

Федеральное агентство связи  
Ордена Трудового Красного Знамени федеральное государственное  
бюджетное учреждение высшего образования  
«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и  
информационных технологий

Лабораторная работа №2  
по дисциплине: «Структуры и алгоритмы обработки данных»  
на тему: «Методы поиска»

Выполнил студент  
группы БФИ1902  
Гусев Н. С.  
Проверил:  
Мкртчян Г. М.

Москва, 2021 г.

## **Оглавление**

1. Цель работы.....	3
2. Задание на лабораторную работу .....	3
3. Листинг программы .....	3

## 1. Цель работы

Цель работы: рассмотреть работу различных методов поиска.

## 2. Задание на лабораторную работу

- 1) Реализовать следующие методы поиска: Бинарный поиск, Бинарное дерево, Фибоначчиев, Интерполяционный;
- 2) Реализовать следующие методы рехеширования: Простое рехеширование, рехеширование с помощью псевдослучайных чисел
- 3) Расставить на стандартной 64-клеточной шахматной доске 8 ферзей так, чтобы ни один из них не находился под боем другого». Подразумевается, что ферзь бьёт все клетки, расположенные по вертикалям, горизонталям и обеим диагоналям. Написать программу, которая находит хотя бы один способ решения задач.

## 3. Листинг программы

**BinarySearch.java:**

```
import java.util.Arrays;
import java.util.Scanner;

public class BinarySearch {
    public static void main(String args[]) {
        int counter, num, item, array[], first, last;

        //Создаем объект Scanner для считывания чисел, введенных
        //пользователем
        Scanner input = new Scanner(System.in);
        System.out.println("Введите количество элементов массива: ");
        num = input.nextInt();

        // Создаем массив введенного пользователем размера
        array = new int[num];

        System.out.println("Введите " + num + " чисел");

        //Заполняем массив, вводя элементы в консоль
        for (counter = 0; counter < num; counter++)
            array[counter] = input.nextInt();

        // сортируем элементы массива, так как для бинарного поиска
        // элементы массива должны быть отсортированными
        Arrays.sort(array);
        System.out.print("Массив отсортирован: ");
        System.out.println(Arrays.toString(array));

        System.out.println("Введите элемент для бинарного поиска: ");
        item = input.nextInt();
        first = 0;
        last = num - 1;

        // метод принимает начальный и последний индекс, а также число для
```

```

поиска
    binarySearch(array, first, last, item);
}

// бинарный поиск
public static void binarySearch(int[] array, int first, int last, int
item) {
    int position;
    int comparisonCount = 1;    // для подсчета количества сравнений

    // для начала найдем индекс среднего элемента массива
    position = (first + last) / 2;

    while ((array[position] != item) && (first <= last)) {
        comparisonCount++;
        if (array[position] > item) {    // если число заданного для поиска
            last = position - 1; // уменьшаем позицию на 1.
        } else {
            first = position + 1;    // иначе увеличиваем на 1
        }
        position = (first + last) / 2;
    }
    if (first <= last) {
        System.out.println(item + " является " + ++position + " элементом
в массиве");
        System.out.println("Метод бинарного поиска нашел число после " +
comparisonCount +
            " сравнений");
    } else {
        System.out.println("Элемент не найден в массиве. Метод бинарного
поиска закончил работу после "
            + comparisonCount + " сравнений");
    }
}
}

```

#### BinaryTree:

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Stack;

////////////////////////////////////
class Node {
    public int iData;           // data item (key)
    public double dData;        // data item
    public Node leftChild;      // this node's left child
    public Node rightChild;     // this node's right child

    public void displayNode()   // display ourself
    {
        System.out.print('{');
        System.out.print(iData);
        System.out.print(", ");
        System.out.print(dData);
        System.out.print("} ");
    }
} // end class Node

////////////////////////////////////
class Tree {
    private Node root;          // first node of tree

