NAME : AYUSH PRASHANT KALASKAR.
CLASS : S.E. COMP - I.
ROLL NO. : 69. PRN. : F22111074
SEMESTER : SEM - III (2023-2024)
SUBJECT : DATA STRUCTURE LABORATORY. [DSL]

ASSIGNMENT : E-29 :-

o ALGORITHM OF THE ~~ABOVE~~ CODE:-

STEP:1: Start.
STEP:2: Intialize the Queue and the front and rear pointers.
STEP:3; ~~Perform~~ ~~the~~ ~~selected~~ Display the menu and prompt the user to enter a choice.
STEP:4: Perform the selected operation on the users choice.
- IF the user chooses to add a job, check if the queue is full. IF it is, display an error message and Exit. Otherwise, add the job to the queue and display the updated queue.
- IF the user chooses to delete a job, check if the queue is empty. If it is, display an error message and Exit. Otherwise, remove the job from the front of the queue and display the updated queue.
- IF the user chooses to display the queue, print all the jobs in the queue in order.

STEP:5:-
Ask the user ~~if~~ they want to continue. If they do repeat the steps 2 to 5, Otherwise Exit the program.
STEP:6:-
Stop.

Algorithm for Enqueue Operation:-

1.) Create a function Enqueue to add a job to the end of the queue.
2.) Check if the queue is full or not.
3.) If Full, Display an Error message. and Exit.
4.) Otherwise, Increment the rear pointer to point the next empty space in the queue.
5.) The Function thus adds a new job to the queue at the rear pointer.

Algorithm for Dequeue Operation:-

1.) Create a function Dequeue to remove a job from the front of the queue.
2.) Check if the queue is Empty or not.
3.) If its Empty, Display an Error message and exit.
4.) Otherwise the function removes the front pointer from the queue.
5.) The function thus increments the front pointer to point to the next job in the queue.

Algorithm to Display queue:-

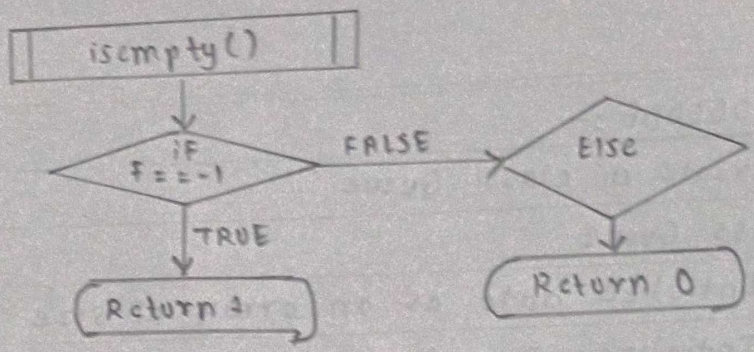
1.) Create a function 'void Display' to print all the jobs in the queue in order.
2.) The Function iterates over the queue from the front pointer to the rear pointer
3.) For each job in the queue, the function prints the job to the console.

PSEUDOCODE :-

```
# create a class Queue
class Queue
  Intialize data as an array of size 20
  Intialize f and r as integers , both set to -1.

  Function Queue()   # Constructor.
    set f to -1
    set r to -1

  Function isEmpty() --> Integer
    If f is -1
       Return 1
    Else
       Return 0

  Function isFull() --> Integer.
    If r is greater than or equal to 20
       Return 1
    Else
       Return 0

  Function Enqueue (x: Integer)
    If isFull() is equal to 1
       Display " Job queue is Full "
    Else
       If f is -1
          Set f to 0
       Increment r by 1
       Set data[r] to x
```

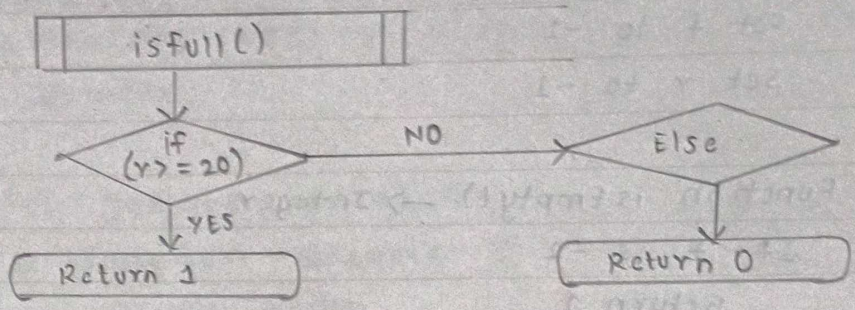
(1.) FLOWCHART FOR THE FUNCTION isempty():-

