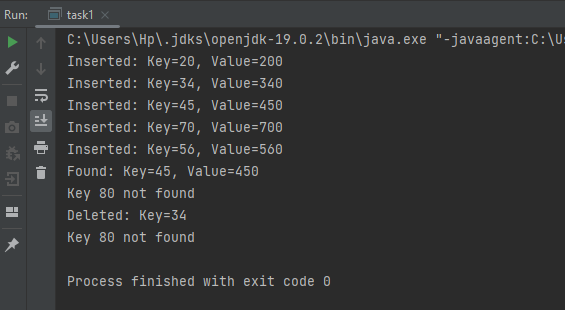
Lab 11

Task1:

class HashNode {  
 int key;  
 int value;  
 HashNode next;  
  
 public HashNode(int key, int value) {  
 this.key = key;  
 this.value = value;  
 this.next = null;  
 }  
}  
  
class HashMap {  
 private static final int *tablesize* = 10;  
 private HashNode[] table;  
  
 public HashMap() {  
 table = new HashNode[*tablesize*];  
 for (int i = 0; i < *tablesize*; i++) {  
 table[i] = null;  
 }  
 }  
  
 private int hashFunction(int key) {  
 return key % *tablesize*;  
 }  
  
 public void delete(int value) {  
 int index = hashFunction(value);  
 HashNode curr = table[index];  
 HashNode prev = null;  
  
 while (curr != null && curr.key != value) {  
 prev = curr;  
 curr = curr.next;  
 }  
  
 if (curr == null) {  
 System.*out*.println("Key " + value + " not found");  
 return;  
 }  
  
 if (prev == null) {  
 table[index] = curr.next;  
 } else {  
 prev.next = curr.next;  
 }  
  
 System.*out*.println("Deleted: Key=" + value);  
 }  
  
 public int find(int key) {  
 int index = hashFunction(key);  
 HashNode curr = table[index];  
  
 while (curr != null) {  
 if (curr.key == key) {  
 System.*out*.println("Found: Key=" + key + ", Value=" + curr.value);  
 return curr.value;  
 }  
 curr = curr.next;  
 }  
  
 System.*out*.println("Key " + key + " not found");  
 return -1;  
 }  
  
  
 public void insert(int key, int value) {  
 int index = hashFunction(key);  
 HashNode newNode = new HashNode(key, value);  
  
 if (table[index] == null) {  
 table[index] = newNode;  
 } else {  
 HashNode curr = table[index];  
 while (curr.next != null) {  
 curr = curr.next;  
 }  
 curr.next = newNode;  
 }  
  
 System.*out*.println("Inserted: Key=" + key + ", Value=" + value);  
 }  
}  
public class task1 {  
 public static void main(String[] args) {  
 HashMap hashMap = new HashMap();  
  
 hashMap.insert(20, 200);  
 hashMap.insert(34, 340);  
 hashMap.insert(45, 450);  
 hashMap.insert(70, 700);  
 hashMap.insert(56, 560);  
  
 hashMap.find(45);  
 hashMap.find(80);  
  
 hashMap.delete(34);  
 hashMap.delete(80);  
 }  
}



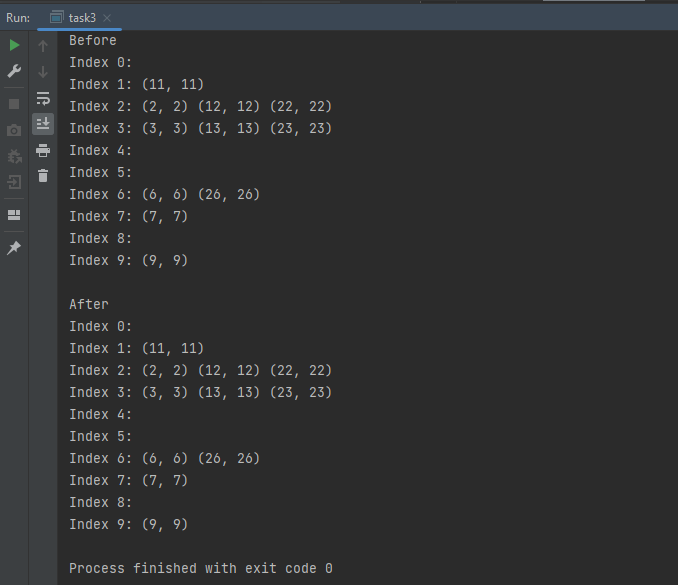
Task2:

class HashNode {  
 int key;  
 int f;  
 HashNode next;  
  
 public HashNode(int key, int frequency) {  
 this.key = key;  
 this.f = frequency;  
 this.next = null;  
 }  
}  
public class task2 {  
 public static void main(String[] args) {  
 int[] a = {1, 5, 7, 1};  
 int K = 6;  
  
