2a. O (1)

2b. O (1)

2c. O (1)

3a. 1. J22

2. J17

3. J56

4. J45

5. J30

6. J12

7. J18

3b. Importantance of job, Importance of job submitter, and execution time.

3c. Starvation occurs when a job doesn’t get executed because the other jobs submitted take priority over and over again.

3d. As jobs that are being starved wait, they age, the longer they wait the more they age. Then, when that job has reached a age threshold, it is executed regardless of its priority. This helps eliminate starvation

3e. I would use a circular que array that has prioritys assigned to the elements with time complexity O(1).

4.

Deq(q,i)

Increment the front pointer to 5, dropping the element 4.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Front |  | Back |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  | 41 | 12 | 34 |  |

Deq (q,j);

Increment the front pointer to 6, dropping the element 5.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | Front | Back |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  |  | 12 | 34 |  |

Enq(q,81);

Add 81 to the element following the Back, element 8.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | Front |  | Back |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  |  | 12 | 34 | 81 |

Enq(q,72);

Add 82 to the element following the Back, element 0.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Back |  |  |  |  |  | Front |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 72 |  |  |  |  |  | 12 | 34 | 81 |

Enq(q,38);

Add 38 to the element following the Back, element 1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Back |  |  |  |  | Front |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 72 | 38 |  |  |  |  | 12 | 34 | 81 |

deq(q,k);

Increment the front pointer to 7, dropping the element 6

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Back |  |  |  |  |  | Front |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 72 | 38 |  |  |  |  |  | 34 | 81 |

b. Adv:The implementation of a circular que is simple, does not use dynamic memory, and uses constant time. This means there is no memory loss, and at the time, we can add keep adding elements as long as deletion occurs and the stack is not full. In a linear que, this is not possible.

c. Dis: we must know how many elements we will need to store. We cannot manipulate the number of elements we can store.