→



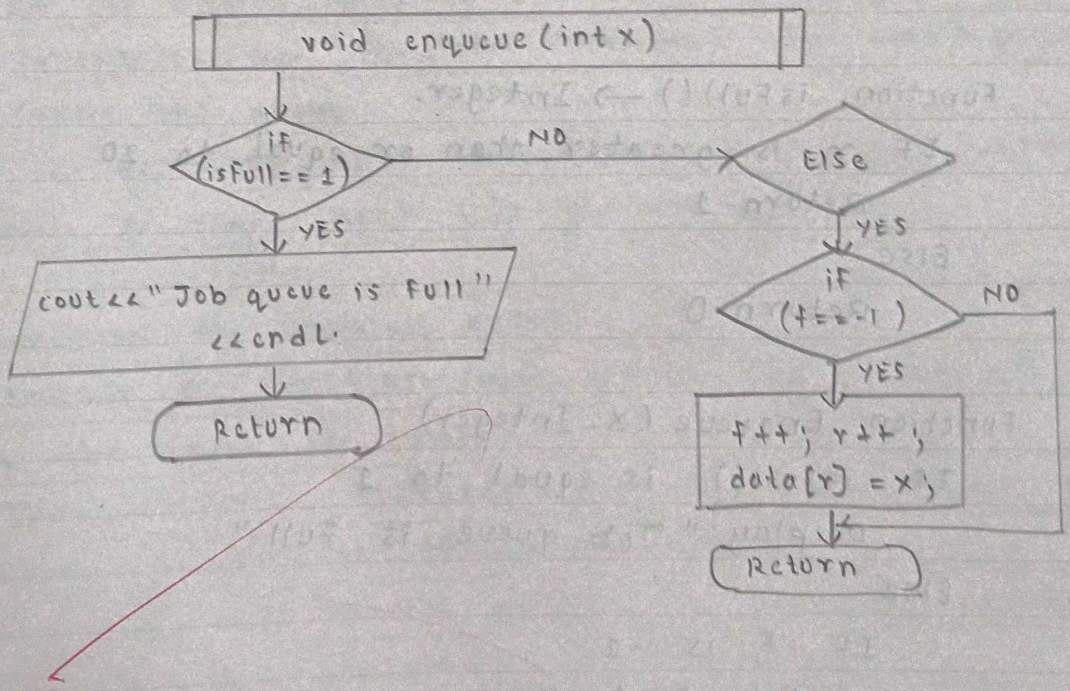
(2.) FLOWCHART FOR THE FUNCTION isfull():-

→



(3.) FLOWCHART FOR THE FUNCTION void enqueue()

→



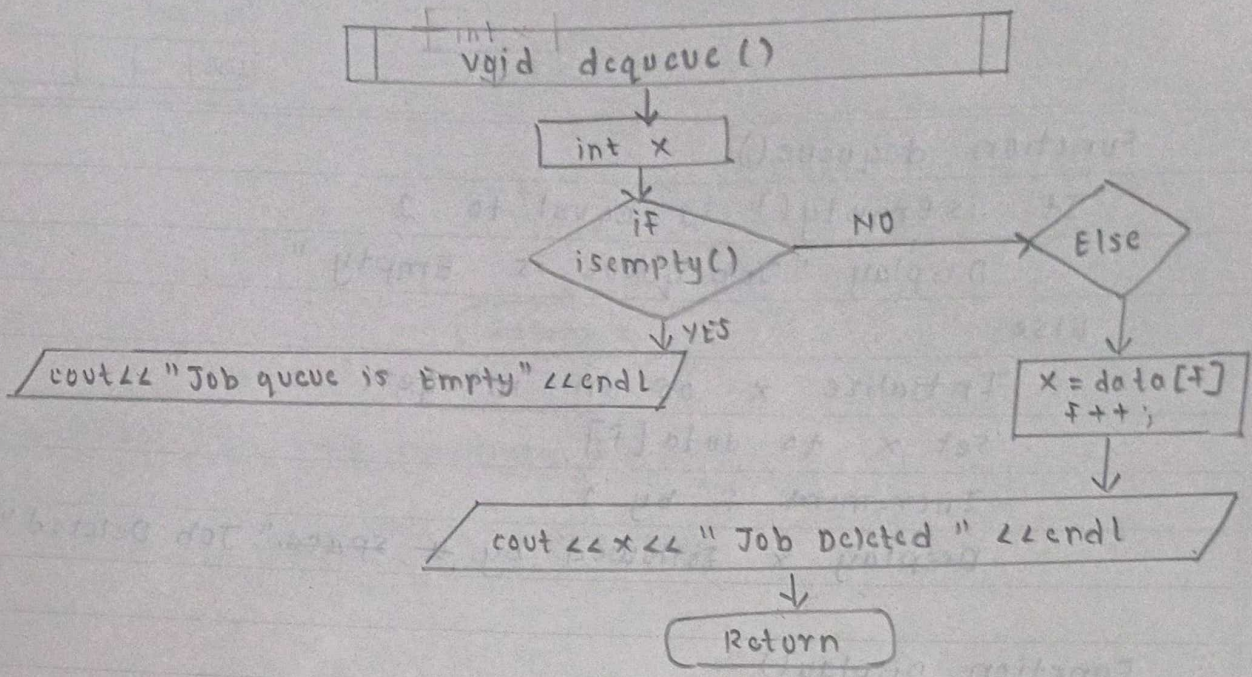
Function dequeue()
If isEmpty() is equal to 1
Display "Job queue is Empty"
Else
Intialize x as an integer.
set x to data[f]
Increment f by 1
Display x followed by ~~a space~~. "Job Delcted"