    // -----

```

```

public Tree() // constructor
{
    root = null;
} // no nodes in tree yet

// -----
public Node find(int key) // find node with given key
{
    // (assumes non-empty tree)
    Node current = root; // start at root
    while (current.iData != key) // while no match,
    {
        if (key < current.iData) // go left?
            current = current.leftChild;
        else // or go right?
            current = current.rightChild;
        if (current == null) // if no child,
            return null; // didn't find it
    }
    return current; // found it
} // end find()

// -----
public void insert(int id, double dd) {
    Node newNode = new Node(); // make new node
    newNode.iData = id; // insert data
    newNode.dData = dd;
    if (root == null) // no node in root
        root = newNode;
    else // root occupied
    {
        Node current = root; // start at root
        Node parent;
        while (true) // (exits internally)
        {
            parent = current;
            if (id < current.iData) // go left?
            {
                current = current.leftChild;
                if (current == null) // if end of the line,
                {
                    // insert on left
                    parent.leftChild = newNode;
                    return;
                }
            }
            // end if go left
            else // or go right?
            {
                current = current.rightChild;
                if (current == null) // if end of the line
                {
                    // insert on right
                    parent.rightChild = newNode;
                    return;
                }
            }
            // end else go right
        } // end while
    } // end else not root
} // end insert()

// -----
public boolean delete(int key) // delete node with given key
{
    // (assumes non-empty list)
    Node current = root;
    Node parent = root;
    boolean isLeftChild = true;

```

```

while (current.iData != key)           // search for node
{
    parent = current;
    if (key < current.iData)           // go left?
    {
        isLeftChild = true;
        current = current.leftChild;
    } else                             // or go right?
    {
        isLeftChild = false;
        current = current.rightChild;
    }
    if (current == null)               // end of the line,
        return false;                 // didn't find it
} // end while
// found node to delete

// if no children, simply delete it
if (current.leftChild == null &&
    current.rightChild == null) {
    if (current == root)               // if root,
        root = null;                 // tree is empty
    else if (isLeftChild)
        parent.leftChild = null;      // disconnect
    else                                // from parent
        parent.rightChild = null;
}

// if no right child, replace with left subtree
else if (current.rightChild == null)
    if (current == root)
        root = current.leftChild;
    else if (isLeftChild)
        parent.leftChild = current.leftChild;
    else
        parent.rightChild = current.leftChild;

// if no left child, replace with right subtree
else if (current.leftChild == null)
    if (current == root)
        root = current.rightChild;
    else if (isLeftChild)
        parent.leftChild = current.rightChild;
    else
        parent.rightChild = current.rightChild;

else // two children, so replace with inorder successor
{
    // get successor of node to delete (current)
    Node successor = getSuccessor(current);

    // connect parent of current to successor instead
    if (current == root)
        root = successor;
    else if (isLeftChild)
        parent.leftChild = successor;
    else
        parent.rightChild = successor;

    // connect successor to current's left child
    successor.leftChild = current.leftChild;
} // end else two children
// (successor cannot have a left child)
return true;                           // success

```

```

} // end delete()

// -----
// returns node with next-highest value after delNode
// goes to right child, then right child's left descendents
private Node getSuccessor(Node delNode) {
    Node successorParent = delNode;
    Node successor = delNode;
    Node current = delNode.rightChild; // go to right child
    while (current != null) // until no more
    { // left children,
        successorParent = successor;
        successor = current;
        current = current.leftChild; // go to left child
    }
    // if successor not
    if (successor != delNode.rightChild) // right child,
    { // make connections
        successorParent.leftChild = successor.rightChild;
        successor.rightChild = delNode.rightChild;
    }
    return successor;
}

// -----
public void traverse(int traverseType) {
    switch (traverseType) {
        case 1:
            System.out.print("\nPreorder traversal: ");
            preOrder(root);
            break;
        case 2:
            System.out.print("\nInorder traversal: ");
            inOrder(root);
            break;
        case 3:
            System.out.print("\nPostorder traversal: ");
            postOrder(root);
            break;
    }
    System.out.println();
}

// -----
private void preOrder(Node localRoot) {
    if (localRoot != null) {
        System.out.print(localRoot.iData + " ");
        preOrder(localRoot.leftChild);
        preOrder(localRoot.rightChild);
    }
}

