**R-3.19 Answer:**

|  |  |
| --- | --- |
| Algorithm removeElement**(**x**)**  Input**:** key x **for** remove item  Output**:** item deletion with the key  p**<-**first position top list  p**<-**find**(**x**,**p**)**//return position of the key  **if** p **!=** NULL  **while** p **!=** NO\_SUCH\_KEY **do**  **if** p**.**before **=** MINUS\_INF **^** p**.**after **=** PLUS\_INF  p**.**before**.**below**.**above **<-** NULL  p**.**after**.**below**.**above **<-** NULL  a**<-**p**.**below  tmp **<-** p**.**before  tmp**.**after **=** p**.**after  p**.**after**.**before **=** tmp  p**<-**a | Algorithm find**(**x**,**p**)**  Input**:**key x**,**postion p  Output**:** postion of the key x  y**<-**key**(**p**.**after**)**  **if** x **=** y  **return** p**.**after  **else** **if** x **>** y then  **return** find**(**x**,**p**.**after**)**  **else**  **return** find**(**x**,**p**.**below**)**  **return** NULL |

**C-4.16 Answer:**

|  |  |
| --- | --- |
| Algorithm isExistTwoEqualElement**(**S**)**  Input**:** Sequence S with n elements  Output**:** **true** or **false** **if** two equal elements exis in the sequence**,**otherwise **false**  D**<-new** Dictionary**(**HashTable**)**  **for** each x of S **do**  cnt **<-**D**.**findElement**(**x**)**  **if** cnt **!=** NO\_SUCH\_KEY then  D**.**insertItem**(**x**,**0**)**  **else**  **return** **true**  **return** **false** | L1: O(1)  L2:O(n)  L3:O(n)  L4:O(n)  L5:O(n)  L6:O(1)  L7:O(1)  Total running time = O(n) |

**C-4.18 Answer:**

|  |
| --- |
| Algorithm inPlacePartition**(**S**,** lo**,** hi**)**  Input**:** Sequence S and ranks lo and hi**,** 0 **<=** lo**,**hi **<** S**.**size**()**  Output**:** Skip duplicate keys in the next partition    p **<--** a random integer between lo and hi  S**.**swapElements**(**S**.**atRank**(** lo **),** S**.**atRank**(** p **))**  pivot **<--** S**.**elemAtRank**(**lo**)**  j **<--** lo **+** 1  k **<--** hi  **while** j **<** k **do**  **while** k **>=** j **^** S**.**elemAtRank**(**k**)** **>** pivot **do**  k **<--** k 1  **while** j **<=** k **^** S**.**elemAtRank**(**j**)** **<** pivot **do**  j **<--** j **+** 1  **if** j **<** k then  S**.**swapElements**(**S**.**atRank**(** j **),** S**.**atRank**(** k **))**  S**.**swapElements**(**S**.**atRank**(** lo **),** S**.**atRank**(** k **))** **{**move pivot to sorted rank**}**  **return** k |

**C-4.19 Answer**

|  |  |
| --- | --- |
| Algorithm countInversion**(**S**,** C**)**  Input **:** sequence S with total order n elements**,** comparator C  Output**:** number of Inversion    **if** S**.**size**()** **>** 1 then  **(**S1**,** S2**)<-**partition**(**S**,** n**/**2**)**  countInversion**(**S1**,** C**)**  countInversion**(**S2**,** C**)**  **(**S**,**cnt**)** **<-**merge**(**S1**,** S2**,** C**)**  count **<-** count **+** cnt  **return** count | Algorithm merge**(**A**,** B**,** C**)**  Input**:** sequences A and B with n**/**2 elements each**,** comparator C  Output**:** count of number of inversion  count**<-**0  S **<-** empty sequence  **while** **!**A**.**isEmpty**()** **^** **!**B**.**isEmpty**()** **do**  **if** C**.**isLessThan**(** B**.**first**().**element**(),** A**.**first**().**element**()** **)** then  S**.**insertLast**(**B**.**remove**(**B**.**first**()))**  count **<-** count **+** 1  **else**  S**.**insertLast**(**A**.**remove**(**A**.**first**()))**  **while** **!**A**.**isEmpty**()** **do**  S**.**insertLast**(**A**.**remove**(**A**.**first**()))**  **while** **!**B**.**isEmpty**()** **do**  S**.**insertLast**(**B**.**remove**(**B**.**first**()))**  **return** count**,**S |

**C-4.25 Answer:**

|  |
| --- |
| Algorithm nutsBoltsMatchup**(**A**,** B**)**  Input **:** Sequence A of nuts**,** sequence B of bolts  Output **:** Matched set of nuts and bolts    T **<-** insertIntoRedBlackTree**(**B**)**  PQ **<-** **new** Priority Queue Array  **for** each x of A **do**  PQ**.**insert**(**x**,** PQ**.**remove**(**x**))**  **return** PQ |