Function Display()
Display "Job queue is as follows:"
For i from f to r
Display data[i] followed by a space.
Display a new line.

Function main()
Intialize ch, n, x, d as integers.
Create a queue object q
Display "Enter the Number of Jobs in the Queue"
Read n
Display (Input) " Enter Jobs "
For i from 1 to n
Read d
Call q.enqueue(d)
Do
Display " ****** MENU ******* "
Display " 1.) Add Job "
Display " 2.) Delete Job "
Display " 3.) Display "
Display a new Line
Display " Enter your Choice"
Read ch.

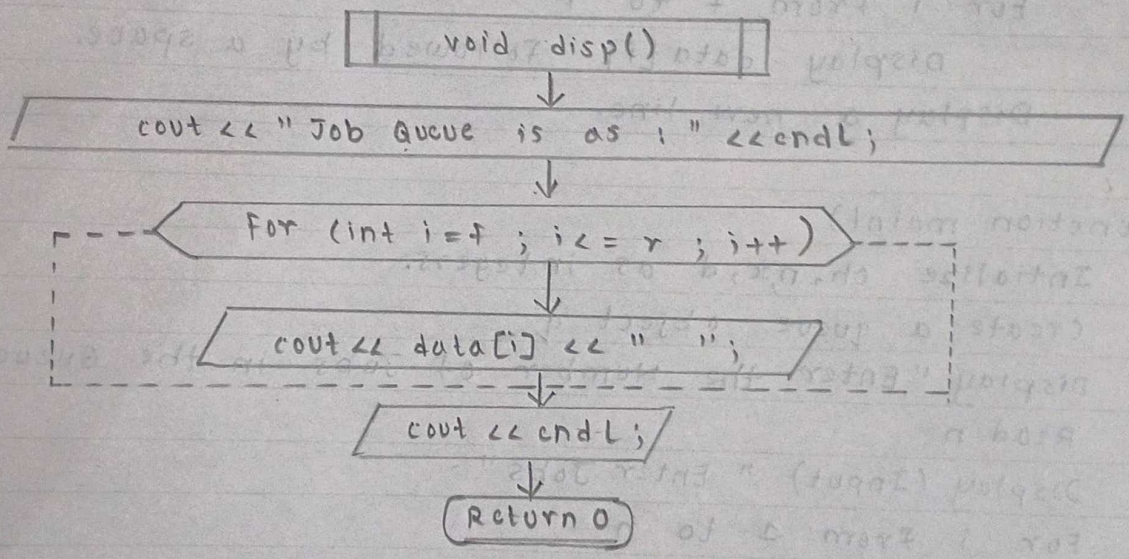
(4.) FLOWCHART FOR THE FUNCTION :- void dequeue()