// -----
private void inOrder(Node localRoot) {
    if (localRoot != null) {
        inOrder(localRoot.leftChild);
        System.out.print(localRoot.iData + " ");
        inOrder(localRoot.rightChild);
    }
}

// -----
private void postOrder(Node localRoot) {
    if (localRoot != null) {

```

```

        postOrder(localRoot.leftChild);
        postOrder(localRoot.rightChild);
        System.out.print(localRoot.iData + " ");
    }
}

// -----
public void displayTree() {
    Stack globalStack = new Stack();
    globalStack.push(root);
    int nBlanks = 32;
    boolean isRowEmpty = false;
    System.out.println(
        ".....");
    while (isRowEmpty == false) {
        Stack localStack = new Stack();
        isRowEmpty = true;

        for (int j = 0; j < nBlanks; j++)
            System.out.print(' ');

        while (globalStack.isEmpty() == false) {
            Node temp = (Node) globalStack.pop();
            if (temp != null) {
                System.out.print(temp.iData);
                localStack.push(temp.leftChild);
                localStack.push(temp.rightChild);

                if (temp.leftChild != null ||
                    temp.rightChild != null)
                    isRowEmpty = false;
            } else {
                System.out.print("--");
                localStack.push(null);
                localStack.push(null);
            }
            for (int j = 0; j < nBlanks * 2 - 2; j++)
                System.out.print(' ');
        } // end while globalStack not empty
        System.out.println();
        nBlanks /= 2;
        while (localStack.isEmpty() == false)
            globalStack.push(localStack.pop());
    } // end while isRowEmpty is false
    System.out.println(
        ".....");
} // end displayTree()
// -----
} // end class Tree

////////////////////////////////////
class TreeApp {
    public static void main(String[] args) throws IOException {
        int value;
        Tree theTree = new Tree();

        theTree.insert(50, 1.5);
        theTree.insert(25, 1.2);
        theTree.insert(75, 1.7);
        theTree.insert(12, 1.5);
        theTree.insert(37, 1.2);
        theTree.insert(43, 1.7);
        theTree.insert(30, 1.5);
        theTree.insert(33, 1.2);
    }
}

```



```

theTree.insert(87, 1.7);
theTree.insert(93, 1.5);
theTree.insert(97, 1.5);

while (true) {
    System.out.print("Enter first letter of show, ");
    System.out.print("insert, find, delete, or traverse: ");
    int choice = getChar();
    switch (choice) {
        case 's':
            theTree.displayTree();
            break;
        case 'i':
            System.out.print("Enter value to insert: ");
            value = getInt();
            theTree.insert(value, value + 0.9);
            break;
        case 'f':
            System.out.print("Enter value to find: ");
            value = getInt();
            Node found = theTree.find(value);
            if (found != null) {
                System.out.print("Found: ");
                found.displayNode();
                System.out.print("\n");
            } else
                System.out.print("Could not find ");
            System.out.print(value + '\n');
            break;
        case 'd':
            System.out.print("Enter value to delete: ");
            value = getInt();
            boolean didDelete = theTree.delete(value);
            if (didDelete)
                System.out.print("Deleted " + value + '\n');
            else
                System.out.print("Could not delete ");
            System.out.print(value + '\n');
            break;
        case 't':
            System.out.print("Enter type 1, 2 or 3: ");
            value = getInt();
            theTree.traverse(value);
            break;
        default:
            System.out.print("Invalid entry\n");
    } // end switch
} // end while
} // end main()

// -----
public static String getString() throws IOException {
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    String s = br.readLine();
    return s;
}

// -----
public static char getChar() throws IOException {
    String s = getString();
    return s.charAt(0);
}

```

```

//-----
public static int getInt() throws IOException {
    String s = getString();
    return Integer.parseInt(s);
}
}

Fibonacci.java:
import java.util.Arrays;
import java.util.Scanner;

class Fibonacci {

    // Сервисная функция для поиска минимума из двух элементов
    public static int min(int x, int y) {
        return Math.min(x, y);
    }

