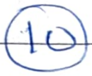
Aniket Gupta
2022073

59 PAGE NO.:
DATE: / /

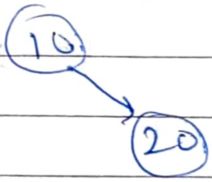
DSA HW-7

Q4. 10, 20, 30, 24, 27, 22, 6, 4, 23, 1

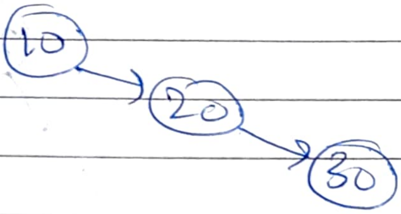
①



②

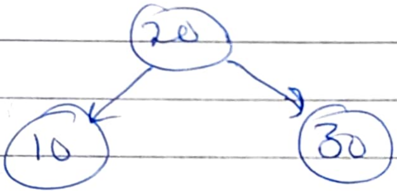


③



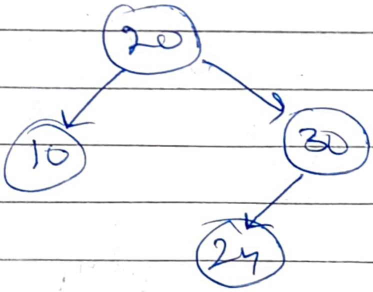
needs RR rotation

④

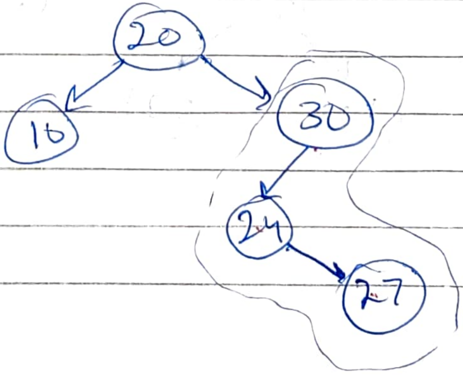


after RR rotation

⑤

