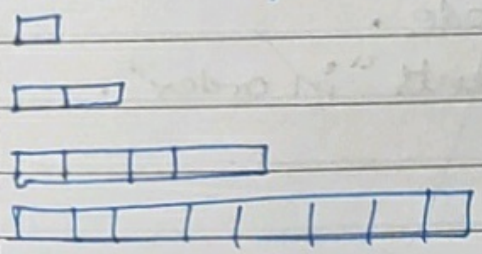
Homework 3.1

Yes. We can use Growable Array Based queue

```
Algorithm enqueue(Object o)
    if size() < N
        then Q[r] = o
             r++
    else Object A[] = new Object[2N]
         for i ← 0 to N-1
             A[i] ← S[i]
         S ← A
         S[r] = o
```

↳ We are using growth strategy since in next part, n is very large. We could use tight strategy according to the situation.

Steps to enqueue n elements



Phase 0 size 1

Phase 1 Size 2

} each phase

Phase 3 size 8

In i^{th} phase, size of array is 2^i

Total cost = $2^i + 2^{i-1} + 2^{i-1}$

- 2^i ↑ allocating fresh memory
- 2^{i-1} ↑ cost of copying
- 2^{i-1} ← cost of all addi... done in this ph...

for n elements, we will have $\log n$ such phases.

cost = $2 + 4 + \cdots + 2^{\log n + 1} = 4n - 1$

- Total steps required = $4n - 1$
- Total phases = $\log n$

Page No.
Date

Homework 3.2

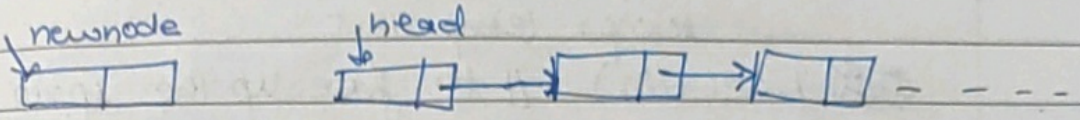
- Enqueue adds to the head, dequeue removes from tail.
- List is singly linked.

(a) enqueue()

Enqueue will take constant time i.e O(1)

Step-1.

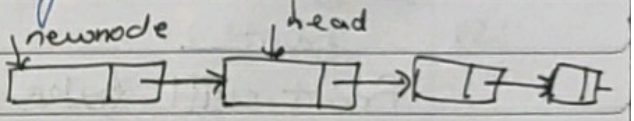
create a newnode and add data to it.



Step-2

put head in the "next" attribute of created node i.e

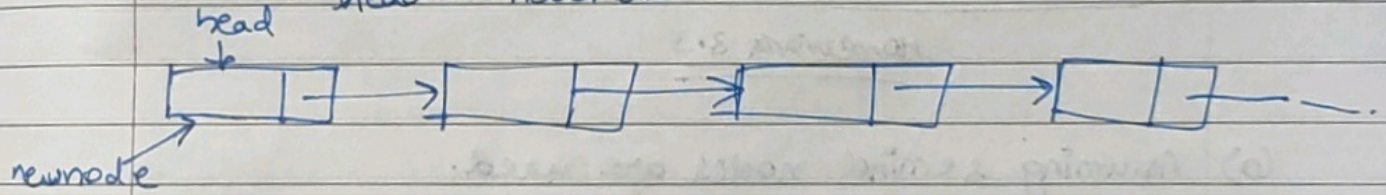
newnode.next ← head



Step-3

Change the head pointer so that it now points to the newnode.

head ← newnode



Data enqueued succesfully!!!

Page No
Date

Homework 3-3

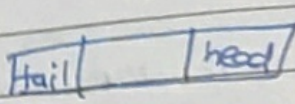
A) Assuming sentinel nodes are not present

Assume that we are given the position

Algorithm deleteNode (DoublyLinkedList list, int pos)

```
if (list.size() = 0)
    System.out.println("List is Empty.")
else
  if (targetNode.prev = NULL)  // Node is head node
      temp ← targetNode
      targetNode ← targetNode.next
else
    if (pos = 1)              // Node is head node
      temp ← list.head
      list.head ← list.head.next // move head
      free (temp) // free memory
      list.head.prev = NULL  // setup head reference
else
      i ← 1
      temp ← list.head
      while (temp != NULL and i < pos)
          temp = temp.next , i++
      node1 = temp.prev
      node2 = temp.next
      node1.next = node2
      node2.prev = node1
      free (temp)   // free memory.
```

B. Assuming sentinal nodes are present

Sentinel node 

Assume that we are given the position

```
Algorithm deleteNode (DoublyLinkedList list, int position)
    head ← list.sentinel.next
    tail ← list.sentinel.prev

    if (list.size = 0)
        System.out.println("List is empty")

    else
        if (pos = 1)    // Node is headnode
            list.sentinel.next ← head.next
            free(head)
            head ← head.next
            head.prev ← NULL
        else if (pos = list.size())
            list.sentinel.prev ← tail.prev
            free(tail)
            tail ← tail.prev
            tail.next ← NULL
        else i ← 1, temp ← head
            while (while temp != NULL and i < N)
                temp ← temp.next
            node1 = temp.prev
            node2 = temp.next
            node1.next = node2
            node2.prev = node1
            free(temp)
```

Homework 3.4

As we know, we will maintain an "entry point" reference to the list. Let's call it head.

Algorithm insertAtBeginning (Node newnode)

```
if( head = NULL)
        head ← newnode
        head.next ← newnode
else
        newnode.next ← head.next
        head.next ← newnode
```