    /*
    Возвращает индекс x, если присутствует, иначе возвращает -1
    */

    public static int fibMonaccianSearch(int arr[], int x, int n) {

        /*Инициализировать числа Фибоначчи */
        int fibMMm2 = 0; // (m-2) -ый номер Фибоначчи
        int fibMMm1 = 1; // (m-1) -ый номер Фибоначчи
        int fibM = fibMMm2 + fibMMm1; // m Фибоначчи

        /*fibM собирается хранить самые маленькие Число Фибоначчи, большее
или равное n */
        while (fibM < n) {
            fibMMm2 = fibMMm1;
            fibMMm1 = fibM;
            fibM = fibMMm2 + fibMMm1;
        }

        // Отмечает удаленный диапазон спереди
        int offset = -1;

        /*
        Пока есть элементы для проверки. Обратите внимание, что мы сравниваем
arr[fibMm2] с x. Когда fibM становится 1, fibMm2 становится 0
        */

        while (fibM > 1) {

            // Проверяем, является ли fibMm2 действительным местоположением
            int i = min(offset + fibMMm2, n - 1);

            /*Если x больше значения в индекс fibMm2, вырезать массив
подмассива от смещения до i */
            if (arr[i] < x) {
                fibM = fibMMm1;
                fibMMm1 = fibMMm2;
                fibMMm2 = fibM - fibMMm1;
                offset = i;
            }

            /*Если x больше, чем значение в индексе fibMm2, вырезать
подрешетку после i + 1 */
            else if (arr[i] > x) {
                fibM = fibMMm2;
                fibMMm1 = fibMMm1 - fibMMm2;

```

```

        fibMMm2 = fibM - fibMMm1;
    }
    //элемент найден. индекс возврата
    else return i + 1;
}

/* сравнение последнего элемента с x */
if (fibMMm1 == 1 && arr[offset + 1] == x)
    return offset + 1;

/* элемент не найден. возврат -1 */
return -1;
}

// код драйвера
public static void main(String[] args) {
    int x;
    Scanner scanner = new Scanner(System.in);

    System.out.println("Вы хотите ввести числа, или вставить готовый
массив? Напишите 'В' для ввода чисел или 'Г' для вставки готового массива:");
    String s = scanner.nextLine();
    int ch = s.charAt(0);
    switch (ch) {
        case 'В':
            System.out.println("Введите размер массива:");
            int kolvo = scanner.nextInt();
            int arr[] = new int[kolvo];
            for(int i = 0; i < kolvo; i++) {

                System.out.println("Введите число для заполнения
массива:");

                arr[i] = scanner.nextInt();
            }
            System.out.println("Введите число для поиска");
            x = scanner.nextInt();
            System.out.print("Found at index: " + fibMonaccianSearch(arr,
x, kolvo));
            break;
        case 'Г':
            int array[] = {10, 22, 35, 40, 45, 50, 80, 82, 85, 90, 100};
            System.out.println("Был сформирован массив: " +
Arrays.toString(array));
            System.out.println("Введите число для поиска");
            x = scanner.nextInt();
            System.out.print("Found at index: " +
fibonaccianSearch(array, x, array.length));
            break;
    }
}
}

```

#### Interpolation.java:

```

import java.util.Arrays;
import java.util.Scanner;

public class Interpolation {
    public static void main(String[] args) {
        int x;
        Scanner scanner = new Scanner(System.in);

        System.out.println("Вы хотите ввести числа, или вставить готовый
массив? Напишите 'В' для ввода чисел или 'Г' для вставки готового массива:");
    }
}

```

