Q) Design & implement an SLList method, secondLast() returns second_last element of SLList.

```
if head == NULL || head.next == NULL
    return -1
secondLast = head
last = head.next
while last.next is not null
    secondLast = secondLast.next
    last = last.next
return secondLast.next
```

Q) Why does get_node(i) of DLList has faster $\min(i, n-i)$ in time complexity $O(1, \min\{i, n-i\})$?

DLIST can either use next or prev ptr to start traversal in forward/backward from dummy node depending on which is nearest to dummy node

Q) Prove that binary tree having $n \geq 1$ nodes has $n-1$ edges.

Base Case: for $n=1$ there are zero edges hence $1-1=0$.

Inductive Step: Assume statement is true for binary tree with k nodes, which has $k-1$ edges. When adding one more node to tree we add one more edge. So $k+1$ nodes will have $k+1-1=k$ edges

Q) Prove that BT has $n \geq 1$ real nodes has $n+1$ external nodes.

Let n be no of real nodes in BT. Each real node can have 0, 1, or 2 children. Let E represent no of external nodes. In BT we have $n-1$ edges and since each edge connects two nodes $2(n-1)$. However each node connect to it's parent with one edge, $n-1$ edges.

$$2(n-1) - (n-1) + 1 = n+1$$

Q) DLList method is_palindrome() that returns TRUE.

```
bool is_palindrome(Node* left){
  if left == null; return true;
  Node* right = left;
  while (right->next != NULL)
    right = right->next;
  while (left != right){
    if (left->data != right->data)
      return false;
    if (left->next == right){
      break;
    left = left->next;
    right = right->next;
  }
  return true;
}
```

* max height of BTree:- $\lfloor \log_d ((n+1)/2) \rfloor$

	Root	Non Root
Min Keys	1	$\lceil m/2 \rceil - 1$
Min Empty Str	2	$\lceil m/2 \rceil$
Max Key	$m-1$	$m-1$
Max Empty Subtree	m	m

Q) Why BTree? → reduces no of reads → easily optimized to adjust size. → handle bulky data

MULTIMAP: $O(n)$ best, $O(n^2)$ worst

Q) What is max no of keys in BTree of order m of height h?

→ at Root 1 node with max $m-1$ keys
→ next level $m(m-1)$ keys
→ level 2 $m^2(m-1)$ keys

Total Keys $= (m-1) \times (1 + m + m^2 + \dots + m^h)$

$$= (m-1) \times \left(\frac{m^{h+1}-1}{m-1}\right)$$

$$= m^{h+1} - 1$$

Q) Describe an algorithm, relying on BT operation that count the no of leaves in BT that are left child of parent.

```
countleftLeaves(node)
  if node is null
    return 0
  leftcount = 0
  if node.left is not null and node.left.left is null and node.left.right is null
    leftcount = 1
  return leftCount + countleftLeaves(node.left) + countleftLeaves(node.right)
```

Height BTree $h = \lfloor n/2 \rfloor$

→ Since n-way multiway tree has $n+1$ external nodes

$$2^h \leq n+1 \leq 4^h$$

$$h \leq \log_2(n+1) \leq 2h$$

→ $O(\log n)$ (height of BTree)

Q) What is max no of nodes in multiway tree of height h?

→ multiway tree each node can have more than two children the max nodes of height h depends on BF. Let's denote BF as b, for height h

max nodes $= 1 + b + b^2 + \dots + b^h$

$$= \frac{1(1-b^{h+1})}{1-b}$$

Q) How many keys can BTree of order m and height h hold?

→ in BTree each node can hold $m-1$ keys and m children. The height h is max edges.

Max keys $= (m-1) + (m-1)^2 + \dots + (m-1)^h$

$(m-1)(h-1)$

Q) Design and implement permute(a) method that takes input an array, a, that contains n distinct values and randomly permutes n.

```
def permute(a)
    n = len(a)
    for i in range(n-1, 0, -1):
        j = random.randint(0, i)
        a[i], a[j] = a[j], a[i]
    return a
```

Q) Show how to find the second smallest value in Binary heap or meldable heap.

```
findsecondsmallest(heap)
  if heap is empty
    return none
  root = heap.root
  if root.left is NULL
    return root.right.key
  " " left
  return min(root.left.key, root.right.key)
```

BINARY HEAP
→ Root Array[0]
→ Left Array[2i+1] → Right Array[2(i+1)]
→ Parent Array[(i-1) div 2]

* Probability of empty cell = 1/M
* " " following cluster = size of cluster + 1 / M
* Chaining can tolerate load factor above 1 because it stores collisions in linked list. At each slot
* Open addressing cannot tolerate as each entry must occupy slot within the table

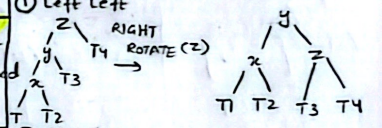
* BST Deletion (3) deleting internal node replace with inorder successor (RL) or predecessor (LR).

