R-2.7 **Answer:**

|  |
| --- |
| Algorithm root**()**  Output**:** position of the root of the tree T in S  **return** 1 |
| Algorithm parent**(**p**)**  Input **:** a position p of a node in the vector S  Output**:** position of the root of the tree T in S  **return** p **/** 2 |
| Algorithm leftChild**(**p**)**  Input **:** a position p of a node in the vector S  Output**:**position of the leftChild of the tree T in S  **return** 2 **\*** p |
| Algorithm rightChild**(**p**)**  Input **:** a position p of a node in the vector S  Output**:**position of the leftChild of the tree T in S  **return** 2 **\*** p **+** 1 |
| Algorithm isInternal**(**p**)**  Input **:** a position p of a node in the vector S  Output**:**position of the leftChild of the tree T in S  **if** 2 **\*** p **<=** S**.**size**()** or **(**2 **\*** p **+** 1**)** **<=** S**.**size**()**  **return** **true**  **return** **false** |
| Algorithm isExternal**(**p**)**  Input **:** a position p of a node in the vector S  Output**:**position of the leftChild of the tree T in S  **if** 2 **\*** p **>** S**.**size**()** **^** **(**2 **\*** p **+** 1**)** **>** S**.**size**()**  **return** **true**  **return** **false** |

R-2.8 **Answer:**

|  |  |
| --- | --- |
| **A** | **C:\Users\Jainal Uddin\Downloads\Shareit\Photo\20150302_014624 (2).jpg** |
| **B** | Minimum number of external nodes for binary tree with height = h + 1.  Justification: A binary tree can have minimum number of external nodes if the tree is drawn in a way that in each level at least one child is internal node.  In below tree, no of external nodes = 6,height = 5 which proofs min external node = h + 1  C:\Users\Jainal Uddin\Downloads\Shareit\Photo\20150302_004155 (2).jpg |
| **C** | Maximum number of external node = 2h  We know that external nodes don’t have child nodes. So, the maximum number of externals node we can get when all the nodes in the same depth. |
| **D** | External node, e = i – 1, i=internal node  Total node, n = e + i  =>n = 2e – 1  =>e = (n + 1) / 2  Since, e <= 2h  => (n+1)/2< 2h  => log (n+1)/2 < log 2h  => log (n+1) - log 2 < h log 2  => log (n+1) - log 2 < h log 2  => log (n+1) - 1 < h [log 2 = 1]  Again, h+1 < e  => h+1 < (n+1)/2  => h < (n+1)/2 - 1  => h < (n-1)/2  Combining both equation we get, log (n+1) -1 < h < (n-1)/2 |
| **E** | For n=1 and h=0, lower and upper bounds on h be attained with equality.  log(1+1) -1 ≤ h ≤ (1-1)/2  =>0≤ h ≤ 0 |

C-2.2 **Answer:**

|  |
| --- |
| As push, pop and size methods support constant time,  Running time of the dequeue and enqueue = n + n \* k.  T(n) = O(n).  Amortized running time = T(n) / n = 1 |

C-2.7 **Answer:**

|  |
| --- |
| Algorithm putSequenceInRandomOrder**(**S**)**  Input:Sequence S with n elements  Output: S in random order  r **<-** n  **while** r **>** 0 **do**  rand **<-** randomInt**(**r**)**  S**.**swapElements**(**S**.**atRank**(**r**),**S**.**atRank**(**rand**))**  r **<-** r **-** 1  **return** S |
| Running Time, for array based implementation  L1: O(1)  L2: O(n)  L3: O(n)  L4: O(n)  L5: O(n)  L6: O(1)  Total = O(n) |
| Running time, for inked list based implementation  L1: O(1)  L2: O(n)  L3: O(n)  L4: O(n2)=>for finding rank, it takes O(n) time.  L4: O(n)  L5: O(1)  Total = O(n2) |