 HashMap hashMap = new HashMap();  
 int count = 0;  
 for (int num : a) {  
 int complement = K - num;  
 int complementCount = hashMap.find(complement);  
  
 if (complementCount > 0) {  
 count += complementCount;  
 }  
  
 HashNode newNode = new HashNode(num, hashMap.find(num) + 1);  
 hashMap.inserting(newNode);  
 }  
  
 System.*out*.println("No of pairs with sum " + K + ": " + count);  
 }  
}  
  
  
class HashMap {  
 private static final int *tablesize* = 10;  
 private HashNode[] htable;  
  
 public HashMap() {  
 htable = new HashNode[*tablesize*];  
 for (int i = 0; i < *tablesize*; i++) {  
 htable[i] = null;  
 }  
 }  
  
 private int hashFunction(int key) {  
 return key % *tablesize*;  
 }  
  
  
 public int find(int value) {  
 int index = hashFunction(value);  
 HashNode curr = htable[index];  
  
 while (curr != null) {  
 if (curr.key == value) {  
 return curr.f;  
 }  
 curr = curr.next;  
 }  
  
 return 0;  
 }  
  
 public void inserting(HashNode m) {  
 int index = hashFunction(m.key);  
 if (htable[index] == null) {  
 htable[index] = m;  
 } else {  
 HashNode curr = htable[index];  
 while (curr.next != null) {  
 curr = curr.next;  
 }  
 curr.next = m;  
 }  
 }  
}



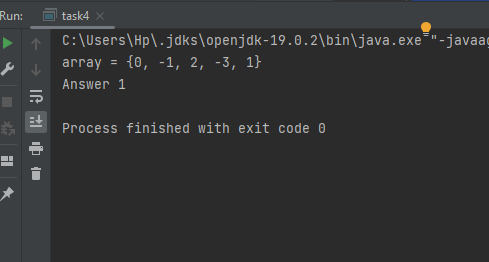
Task3:

class HashNode {  
 int key;  
 int value;  
 HashNode next;  
  
 public HashNode(int key, int value) {  
 this.key = key;  
 this.value = value;  
 this.next = null;  
 }  
}  
  
class HashMapWithChaining {  
 private static final int *tablesize* = 10;  
 private HashNode[] htable;  
  
   
  
 private int hashFunction(int key) {  
 return key % *tablesize*;  
 }  
 public HashMapWithChaining() {  
 htable = new HashNode[*tablesize*];  
 }  
   
  
 public void sortChain() {  
 for (int i = 0; i < *tablesize*; i++) {  
 if (htable[i] != null) {  
 htable[i] = sortChain(htable[i]);  
 }  
 }  
 }  
 public void print() {  
 for (int i = 0; i < *tablesize*; i++) {  
 HashNode curr = htable[i];  
 System.*out*.print("Index " + i + ": ");  
 while (curr != null) {  
 System.*out*.print("(" + curr.key + ", " + curr.value + ") ");  
 curr = curr.next;  
 }  
 System.*out*.println();  
 }  
 }  
 public void inserting(HashNode node) {  
 int index = hashFunction(node.key);  
  
 if (htable[index] == null) {  
 htable[index] = node;  
 } else {  
 HashNode curr = htable[index];  
 while (curr.next != null) {  
 curr = curr.next;  
 }  
 curr.next = node;  
 }  
 }  
  
 private HashNode sortChain(HashNode head) {  
 HashNode sortedHead = null;  
 HashNode curr = head;  
  
 while (curr != null) {  
 HashNode next = curr.next;  
  
 if (sortedHead == null || curr.key < sortedHead.key) {  
 curr.next = sortedHead;  
 sortedHead = curr;  
 } else {  
 HashNode sortedCurrent = sortedHead;  
 while (sortedCurrent.next != null && sortedCurrent.next.key < curr.key) {  
 sortedCurrent = sortedCurrent.next;  
 }  
 curr.next = sortedCurrent.next;  
 sortedCurrent.next = curr;  
 }  
  
 curr = next;  
 }  
  
 return sortedHead;  
 }  
  
   
}  
public class task3 {  
 public static void main(String[] args) {  
 int[] values = {3, 2, 9, 6, 11, 13, 7, 12, 23, 22, 26};  
 int m = 10;  
  
