Câu 1 :

A. A. Trình bày kết quả từng bước khi sắp xếp dãy số sau bằng thuật toán Bubble/ Selection/InsertionSort.5 10 6 12 19 2 9 4

Lời giải InsertionSort

5 10 6 12 19 2 9 4

5

5 10

5 6 10

5 6 10 12

5 6 10 12 19

2 5 6 10 12 19

2 5 6 9 10 12 19

2 4 5 6 9 10 12 19

Lời giải BubbleSort

5 10 6 12 19 2 9 4

5 6 10 12 2 9 4 19

5 6 10 2 9 4 12 19

5 6 2 9 4 10 12 19

5 2 6 4 9 10 12 19

2 5 4 6 9 10 12 19

2 4 5 6 9 10 12 19

Lời giải SelectionSort

5 10 6 12 19 2 9 4

2 10 6 12 19 5 9 4

2 4 6 12 19 5 9 10

2 4 5 12 19 6 9 10

2 4 5 6 19 12 9 10

2 4 5 6 9 12 19 10

2 4 5 6 9 10 19 12

2 4 5 6 9 10 12 19

B. Trình bày kết quả từng bước khi sắp xếp dãy số sau bằng thuật toán Quick Sort.14 15 18 10 12 15 18 13

Lời giải:

Pivot : 14

14 15 18 10 12 15 18 13

14 10 12 13 18 15 18 15

Pivot 13 Pivot 18

13 10 12 14 18 15 18 15

12 10 13 14 15 15 18 18

Pivot 12 Pivot 15

10 12 13 14 15 15 18 18

Pivot 12 Pivot 14 Pivot 15 Pivot 18

10 12 13 14 15 15 18 18

10 12 13 14 15 15 18 18

**Câu 2:** Danh sách liên kết đơn

1. Khai báo class Sinh viên gồm có: mã số sinh viên (mssv), họ tên (hoten), khoa quản lý (khoaql). Khai báo danh sách liên kết đơn chứa các sinh viên.
2. Viết phương thức kiểm tra một mssv có trong danh sách liên kết hay không, nếu có trả về true, còn không trả về false.
3. Viết phương thức thêm một sinh viên vào đuôi danh sách liên kết, có kiểm tra trùng mssv, nếu trùng thì không được thêm.

// Câu A

public class SV {

private String MSSV, HoTen, Khoaql;

public SV() {

}

public SV(String ms, String hoten, String khoaql) {

MSSV = ms;

HoTen = hoten;

Khoaql = khoaql;

}

public String getMSSV() {

return MSSV;

}

public String getHoTen() {

return HoTen;

}

public String getKhoaql() {

return Khoaql;

}

}

class linkSV {

public SV data;

public linkSV next;

public linkSV(SV d)

{

data = d;

}

}

class LinkedListSv {

private linkSV first, pos, last;

private int count;

public LinkedListSv () {

first = pos = last = null;

count = 0;

}

public boolean isEmpty()

{

return first == null;

}

private void insert(SV d, linkSV p)

{

linkSV newLink = new linkSV(d);

if (p == null)

{

newLink.next = first;

first = newLink;

}

else

{

newLink.next = p.next;

p.next = newLink;

}

if (newLink.next == null)

last = newLink;

count++;

}

public void insertFirst(SV d) {

insert(d, null);

}

public void insertPos(SV d) {

insert(d, pos);

}

public void insertLast(SV d) {

insert(d, last);

}

private void delete(linkSV p) {

linkSV t;

if (p==null)

{

t = first;

first = t.next;

}

else

{

t = p.next;

p.next = t.next;

}

if (t.next == null) last = p;

count--;

}

public void deleteFirts () {

delete(null);

}

public void deleteFPos () {

delete(pos);

}

public boolean searchList(SV d) {

linkSV c;

c = first;

pos = null;

while (c!=null && c.data.getMSSV().compareTo(d.getMSSV()) != 0)

{

pos = c;

c = c.next;

}

return c!=null;

}

public boolean searchOrderList(SV d) {

linkSV c;

c = first;

pos = null;

while (c!=null && c.data.getMSSV().compareTo(d.getMSSV())<0)

{

pos = c;

c = c.next;

}

if (c!=null && c.data.getMSSV().compareTo(d.getMSSV())>0)

return false;

return c!=null;

}

public void startList() {

pos = first;

}

public void nextLink() {

pos = pos.next;

}

public boolean endList() {

return pos == null;

}

public SV getData() {

return pos.data;

}

public int nItem() {

return count;

}

// Câu B

public boolean chuaMSSV(String mssv) {

linkSV hienTai = first;

while (hienTai != null) {

if (hienTai.data.getMSSV().equals(mssv)) {

return true;