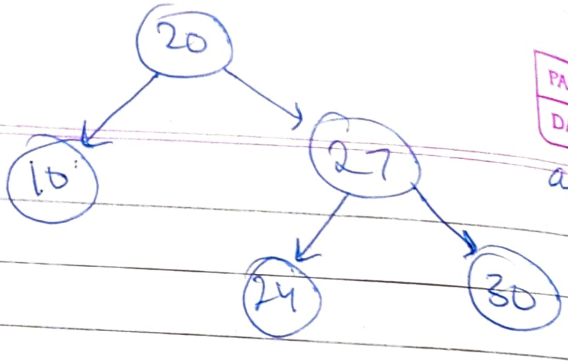


⑥



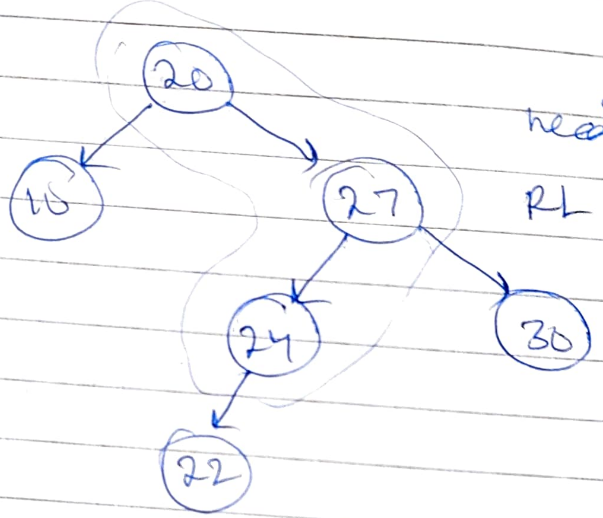
needs LR rotation

7



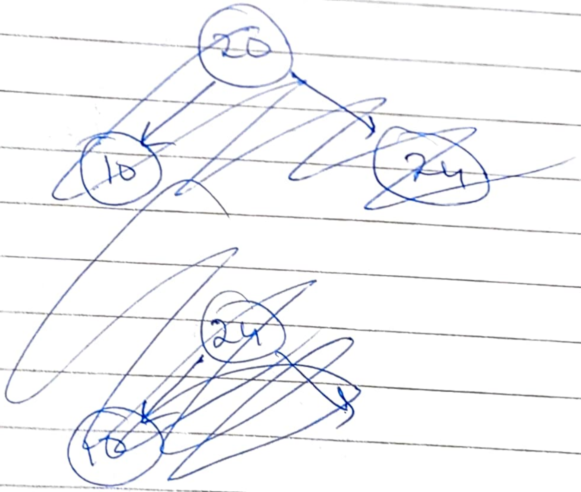
after rota

8

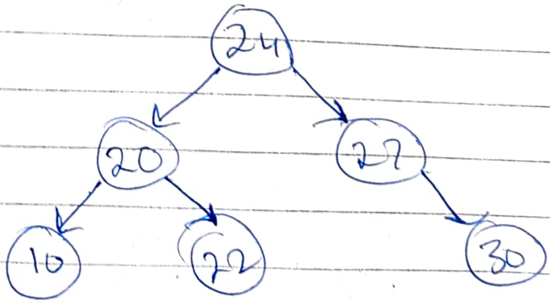


needs RL rotation

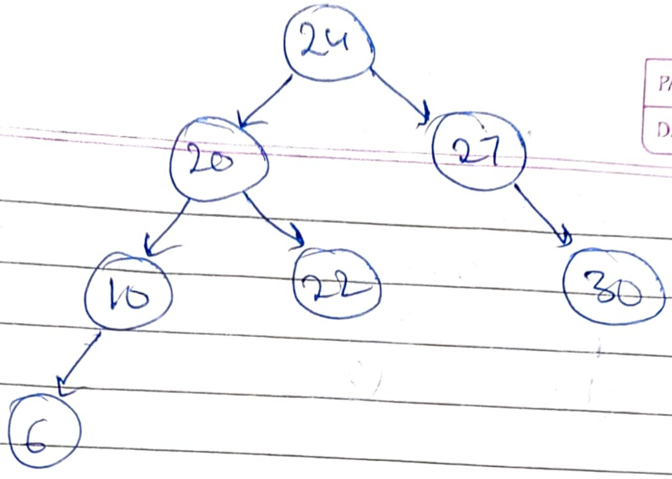
~~9~~



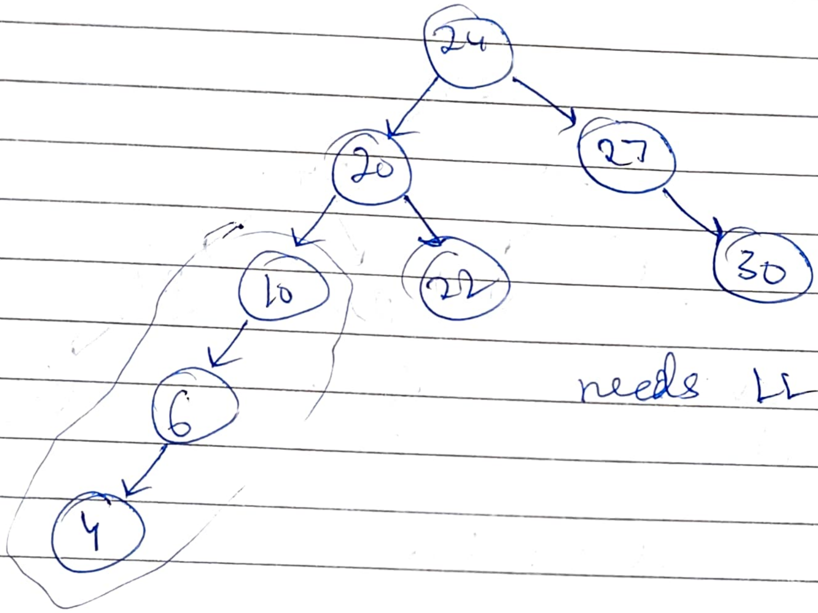
9



10

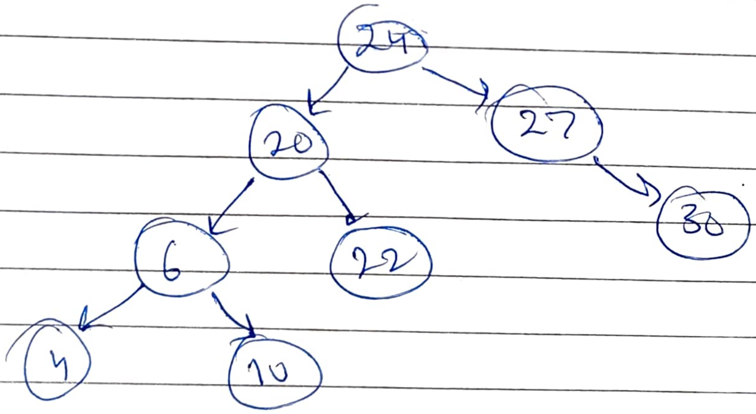


11

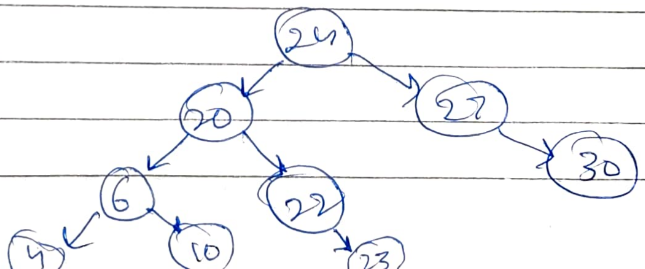


