R-2.1 **Answer:**

|  |
| --- |
| Algorithm insertBefore**(**p**,**e**)**  Input **:** position of node p where newnode will be inserted before **this** node and element of newNode  Output**:** newNode with element e will be inserted into list**.**  newNode**<-**createNewNode**(**e**)**  tmp**<-**p**.**prev  tmp**.**next **<-** newNode  newNode**.**next **<-** p  newNode**.**prev **<-** tmp  p**.**prev **<-** newNode |
| Algorithm insertFirst**(**e**)**  Input **:** element e**,**which will be inserted into first of the linked list  Output**:** newNode with element e will be inserted into first position of list**.**  newNode**<-**createNewNode**(**e**)**  tmp**<-**head**.**next  head**.**next**<-**newNode  newNode**.**next**<-**tmp  tmp**.**prev **<-** newNode  newNode**.**prev **<-** head |
| Algorithm insertLast**(**e**)**  Input **:** element e**,**which will be inserted into last position of the linked list  Output**:** newNode with element e will be inserted into last position of list**.**  newNode**<-**createNewNode**(**e**)**  tmp**<-**tail**.**prev  tail**.**prev **<-**newNode  newNode**.**prev**<-**tmp  tmp**.**next **<-** newNode  newNode**.**next **<-** tail |

C-2.1 **Answer:**

|  |
| --- |
| Algorithm findMiddle**(**L**)**  Input **:** List L with odd number of nodes  Output **:** middle position of L  p**<-**L**.**first**()**  q**<-**L**.**last**()**  **while** p **!=** q **do**  p**<-**L**.**after**(**p**)**  q**<-**L**.**before**(**q**)**  **return** p |

C-2.2 **Answer:**

|  |  |
| --- | --- |
| S1**<-**Empty Stack  S2**<-**Empty Stack  enqueue**(**val**)**  **if** size**()** **=** N **-** 1 then  **throw** FullQueueException  S1**.**push**(**val**)** | For, enqueuer(),running time = O(1) |
| dequeue**()**  **if** S2**.**isEmpty**()** then  **while** **!**S1**.**isEmpty**()** **do**  S2**.**push**(**S1**.**pop**())**  **if** **!**S2**.**isEmpty**()** then  **return** S2**.**pop**()**  **else**  **throw** EmptyStackException | O(1)  O(n)  O(n)  O(1)  O(1)  So, total running time = O(n) |

**C-2-3 Answer:**

|  |  |
| --- | --- |
| Q1**<-**Empty Queue  Q2**<-**Empty Queue  push**(**val**)**  **if** size**()** **=** N **-** 1 then  **throw** FullStackException  Q1**.**enqueue**(**val**)** | Running time of push() operation = O(1) |
| pop**()**  **if** Q2**.**isEmpty**()** then  **while** **!**Q1**.**isEmpty**()**  Q2**.**enqueue**(**Q1**.**dequeue**())**  **if** **!**Q2**.**isEmpty**()** then  Q2**.**dequeue**()**  **else**  **throw** EmptyQueueException | O(1)  O(n)  O(n)  O(1)  O(1)  So, total running time for pop operation = O(n) |

**C-2-4 Answer:**

|  |
| --- |
| Algorithm perm**(**S**,** int n**)**  Input**:** Sequence S with n elements  Output**:** List L containing all the permutation    **if** n **=** 1 then  L**.**insertLast**(**S**)**  **return;**  **while** i **<** S**.**size**()** **do**  S**.**swapElements**(**S**.**rankOf**(**i**),**S**.**rankOf**(**n**-**1**))**  perm**(**S**,**n**-**1**)**  S**.**swapElements**(**S**.**rankOf**(**i**),**S**.**rankOf**(**n**-**1**))** |
| Running time = O(n!) |

**C-2-5 Answer:**

|  |
| --- |
| Algorithm insertAtRank0**(**obj**)**  Input**:** the object obj **for** inserting  **if** V**.**size**()** **=** n**-**1  **throw** fullException  f**<-(**f **-** 1 **+** n **)** mod n  V**[**f**]** **<-** obj |
| Algorithm removeAtRank0**()**  **if** **!**V**.**isEmpty**()**  f**<-(**f **+** 1 **)** mod n  **else**  **throw** emptyVectorException |
| Algorithm insertAtRankEnd**(**obj**)**  Input**:** the object obj **for** inserting  **if** V**.**size**()** **=** n**-**1  **throw** fullException  V**[**r**]** **<-** obj  r **<-** **(**r **+** 1**)** mod n |
| Algorithm removeAtRankEnd**()**  **if** **!**V**.**isEmpty**()**  r**<-(**r **-** 1 **+** n**)** mod n  **else**  **throw** emptyVectorException |