→



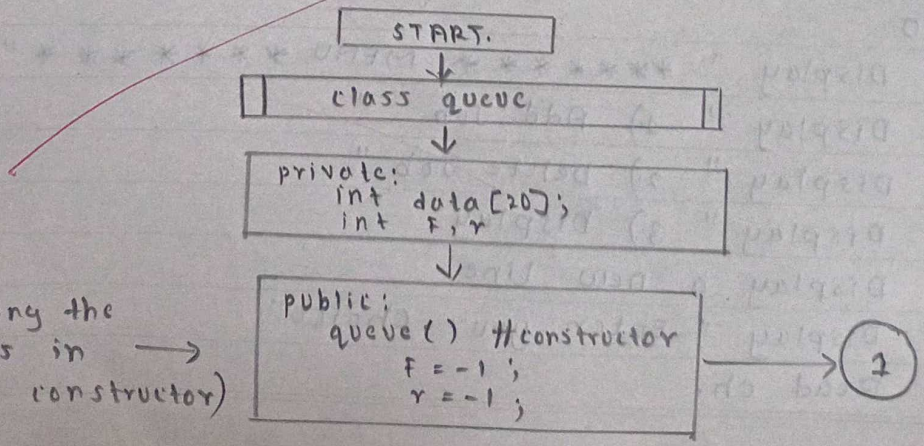
(5.) FLOWCHART FOR THE FUNCTION:- void display()

→



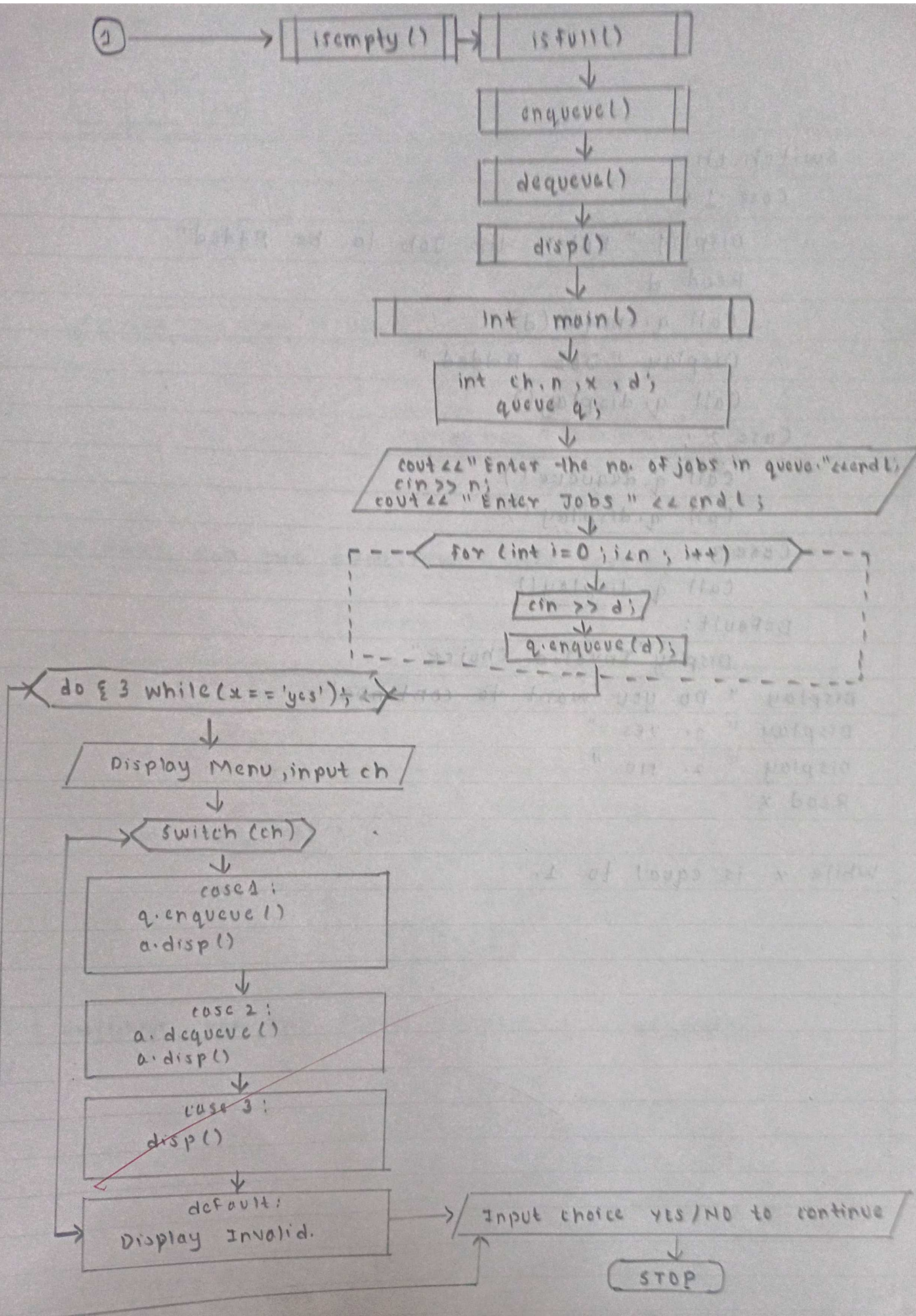
(6.) FLOWCHART FOR THE MAIN FUNCTION :- int main()

