Question			Marks
1	1	**All marks for AO1 (understanding)** **Max 2 for advantages of dynamic data structures** No wasted memory; Can grow as more data is added to the data structure // no limit on number of items that can be added (except due to hardware limitations); Resources only allocated as they are needed (with static data structures they are allocated at creation even if not needed until later); **Max 2 for disadvantages of dynamic data structures** Additional memory needed for pointers; Can result in memory leak (if memory that is no longer needed is not returned to the heap); Can take longer to access an item directly (for data structures that allow this); **A.** can take longer to add a new item to the data structure (as memory needs to be allocated)	4
1	2	**All marks for AO1 (understanding)** Check that the queue is not already full; (if it isn't) then add 1 to the value of the rear pointer; then add the new item to the position indicated by the rear pointer; **Alternative answer** Check that the queue is not already full; (if it isn't) then add the new item to the position indicated by the rear pointer; then add 1 to the value of the rear pointer; **Max 2** if any errors **Max 1** if circular queue has been described	3

1	3	**All marks for AO1 (understanding)** Starting with the item at the rear of the queue move each item back one place in the array; Until you (reach the start of the queue or) find an item with the same or higher priority than the item to add; **NE.** same priority **NE.** higher priority Add the new item in the position before that item; **A.** answers which have the front of the queue as the last item in the array, start at the front and move each item forward one until the correct insertion point is found. **A.** answers that start from the front of the queue until position to insert item is found and then start at the back and move each item back one until position to insert item is found.	3

Question			Marks
02	1	**All marks for AO1 (understanding)** Static data structures have storage size determined at compile-time / before program is run / when program code is translated / before the data structure is first used // dynamic data structures can grow / shrink during execution / at run-time // static data structures have fixed (maximum) size // size of dynamic data structures can change; Static data structures can waste storage space / memory if the number of data items stored is small relative to the size of the structure // dynamic data structures only take up the amount of storage space required for the actual data; Dynamic data structures require (memory to store) pointers to the next item(s) // static data structures (typically) do not need (memory to store) pointers; Static data structures (typically) store data in consecutive memory locations // dynamic data structures (typically) do not store data in consecutive memory locations; **Max 3**	**3**

Question			Marks
3		**All marks AO1 (understanding)** 1. Check the queue is not already full; 2. Compare the value of the (rear) pointer with the maximum size of the array; 3. If equal then (rear) pointer becomes zero; **A.** index of the first position in the array instead of zero 4. Otherwise, add one to the (rear) pointer; 5. Insert new item in position indicated by (rear) pointer; **Alternative answer 1** 1. Check the queue is not already full; 2. Compare the value of the (rear) pointer with the maximum size of the array minus one; 3. If equal then (rear) pointer becomes one; **A.** index of the first position in the array instead of one 4. Otherwise, add one to the (rear) pointer; 5. Insert new item in position indicated by (rear) pointer; **Alternative answer 2** 1. Check the queue is not already full; 2. Add one to the (rear) pointer; 3. Compare the value of the (rear) pointer with the maximum size of the array; 4. If equal then (rear) pointer becomes zero; **A.** index of the first position in the array instead of zero 5. Insert new item in position indicated by (rear) pointer; **Alternative answer 3** 1. Check the queue is not already full; 2. Add one to the (rear) pointer; 3. Compare the value of the (rear) pointer with the maximum size of the array plus one; 4. If equal then (rear) pointer becomes one; **A.** index of the first position in the array instead of one 5. Insert new item in position indicated by (rear) pointer; **Alternative answer 4** 1. Check the queue is not already full; 2. Add one to the (rear) pointer; 3. Use modulus/modulo operator/function with new value of (rear) pointer; 4. Use modulus/modulo operator/function with maximum size of array; 5. Insert new item in position indicated by (rear) pointer; **Max 4** if any errors	**5**

Question			Marks
4	1	**All marks AO1 (understanding)** When an item is removed from a linear queue; **A.** by implication that problem arises when items have been deleted all other items need to be shuffled up one // this can result in unusable space in the array // this can result in only being able to add a limited number of items to the queue (eg only ever being able to add five items to a queue which uses an array of size 5); **Alternative answer** No need to shuffle items up one // no unusable spaces; After deleting an item from a circular queue;	2
4	2	**All marks AO1 (understanding)** 1. Check that the queue is not already empty // check to see if the current size is 0; **R.** reference to array instead of queue 2. if it is then deal with (underflow) error // (If it is not then) process/dequeue the front item in the queue; **R.** reference to array instead of queue (unless reference to array is made using value of front pointer) **R.** if dequeue would only happen under some of the correct circumstances 3. Reduce the value of the variable used to store the current size by 1; 4. Check if the front is in the last position of the array and if it is set it to the first position in the array (**A.** 0 or 1 instead of first position in the array); 5. (Else) add 1 to the value of the front pointer; **Alternative to mark points 1 and 3 if no current size variable is used:** calculate the current size; compare this to 0; **Alternative to mark points 1 and 3 if no current size variable is used:** check to see if the front/rear is −1;; **Alternative to mark points 4 and 5:** add 1 to the value of the front pointer; then set it to the remainder from dividing the value of front by the maximum size of the queue; **Alternative to mark points 4 and 5:** add 1 to the value of the front pointer; then if the pointer has passed the end of the array set it to point to the start of the array. **DPT.** Rear instead of front **Max 4 if any errors**	5