needs LL rotation

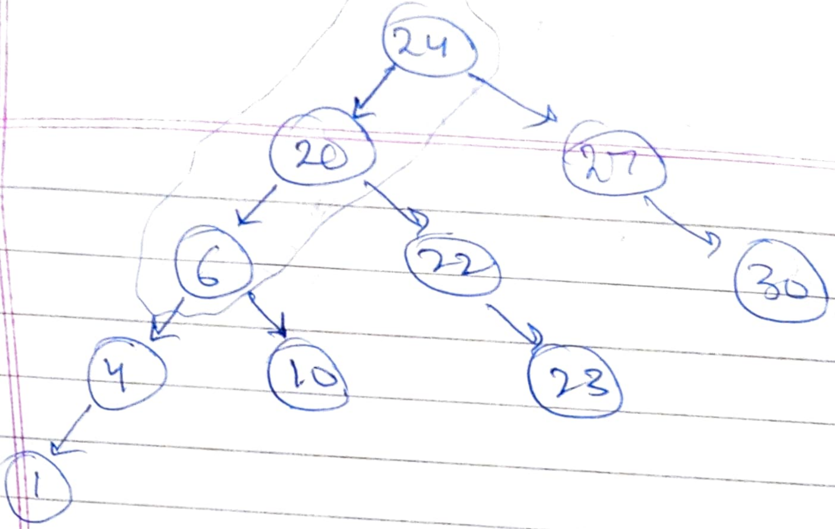
12



13

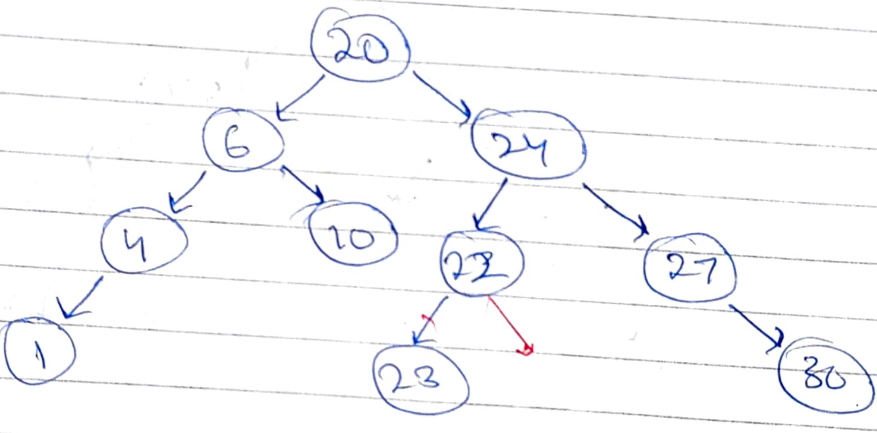


(14)



needs LL rotation

(15)



09

final AVL tree

let $f(h)$ represent the minimum no. of nodes in an AVL tree of height 'h'

if height of tree is h, height of one subtree is h-1 and height of other subtree must be h-1 or h-2 because of the height balance property