```

String s = scanner.nextLine();
int ch = s.charAt(0);
switch (ch) {
    case 'B':
        System.out.println("Введите размер массива:");
        int kolvo = scanner.nextInt();
        int arr[] = new int[kolvo];
        for (int i = 0; i < kolvo; i++) {
            System.out.println("Введите число для заполнения
массива:");
            arr[i] = scanner.nextInt();
        }
        System.out.println("Введите число для поиска");
        x = scanner.nextInt();
        System.out.print("Found at index: " +
interpolationSearch(arr, x));
        break;
    case 'Г':
        int array[] = {10, 22, 35, 40, 45, 50, 80, 82, 85, 90, 100};
        System.out.println("Был сформирован массив: " +
Arrays.toString(array));
        System.out.println("Введите число для поиска");
        x = scanner.nextInt();
        System.out.print("Found at index: " +
interpolationSearch(array, x));
        break;
}
}

public static int interpolationSearch(int[] integers, int
elementToSearch) {
    int startIndex = 0;
    int lastIndex = (integers.length - 1);

    while ((startIndex <= lastIndex) && (elementToSearch >=
integers[startIndex]) &&
(elementToSearch <= integers[lastIndex])) {
        int pos = startIndex + (((lastIndex - startIndex) /
(integers[lastIndex] - integers[startIndex])) *
(elementToSearch - integers[startIndex]));

        if (integers[pos] == elementToSearch)
            return pos + 1;

        if (integers[pos] < elementToSearch)
            startIndex = pos + 1;

        else
            lastIndex = pos - 1;
    }
    return -1;
}
}

```

#### HashChain.java:

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

class Link {
    public Link next;           // next link in list
    private final int iData;    // data item

    // -----

```

```

public Link(int it)                                // constructor
{
    iData = it;
}

public int getKey() {
    return iData;
}

public void displayLink()                          // display this link
{
    System.out.print(iData + " ");
}
} // end class Link

/////////////////////////////////////////////////////////////////
class SortedList {
    private Link first;                            // ref to first list item

    // -----
    public void SortedList()                      // constructor
    {
        first = null;
    }

    // -----
    public void insert(Link theLink)              // insert link, in order
    {
        int key = theLink.getKey();
        Link previous = null;                      // start at first
        Link current = first;
        // until end of list,
        while (current != null && key > current.getKey()) {
            previous = current;
            current = current.next;
        }
        if (previous == null)
            first = theLink;
        else
            previous.next = theLink;
        theLink.next = current;
    }

    public void delete(int key)
    {
        Link previous = null;
        Link current = first;
        // until end of list,
        while (current != null && key != current.getKey()) {
            // or key == current,
            previous = current;
            current = current.next;                // go to next link
        }
        // disconnect link
        if (previous == null)                      // if beginning of list
            first = first.next;                    // delete first link
        else                                       // not at beginning
            previous.next = current.next;          // delete current link
    } // end delete()

    // -----
    public Link find(int key)                     // find link

```

```

    {
        Link current = first;           // start at first
        // until end of list,
        while (current != null && current.getKey() <= key) {
// or key too small,
            if (current.getKey() == key)    // is this the link?
                return current;           // found it, return link
            current = current.next;        // go to next item
        }
        return null;                     // didn't find it
    } // end find()

// -----
public void displayList() {
    System.out.print("List (first-->last): ");
    Link current = first;           // start at beginning of list
    while (current != null)         // until end of list,
    {
        current.displayLink();    // print data
        current = current.next;   // move to next link
    }
    System.out.println();
}
} // end class SortedList

////////////////////////////////////
class HashTable {
    private final SortedList[] hashArray;    // array of lists
    private final int arraySize;

// -----
public HashTable(int size)                // constructor
{
    arraySize = size;
    hashArray = new SortedList[arraySize]; // create array
    for (int j = 0; j < arraySize; j++)    // fill array
        hashArray[j] = new SortedList();  // with lists
}

// -----
public void displayTable() {
    for (int j = 0; j < arraySize; j++) // for each cell,
    {
        System.out.print(j + ". "); // display cell number
        hashArray[j].displayList(); // display list
    }
}

public int hashFunc(int key)
{
    return key % arraySize;
}

public void insert(Link theLink)
{
    int key = theLink.getKey();
    int hashVal = hashFunc(key);
    hashArray[hashVal].insert(theLink);
}

// -----
public void delete(int key)                // delete a link
{
    int hashVal = hashFunc(key);           // hash the key

