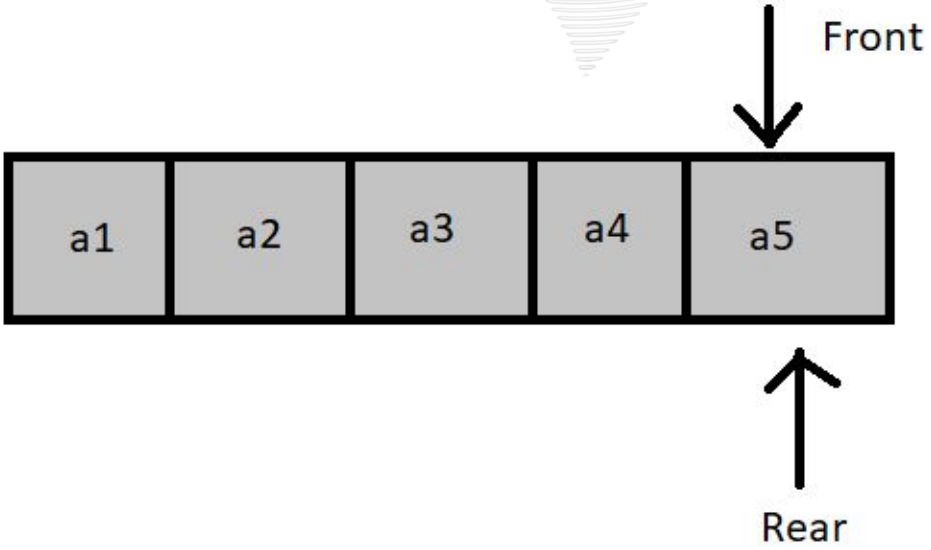
Disadvantage of Implementing Queue using Array

1)Waste of Space:

While deleting elements , we are moving the front pointer to the next index of the array . For example after deleting the first element of array i.e index 0 , our front pointer will move to the next index i.e index 1 .Now we will never go back to that index of array since there is no element present there and hence that index is useless for us . But still , since that index is part of our array it will occupy some space in our memory which we are never going to use . Similar thing will happen if we remove one more element and hence more space will get unused and hence more space will get wasted . Hence there is a waste of space while implementing a queue using arrays.

2)A condition will arise when a queue will be Empty as well as full

Now let us see this condition with the help of a diagram.



Now in the above case , the queue is empty as well as full and this is very ambiguous . The rear pointer is pointing to the last index of the array and hence our queue is full . On the other hand , as all the elements are deleted from the queue and hence now the front is pointing to the same index as that of the rear . Since , rear==front , the queue is empty as well . But this is impossible . A queue can't be full and empty at the same time . The problem is arising because we are only using each location only once . In ideal cases , as soon as an element is removed from the queue and that index becomes empty , we should use it again to store any new incoming element . But we are not doing any such thing here , we are just using each location once and hence this problem arises .

Solution of these Disadvantages

CIRCULAR QUEUE

Now ,the biggest problem we are facing while using an array for implementing a queue is that we are not reusing the location from where the element is deleted . As soon as the element is deleted from a particular index , we are abandoning that index and not using that index anymore . But in ideal cases , we should reuse the index to store any new incoming element. This gives rise to the concept of circular queue. In a circular queue we cover up the limitation of a linear queue of not using the same location twice by just adding one single condition . As soon as the queue becomes full (i.e rear pointer value becomes equal to size-1) , instead of terminating the program and saying the queue is full , we redirect our rear pointer to the beginning of the array (i.e index 0) by using modulus operator(%) . Now , if the element has been deleted from that index , we can insert our new element there by using the rear pointer as we normally do , hence using the same location twice , we cover up the limitation of the linear queue . Hence here we are avoiding wastage of space in the ambiguous case by just adding one condition.