for min. nodes, we take it to be h-2

$\therefore$ $f(h) = f(h-1) + f(h-2) + 1$ where $f(0) = 1$ and $f(1) = 2$

$\Rightarrow$ $f(2) = f(1) + f(0) + 1 = 4$

$\Rightarrow$ $f(3) = f(2) + f(1) + 1 = 7$

$f(4) = f(3) + f(2) + 1 = 12$

$f(5) = f(4) + f(3) + 1 = 20$

~~$f(6) = f(5) + f(4)$~~ $f(6) = 33$

$f(7) = 54$

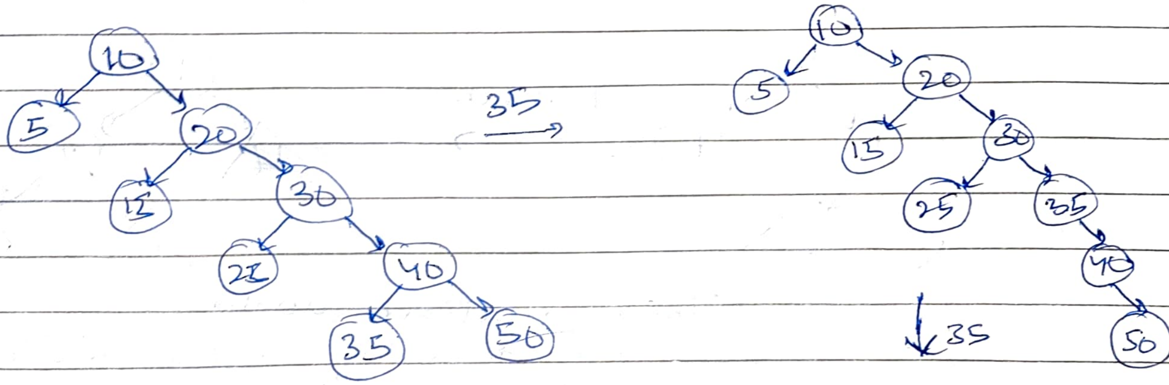
$f(8) = 88$

$f(9) = 143$

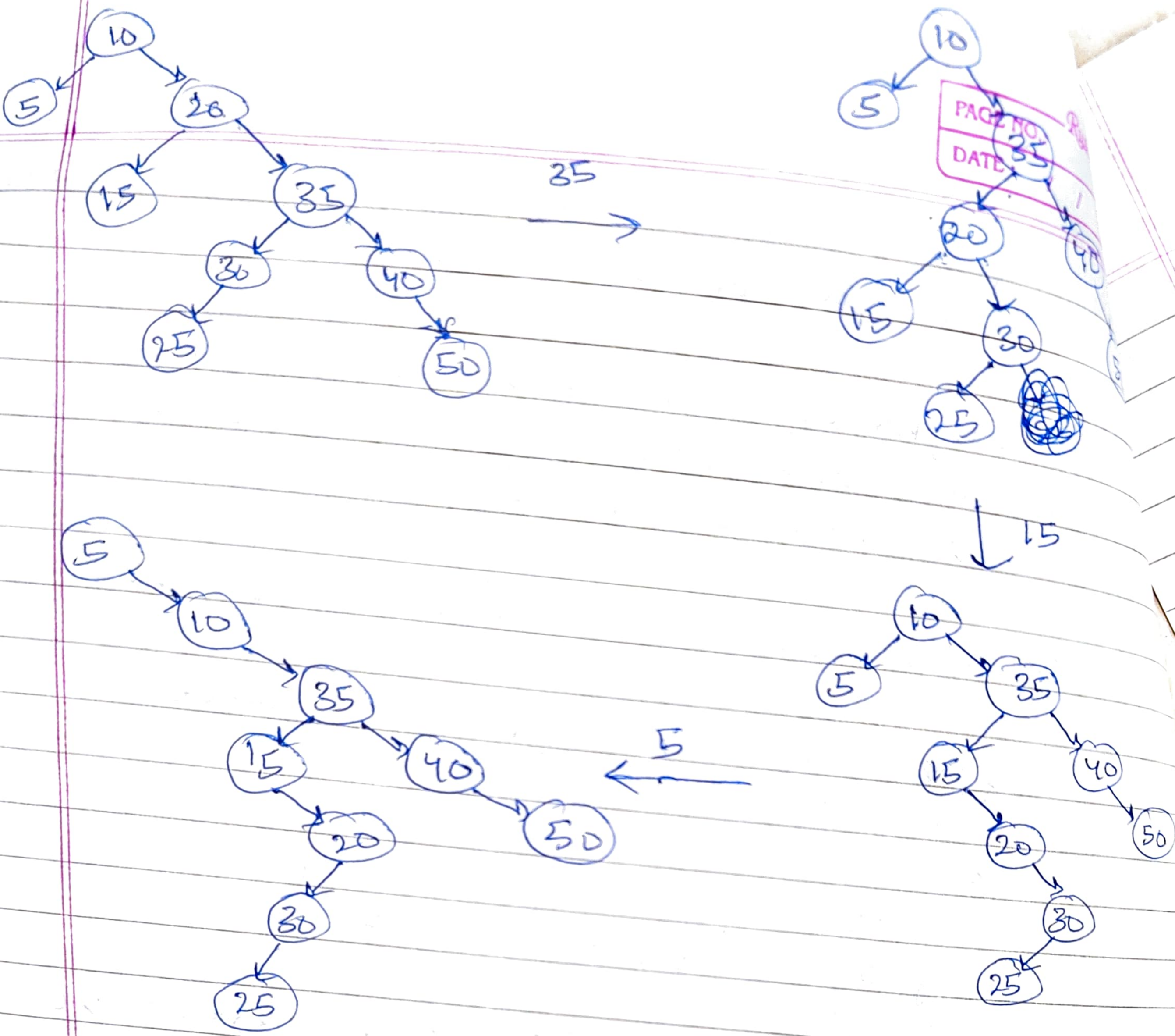
$\boxed{f(10) = 282}$

5

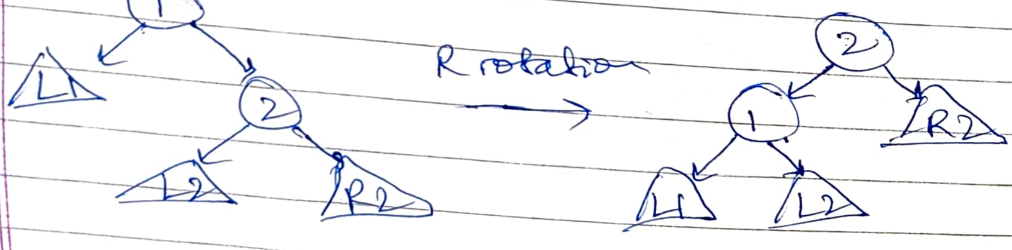
Q5. Quicksearch implementation

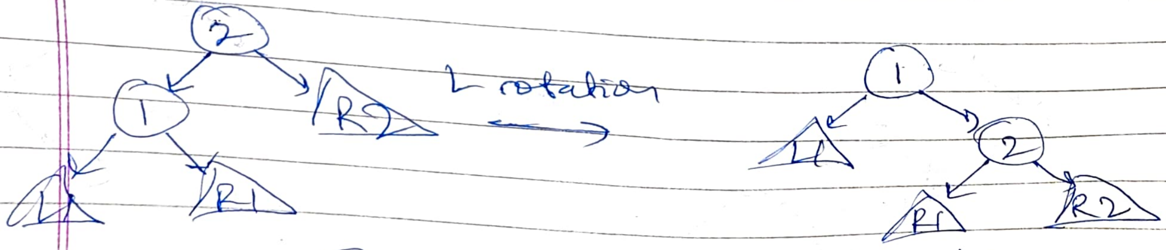


P.T.O



Rotations :





These rotations would help us in designing the ~~QuickSelect~~ QuickSearch Algorithm since they level up a node

P.T.O

QuickSearch algorithm :

```
Struct Tree {
    int val;
    struct Tree* left;
    struct Tree* right;
};

struct return_type {
    struct Tree* address;
    struct Tree* root;
}

// Functions to perform rotations

struct Tree* L(struct Tree* n) { // for levelling up
                                 //  a node which is
                                 //  on left of its
                                 //  parent
    struct Tree * new_root = n->Left;
