Q2-B. Report [10 marks] You should also submit a pdf file with no more than 500 words that have a short-written justification for your design of the program reflecting on the algorithm and efficiency of the code. It should contain a screenshot showing an example of the output for part 2.

In question two, I was tasked to write two methods for a circular array. These were an enqueue method and dequeue method. I was provided with a skeleton explaining how to do so as well as all other necessary code and tests. I started by looking at the code and understanding what had been given to me. The variable names and extensive comments gave me many clues. I also ran the program and found errors for only the enqueue and dequeue program. This helped work out exactly what I needed to do.

My enqueue algorithm is solid and simple. The algorithm uses an if block to check if the queue is full. If it is, a new array is created double the size and all items are copied to this new queue. After this the new queue is set as the main queue and front set to 0. I thought this was simple as it is using the provided method (is full) to check if a queue needs to be extended. This way also only requires one if block as if the queue isn't full, that section can be skipped. Outside the if block, the tail is declared. It took me a little while to work out how to find this, however I worked it out using modulus after understanding how items need to wrap around the array. Finally, the item is added, and the number of elements is incremented. I thought this was a very efficient and elegant solution to the problem. If I could change anything, I would add more tests and perhaps a try catch block to rule out all errors.

My dequeue algorithm was similar to my enqueue. I started by using an if block and the given (is empty) method to check if the queue was empty. If this was true, I threw an illegal state exception as specified in the comments above. I thought this was an efficient way to do it as it utilised the given method and threw an exception in minimal lines. Outside this if block, I created a remove object. This was the item to be removed from the front of the queue. Next, I set the front of the queue to null as it helped show the size of the array and exactly what had been dequeued. I liked this idea as I originally found it hard to tell what had been removed when running the program. After this I set the front equal to the item before it. Finally, I decremented the number of elements before returning the removed item. Overall, I thought this was a good solution as it was simple, small and efficient. I also utilised many of the given methods and outputted the result in an easy to understand way. As mentioned in the enqueue method, I would improve by adding more tests.

SCREENSHOT BELOW

```
PS C:\Users\Admin>  & 'C:\Program Files\Ja
[null, null, null]
ENQUEUE
[a, null, null]
ENQUEUE
[a, b, null]
ENQUEUE
[a, b, c]
DEQUEUE
a
[null, b, c]
ENQUEUE
[d, b, c]
ENQUEUE
[b, c, d, e, null, null]
ENQUEUE
[b, c, d, e, f, null]
ENQUEUE
[b, c, d, e, f, g]
DEQUEUE
b
[null, c, d, e, f, g]
DEQUEUE
c
[null, null, d, e, f, g]
ENQUEUE
[h, null, d, e, f, g]
DEQUEUE
d
[h, null, null, e, f, g]
DEQUEUE
e
[h, null, null, null, f, g]
DEQUEUE
f
[h, null, null, null, null, g]
DEQUEUE
g
[h, null, null, null, null, null]
DEQUEUE
h
[null, null, null, null, null, null]
DEQUEUE
Queue is empty
[null, null, null, null, null, null]
PS C:\Users\Admin>
```