 HashMapWithChaining hashMap = new HashMapWithChaining();  
  
 for (int value : values) {  
 HashNode node = new HashNode(value, value);  
 hashMap.inserting(node);  
 }  
  
 System.*out*.println("(x, y) = (key, value) ");  
 System.*out*.println("Before");  
 hashMap.print();  
  
 hashMap.sortChain();  
  
 System.*out*.println("\nAfter ");  
 hashMap.print();  
 }  
}



Task4:

public class task4 {  
 public static void main(String[] args) {  
 int[] arr = {0, -1, 2, -3, 1};  
 int result = *Triplets*(arr);  
 System.*out*.println("array = {0, -1, 2, -3, 1}");  
 System.*out*.println("Answer " + result);  
 }  
 private static boolean doescontain(int[] a, int target, int start) {  
 for (int i = start; i < a.length; i++) {  
 if (a[i] == target) {  
 return true;  
 }  
 }  
 return false;  
 }  
 public static int Triplets(int[] a) {  
 int n = a.length;  
  
 for (int i = 0; i < n - 1; i++) {  
 for (int j = i + 1; j < n; j++) {  
 int x = -(a[i] + a[j]);  
 if (*doescontain*(a, x, j + 1)) {  
 return 1;  
 }  
 }  
 }  
  
 return 0;  
 }  
}



Task5:

import java.util.Arrays;  
  
public class task5{  
 public static void main(String[] args) {  
 char[] nuts = {'@', '%', '$', '#', '^'};  
 char[] bolts = {'%', '@', '#', '$', '^'};  
  
 *NutsAndBolts*(nuts, bolts);  
  
 System.*out*.println("Matching the Nuts and Bolts:");  
 System.*out*.println("nuts: " + Arrays.*toString*(nuts));  
 System.*out*.println("bolts: " + Arrays.*toString*(bolts));  
 }  
  
 private static final char[] *order* = {'!', '#', '$', '%', '&', '\*', '@', '^', '~'};  
  
   
  
 private static int[] BuildIndexMap(char[] a) {  
 int[] indexMap = new int[256];  
  
 for (int i = 0; i < a.length; i++) {  
 indexMap[a[i]] = i;  
 }  
  
 return indexMap;  
 }  
 private static int partitioning(char[] arr, int l, int h, char pivot, int[] indexMap) {  
 int i = l - 1;  
  
 for (int j = l; j < h; j++) {  
 if (*compare*(arr[j], pivot) < 0) {  
 i++;  
 *swapping*(arr, i, j);  
 } else if (*compare*(arr[j], pivot) == 0) {  
 *swapping*(arr, j, h);  
 j--;  
 }  
 }  
  
 i++;  
 *swapping*(arr, i, h);  
  
 if (indexMap != null) {  
 int pivotIndex = indexMap[pivot];  
 *swapping*(indexMap, i, pivotIndex);  
 }  
  
 return i;  
 }  
 public static void quicksort(char[] nuts, char[] bolts, int l, int h, int[] nutIndexMap) {  
 if (l < h) {  
 int pivotIndex = *partitioning*(bolts, l, h, nuts[h], nutIndexMap);  
 *partitioning*(nuts, l, h, bolts[pivotIndex], null);  
  
 *quicksort*(nuts, bolts, l, pivotIndex - 1, nutIndexMap);  
 *quicksort*(nuts, bolts, pivotIndex + 1, h, nutIndexMap);  
 }  
 }  
  
   
 public static void NutsAndBolts(char[] n, char[] b) {  
 if (n == null || b == null || n.length != b.length) {  
 throw new IllegalArgumentException("Invalid input");  
 }  
  
 int[] nutIndexMap = *BuildIndexMap*(n);  
 *quicksort*(n, b, 0, n.length - 1, nutIndexMap);  
 }  
 private static int indexof(char m) {  
 for (int i = 0; i < *order*.length; i++) {  
 if (*order*[i] == m) {  
 return i;  
 }  
 }  
 return -1;  
 }  
 private static void swapping(char[] arr, int i, int j) {  
 char temp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = temp;  
 }  
  
 private static int compare(char a, char b) {  
 return *indexof*(a) - *indexof*(b);  
 }  
 private static void swapping(int[] a, int i, int j) {  
 int temp = a[i];  
 a[i] = a[j];  
 a[j] = temp;  
 }  
  
   
  
  
  
  
}