```

```

        hashArray[hashVal].delete(key); // delete link
    } // end delete()

    // -----
    public Link find(int key) // find link
    {
        int hashVal = hashFunc(key); // hash the key
        Link theLink = hashArray[hashVal].find(key); // get link
        return theLink; // return link
    }
} // end class HashTable

class HashChainApp {
    public static void main(String[] args) throws IOException {
        int aKey;
        Link aDataItem;
        int size, n, keysPerCell = 100;
        // get sizes
        System.out.print("Enter size of hash table: ");
        size = getInt();
        System.out.print("Enter initial number of items: ");
        n = getInt();
        // make table
        HashTable theHashTable = new HashTable(size);

        for (int j = 0; j < n; j++) // insert data
        {
            aKey = (int) (java.lang.Math.random() *
                keysPerCell * size);
            aDataItem = new Link(aKey);
            theHashTable.insert(aDataItem);
        }

        while (true) // interact with user
        {
            System.out.print("Enter first letter of ");
            System.out.print("show, insert, delete, or find: ");
            char choice = getChar();
            switch (choice) {
                case 's':
                    theHashTable.displayTable();
                    break;
                case 'i':
                    System.out.print("Enter key value to insert: ");
                    aKey = getInt();
                    aDataItem = new Link(aKey);
                    theHashTable.insert(aDataItem);
                    break;
                case 'd':
                    System.out.print("Enter key value to delete: ");
                    aKey = getInt();
                    theHashTable.delete(aKey);
                    break;
                case 'f':
                    System.out.print("Enter key value to find: ");
                    aKey = getInt();
                    aDataItem = theHashTable.find(aKey);
                    if (aDataItem != null)
                        System.out.println("Found " + aKey);
                    else
                        System.out.println("Could not find " + aKey);
                    break;
                default:
                    System.out.print("Invalid entry\n");
            }
        }
    }
}

```

```

        } // end switch
    } // end while
} // end main()

public static String getString() throws IOException {
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    String s = br.readLine();
    return s;
}

public static char getChar() throws IOException {
    String s = getString();
    return s.charAt(0);
}

public static int getInt() throws IOException {
    String s = getString();
    return Integer.parseInt(s);
}
}

```

#### Hash.java:

```

import java.io.*;
class DataItem
{
    private int iData; // (could have more data)
                     // data item (key)
//-----
    public DataItem(int ii) // constructor
    { iData = ii; }
//-----
    public int getKey()
    { return iData; }
//-----
} // end class DataItem
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
class HashTable
{
    private DataItem[] hashArray; // array holds hash table
    private int arraySize;
    private DataItem nonItem; // for deleted items
//-----
    public HashTable(int size) // constructor
    {
        arraySize = size;
        hashArray = new DataItem[arraySize];
        nonItem = new DataItem(-1); // deleted item key is -1
    }
//-----
    public void displayTable()
    {
        System.out.print("Table: ");
        for(int j=0; j<arraySize; j++)
        {
            if(hashArray[j] != null)
                System.out.print(hashArray[j].getKey() + " ");
            else
                System.out.print("** ");
        }
        System.out.println("");
    }
//-----
    public int hashFunc(int key)
    {