→



(Intializing the variables in default constructor) →

```
Switch ch
    Case 1 :
        Display " Enter the Job to be Added"
        Read d
        Call q.eneque(d)
        Display " Job Added "
        Call q.display()
    Case 2 :
        Call q.dequeue()
        Call q.display()
    Case 3 :
        Call q.display()
    DeFault:
        Display "Invalid Choice"
Display " DO you want to continue"
Display " 1. YES "
Display " 2. NO "
Read X

While x is equal to 1.
```


1
isempty()
isfull()
enqueue()
dequeue()
disp()
int main()
int ch, n, x, d;
queue q;
cout << "Enter the no. of jobs in queue:" << endl;
cin >> n;
cout << "Enter Jobs " << endl;
for (int i=0; i<n; i++)
cin >> d;
q.enqueue(d);
do { } while(x=='yes');
Display Menu, input ch
switch(ch)
case 1:
q.enqueue()
a.disp()
case 2:
a.dequeue()
a.disp()
case 3:
disp()
default:
Display Invalid.
Input choice YES/NO to continue
STOP

ASSIGNMENT: E-29 :-

D QUESTIONS:-

Q1) Describe Queue Operations and its usuage a job queue?

ANS. The various queue operations are:-

1. Enqueue : Adding a new item or job to the back of the queue.
2. Dequeue : Removing and processing the item or job at the front of the queue.
3. Peek: Inspecting the item at the front of the queue without removing it.
4. Size and Empty checks: Determining the number of items in the queue and checking if its Empty.

The various usuage of a queue as a job queue are: Web servers, customer support, Manufacturing and Data support./Processing.

Q2) Describe how to implement queue using stack

ANS. The implementation of a queue can be achieved by using two stacks to stimulate the behaviour of a queue, where one stack is used for Enqueue (Adding Elements to the back of the queue) and the other stack is used for dequeue (removing elements from the front of the queue.)

STEP: 1:-

Create two stacks 'stack1' and 'stack2'

'stack1' — Used for Enqueue operation.

'stack2' — Used for Dequeue Operation.

STEP: 2:-

To Enqueue an element, push it onto 'stack1'

STEP:3:- Dequeue Operation.

- If 'stack2' is empty, pop all elements from 'stack1' and push them onto 'stack2' in reverse order.
- Then, pop from 'stack2' to dequeue an element

STEP:4:- Front Operation.

To get the front element of the queue without removing it, we can actually check it without dequeing the top of 'stack2' or the bottom of 'stack1'

Eg:-

```
class QueueUsingStacks:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []

    def enqueue(self, element):
        self.stack1.append(element)

    def dequeue(self):
        if not self.stack2:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        if self.stack2:
            return self.stack2.pop()
        else:
            return None.

    def front(self):
        if not self.stack2:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
```

```
if self.stack2
    return self.stack2[-1]
else
    return None.
```

Q3.) What do you mean by Linear Data structures? Give Examples of it.

ANS.
- Linear Data structures are a type of Data structure in Computer science that organize and store data elements in a linear or sequential manner.
- In other words, the Data elements are arranged in a specific order, and each element has a unique predecessor and successor. (except for the first and last elements.)
- Linear Data structures are often used for the tasks that involve processing data sequentially.
- Examples of Linear Data structure:

→ i.) Array
ii.) Linked List.
iii.) Stack
iv.) Queue.
v.) Vectors / Lists.

Q4.) Why Queue is an Efficient Data structure to assign Jobs?

ANS. Queue's are an Efficient Data structure for assigning and managing jobs for several reasons:

1. First-in-First-Out (FIFO) Principle. The job waiting the longest gets processed first which is a fair and predictable way to manage Job assignments.
2. Orderly Processing: Jobs are processed and organized in a systematic order they were recieved. This can

help maintain a structured and organized workflow.

3. Scalability: Queues are scalable, and additional jobs can be added to the queue as they arrive.
4. Reliability: Queues can help ensure that no job is overlooked or forgotten. All jobs are added to the queue, and they remain in a queue until they are successfully processed, reducing the risk of job loss or omission.
5. Error handling: In case of failures or errors during job processing, queues can provide mechanisms for handling these situations, such as requeuing or moving jobs to a dead letter queue for further analysis.
6. Synchorization.
7. Effective Prioritization
8. Concurrency.
9. Load Balancing.
10. Monitoring and Analytics.

These Factors make queue an Efficient Data structure.