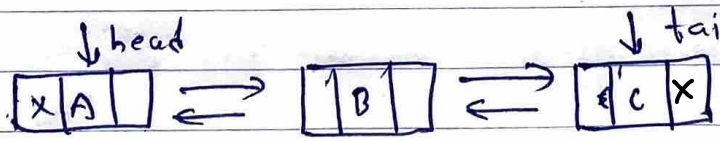
(iii)



(iv)



(v)



No:

Date:

(c) (i) Time complexity refer to the estimation of the amount of time an algorithm takes to run, based on the size of the input. It mesure how the runtime of an algoritm increases as the input size grows.

(ii) Space complexity refers to the estimation of the amount of memory an algoritm requires to run, based on the size of the input. It mesures how the memory usage of an algoritm increases as the input size grows.

(d)
```
void testFunction(int n) {
    for (n)        ---  → n
      for (10)          → 10
        for (n)         → n
          for (20)      → 20
```

Total Time complexity $= n \times 10 \times n \times 20$

$= 200\, n^2$

In Big'O $\Rightarrow O(n^2)$ //