}

hienTai = hienTai.next;

}

return false;

}

// Câu C

public void themSinhVienVaoDuoi(SV sinhVien) {

if (!chuaMSSV(sinhVien.getMSSV())) {

linkSV newLink = new linkSV(sinhVien);

if (isEmpty()) {

first = last = newLink;

} else {

last.next = newLink;

last = newLink;

}

count++;

} else {

System.out.println("Sinh viên với MSSV " + sinhVien.getMSSV() + " đã tồn tại trong danh sách.");

}

}

}

**Câu 3:** Cây nhị phân tìm kiếm

1. Khai báo node và cây nhị phân tìm kiếm có thuộc tính key là số nguyên và name là tên người, cùng với các phương thức khởi tạo.
2. Vẽ cây nhị phân tìm kiếm được tạo với lần lượt các key sau:   
   18, 23, 45, 10, 9, 3, 5, 11, 19, 34, 40, 2.
3. Duyệt cây trên theo tiền tự (NLR), hậu tự (LRN).
4. Viết phương thức duyệt cây theo inorder (xuất ra key và name).
5. Viết phương thức đếm số node có key <= số nguyên **x**.
6. Viết phương thức kiểm tra cây có phải là cây nhị phân đầy đủ hay không.

// Câu A

import java.util.Scanner;

public class CayNhiPhan {

class Node {

public int key;

public String name;

public Node left, right;

public Node(int key, String name) {

this.key = key;

this.name = name;

left = right = null;

}

}

class Tree {

Scanner sc = new Scanner(System.in);

private Node root;

public Tree() {

root = null;

}

public boolean isEmpty() {

return root == null;

}

public int getKey() {

return root.key;

}

public void setKey(int k) {

root.key = k;

}

public Tree getLeft() {

Tree t = new Tree();

t.root = root.left;

return t;

}

public Tree getRight() {

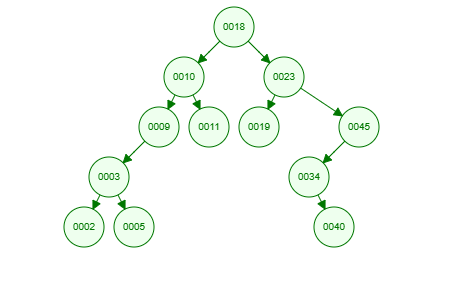
Tree t = new Tree();

t.root = root.right;

return t;

}

// Câu B



// Câu C

18, 23, 45, 10, 9, 3, 5, 11, 19, 34, 40, 2

Duyệt cây trên theo (NLR) : 18,10,9,3,2,5,11,23,19,45,34,40

Duyệt cây trên theo (LNR) : 2 3 5 9 10 11 18 19 23 34 40 45

Duyệt cây trên theo (LRN) : 2 5 3 9 11 10 40 34 19 45 23 18

public void preOrder (Node root) {

if (root != null) {

System.out.print(root.key + " ");

preOrder (root.left);

preOrder (root.right);

}

}

public void postOrder (Node root) {

if (root != null) {

postOrder (root.left);

postOrder (root.right);

System.out.print(root.key + " ");

}

}

public void inOrder (Node root) {

if (root != null) {

inOrder(root.left);

System.out.print(root.key + " ");

inOrder (root.right);

}

}

// Câu D

public void inOrder (Node root) {

if (root != null) {

inOrder (root.left);

System.out.println("Key: " + root.key + ", Name: " + root.name);

inOrder (root.right);

}

}

// Câu E

public int countNodesLessThanOrEqual(Node root, int x) {

if (root == null) {

return 0;

}

int count = 0;

if (root.key <= x) {

count++;

}

count += countNodesLessThanOrEqual(root.left, x);

count += countNodesLessThanOrEqual(root.right, x);

return count;

}

// Câu F

public boolean isFullBinaryTree(Node root) {

if (root == null) {

return true;

}

if (root.left == null && root.right == null) {

return true;

}

if (root.left != null && root.right != null) {

return isFullBinaryTree(root.left) && isFullBinaryTree(root.right);

}

return false;

}

// Thêm phương thức insert để thêm node vào cây

public void insert(int key, String name) {

root = insertRecursive(root, key, name);

}

private Node insertRecursive(Node root, int key, String name) {

if (root == null) {

return new Node(key, name);

}

if (key < root.key) {

root.left = insertRecursive(root.left, key, name);

} else if (key > root.key) {

root.right = insertRecursive(root.right, key, name);

}

return root;

}

// Thêm phương thức getRoot để truy cập root từ bên ngoài

public Node getRoot() {

return root;

}

}

}