```



```

        return key % arraySize;          // hash function
    }
}
// -----
public void insert(DataItem item) // insert a DataItem
// (assumes table not full)
{
    int key = item.getKey();           // extract key
    int hashVal = hashFunc(key);       // hash the key
    // until empty cell or -1,
    while(hashArray[hashVal] != null &&
           hashArray[hashVal].getKey() != -1)
    {
        ++hashVal;                     // go to next cell
        hashVal %= arraySize;          // wraparound if necessary
    }
    hashArray[hashVal] = item;         // insert item
} // end insert()
// -----
public DataItem delete(int key) // delete a DataItem
{
    int hashVal = hashFunc(key);       // hash the key

    while(hashArray[hashVal] != null) // until empty cell,
    {                                   // found the key?
        if(hashArray[hashVal].getKey() == key)
        {
            DataItem temp = hashArray[hashVal]; // save item
            hashArray[hashVal] = nonItem;        // delete item
            return temp;                         // return item
        }
        ++hashVal;                     // go to next cell
        hashVal %= arraySize;          // wraparound if necessary
    }
    return null;                       // can't find item
} // end delete()
// -----
public DataItem find(int key) // find item with key
{
    int hashVal = hashFunc(key);       // hash the key

    while(hashArray[hashVal] != null) // until empty cell,
    {                                   // found the key?
        if(hashArray[hashVal].getKey() == key)
            return hashArray[hashVal]; // yes, return item
        ++hashVal;                     // go to next cell
        hashVal %= arraySize;          // wraparound if necessary
    }
    return null;                       // can't find item
}
// -----
} // end class HashTable
////////////////////////////////////
class HashTableApp
{
    public static void main(String[] args) throws IOException
    {
        DataItem aDataItem;
        int aKey, size, n, keysPerCell;

        // get sizes
        System.out.print("Enter size of hash table: ");
        size = getInt();
        System.out.print("Enter initial number of items: ");
        n = getInt();
        keysPerCell = 10;
    }
}

```

```

// make table
HashTable theHashTable = new HashTable(size);

for(int j=0; j<n; j++) // insert data
{
    aKey = (int) (java.lang.Math.random() *
                keysPerCell * size);
    aDataItem = new DataItem(aKey);
    theHashTable.insert(aDataItem);
}

while(true) // interact with user
{
    System.out.print("Enter first letter of ");
    System.out.print("show, insert, delete, or find: ");
    char choice = getChar();
    switch(choice)
    {
        case 's':
            theHashTable.displayTable();
            break;
        case 'i':
            System.out.print("Enter key value to insert: ");
            aKey = getInt();
            aDataItem = new DataItem(aKey);
            theHashTable.insert(aDataItem);
            break;
        case 'd':
            System.out.print("Enter key value to delete: ");
            aKey = getInt();
            theHashTable.delete(aKey);
            break;
        case 'f':
            System.out.print("Enter key value to find: ");
            aKey = getInt();
            aDataItem = theHashTable.find(aKey);
            if(aDataItem != null)
            {
                System.out.println("Found " + aKey);
            }
            else
                System.out.println("Could not find " + aKey);
            break;
        default:
            System.out.print("Invalid entry\n");
    } // end switch
} // end while
} // end main()

//-----
public static String getString() throws IOException
{
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    String s = br.readLine();
    return s;
}

//-----
public static char getChar() throws IOException
{
    String s = getString();
    return s.charAt(0);
}

//-----
public static int getInt() throws IOException

```

```

    {
        String s = getString();
        return Integer.parseInt(s);
    }
}
//-----
}

```

## 4. Результат работы программы



Рисунок 1 – Ферзи

Введите количество элементов массива:

5

Введите 5 чисел

5 7 1 3 4

Массив отсортирован: [1, 3, 4, 5, 7]

Введите элемент для бинарного поиска:

Рисунок 2 – Бинарный поиск

```
Enter size of hash table: 5
Enter initial number of items: 7
Enter first letter of show, insert, delete, or find: s
0. List (first-->last): 395 465
1. List (first-->last): 206
2. List (first-->last): 107 212
3. List (first-->last): 133
4. List (first-->last): 404
Enter first letter of show, insert, delete, or find:
```

Рисунок 3 – Рехеширование с помощью цепочек

```
"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...
Вы хотите ввести числа, или вставить готовый массив? Напишите 'В' для ввода чисел или 'Г' для вставки готового массива:
Г
Был сформирован массив: [10, 22, 35, 40, 45, 50, 80, 82, 85, 90, 100]
Введите число для поиска
```

Рисунок 4 – Фибоначчи

```
"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...
Вы хотите ввести числа, или вставить готовый массив? Напишите 'В' для ввода чисел или 'Г' для вставки готового массива:
Г
Был сформирован массив: [10, 22, 35, 40, 45, 50, 80, 82, 85, 90, 100]
Введите число для поиска
80
Found at index: 7
```

Рисунок 5 – Интерполяционный поиск

## 5. Вывод

Я рассмотрел различные методы поиска чисел в массиве.