* TREAP
→ after inserting
→ fix heap property → do rotations.

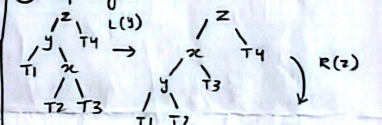
DELETION
→ when deleting internal node replace priority with $(-\infty$ MAX) or $(+\infty$ MIN).
→ if node has two children rotate it with child that has larger priority.

AVLTREE

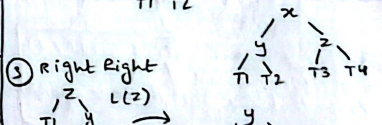
(1) Left Left



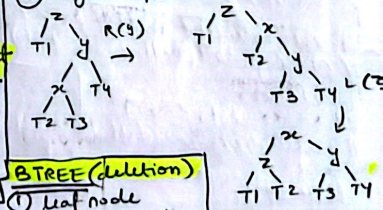
(2) Left Right



(3) Right Right



(4) Right Left Case



BTREE (deletion)

(1) leaf node
i) with more than min key (delete it)
ii) with min key (try to borrow a key from sibling if they have more (ROTATION)
iii) else merge parent b/w both left right

(2) internal node
i) delete
ii) move min key replace with inorder predecessor/successor.
iii) else merge siblings

(iv) node to be deleted has less than min keys then merge with sibling.

D) Write function that checks whether a Binary Tree is perfectly balanced.

```
def perfect_balance(root):
    if not root:
        return true;
    def check(node):
        if not node:
            return (True, 0)
        left_b, left_d = check(node.left)
        right_b, right_d = check(node.right)
        is_balance = (left_b and right_b and
            left_d = right_d and ((node.left and
            node.right or (not node.left and
            not node.right)))
        return (is_balance, left_d+1)
    return check(root)[0]
```

* preorder and inorder traversal generate same sequence for degenerated tree.

Finding an element in skiplist

```
Node* findPredNode(T, x) {
    Node * u = sentinel;
    while (r>=0) {
        while (u->next[r] != NULL)
            && compare(u->next[r] -> x, x) < 0)
            u = u->next[r] //go R
        r--; //go down
    }
    return u.
```

Array Stack — add/remove O(n-i)
ArrayDeque — O(min{i, n-i})

ANALYSIS OF SKIPLIST HEIGHT

$I_r = \begin{cases} 0 & \text{if } L_r \text{ is empty} \\ 1 & \text{if } L_r \text{ is non empty} \end{cases}$

$h = \sum_{r=1}^{\infty} I_r$

$E[h] = E\left[\sum_{r=1}^{\infty} I_r\right]$

$= \sum_{r=1}^{\infty} E[I_r]$

$= \sum_{r=1}^{\log n} E[I_r] + \sum_{r=\log n+1}^{\infty} E[I_r]$

$\leq \sum_{r=1}^{\log n} 1 + \sum_{r=\log n+1}^{\infty} n/2^r$

$\leq \log n + \sum_{r=0}^{\infty} 1/2^r$

$= \log n + 2$

ANALYSIS OF SKIPLIST SEARCH

worst case of skiplist O(n)

$E[S] = E\left[h + \sum_{r=0}^{\infty} S_r\right]$

$= E[h] + \sum_{r=0}^{\infty} E[S_r]$

$= E[h] + \sum_{r=0}^{\log n} E[S_r] + \sum_{r=\log n+1}^{\infty} E[S_r]$

$\leq E[h] + \sum_{r=0}^{\log n} 1 + \sum_{r=\log n+1}^{\infty} n/2^r$

$\leq E[h] + \log n + 3$

$= 2\log n + 5$

$IDF = \log\left(\frac{\text{No of sentences}}{\text{No of sentences containing words}}\right)$

$TF = \frac{\text{No of rep word in sentence}}{\text{No of word in sentence}}$

$TF * IDF$

Q) Implement a BST method getLE(x) that returns a list of all item in tree that are less than or equal to x.

```
method getLE(x) {
    list = new List()
    inOrderTraversal(root, x, list)
    return list
}
```

(DLL)

```
void deleteSecondLast() {
    if (tail==nullptr || head==tail)
    {
        return;
    }
    if(tail->prev==head){
        Node* second_last = head;
        head = tail
        head->prev = nullptr
        delete second_last
        return
    }
    Node* second_last = tail->prev
    Node* third_last = second_last->prev
    third_last->next = tail;
    tail->prev = third_last
    delete second_last
```

```
def reverseLL():
    if head is None or head.next is None
        return head
    prev = None
    current = head
    next_node = current.next
    while current is not None:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node
    return prev
```