    n->left = new_root -> right;
    new_root -> right = n;
    return new_root;
}

struct Tree* R(struct Tree* n) { // for levelling up a
                                 //  node which is on
                                 //  right of its parent
    struct Tree* new_root = n->right;
    n->right = new_root-> left;
    new_root -> left = n;
    return new_root;
```

P.T.O

```
struct return_type QuickSearch(struct Tree* root, int key)
    struct return_type res;

    if (root == NULL) {
        res.address = NULL;
        res.root = NULL;
        return res;
    }

    ~~if (root -> left != NULL && root -> le~~
    if (root -> left != NULL && root -> left -> val == key) {
        res.address = L(root);
        res.root = res.address;
        return res;
    }

    if (root -> right != NULL && root -> right -> val == key) {
        res.address = R(root);
        res.root = res.address;
        return res;
    }

    if (root -> val > key) {
        res = QuickSearch(root -> left, key);
        if (res.address != NULL) {
            root -> left = res.root;
            res.root = root;
        }
        return res;
    }
```

P.T.O

```
        if (root→val < key) {
            res = QuickSearch(root→right, key);
            if (res.address != NULL) {
                root→right = res.root;
                res.root = root;
            }
            return res;
        }

        res.address = root;
        res.root = root;
        return rel;
    }
```

```
Q1.   // for BST
      int lca(struct node* root, int v1, int v2) {
          if (root == NULL)
              return -1;  // indicating there is no LCA

          if (v1 < root→val && v2 < root→val)
              return lca(root→left, v1, v2);

10        if (v1 > root→val && v2 > root→val)
              return lca(root→right, v1, v2);

          // for any other case
          return root→val;
      }
```

P.T.O

```
int lca(struct node* root, int v1, int v2) {
      if (root == NULL)
            return -1;

      if (root -> val == v1 || root -> val == v2)
            return root -> val;

      int left = lca(root -> left, v1, v2);
      int right = lca(root -> right, v1, v2);

      if (left == -1)
            return right;

      else if (right == -1)
            return left;

      else
            return root -> val;
}
```

struct?