Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОННОЙ ТЕХНИКИ»

Институт системной и программной инженерии

и информационных технологий (СПИНТех)

**Отчёт**

по дисциплине «Алгоритмы и структуры данных»

**Лабораторная работа №1**

**Вариант-10**

Руководитель

\_\_\_\_\_\_\_\_\_\_\_ Волков A. C.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Студент группы ПИН-23

\_\_\_\_\_\_\_\_\_\_\_ Исламов Р. Р.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

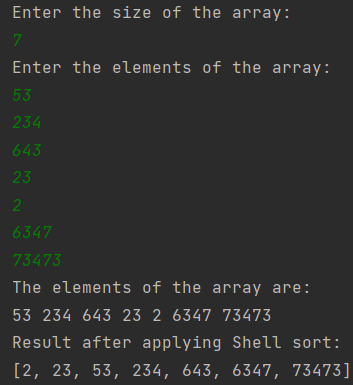
*Москва*

*2023*

# 1.1 Листинг. Метод Шелла

import java.util.Arrays;  
import java.util.Scanner;  
public class ShellSort  
{  
  
 public static int[] shellSort(int[] array) {  
 int h = 1;  
  
 while (h <= array.length / 3) {  
 h = h \* 3 + 1;  
 }  
  
 while (h > 0) {  
 for (int outer = h; outer < array.length; outer++) {  
 int tmp = array[outer];  
 int inner = outer;  
  
 while (inner > h - 1 && array[inner - h] > tmp) {  
 array[inner] = array[inner - h];  
 inner -= h;  
 }  
  
 array[inner] = tmp;  
 }  
  
 h = (h - 1) / 3;  
 }  
 return array;  
 }  
  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
  
 System.*out*.println("Enter the size of the array: ");  
 int arr\_size = 0;  
 if (sc.hasNextInt()) {  
 arr\_size = sc.nextInt();  
 }  
  
 int[] arr = new int[arr\_size];  
  
 System.*out*.println(  
 "Enter the elements of the array: ");  
 for (int i = 0; i < arr\_size; i++) {  
 if (sc.hasNextInt()) {  
 arr[i] = sc.nextInt();  
 }  
 }  
  
 System.*out*.println(  
 "The elements of the array are: ");  
 for (int i = 0; i < arr\_size; i++) {  
 System.*out*.print(arr[i] + " ");  
 }  
 sc.close();  
 // int[] arr = {53,423,234,423,234,23,423,42,234,2342,34};  
 System.*out*.println();  
 System.*out*.println("Result after applying Shell sort: ");  
 System.*out*.println(Arrays.*toString*(*shellSort*(arr)));  
 }  
}

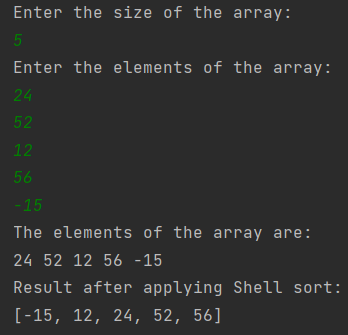
# 1.2 Результат работы

****

# 2.1 Листинг. Метод Хоара

import java.util.Arrays;  
import java.util.Scanner;  
  
public class QuickSort {  
  
 public static void quickSort(int[] array, int low, int high) {  
 if (array.length == 0)  
 return;  
  
 if (low >= high)  
 return;  
  
 int middle = low + (high - low) / 2;  
 int opora = array[middle];  
  
 int i = low, j = high;  
 while (i <= j) {  
 while (array[i] < opora) {  
 i++;  
 }  
  
 while (array[j] > opora) {  
 j--;  
 }  
  
 if (i <= j) {//меняем местами  
 int temp = array[i];  
 array[i] = array[j];  
 array[j] = temp;  
 i++;  
 j--;  
 }  
 }  
  
 // вызов рекурсии для сортировки левой и правой части  
 if (low < j)  
 *quickSort*(array, low, j);  
  
 if (high > i)  
 *quickSort*(array, i, high);  
 }  
 public static void main(String[] args) {  
  
  
 Scanner sc = new Scanner(System.*in*);  
  
 System.*out*.println("Enter the size of the array: ");  
 int arr\_size = 0;  
 if (sc.hasNextInt()) {  
 arr\_size = sc.nextInt();  
 }  
  
 int[] arr = new int[arr\_size];  
  
 System.*out*.println(  
 "Enter the elements of the array: ");  
 for (int i = 0; i < arr\_size; i++) {  
 if (sc.hasNextInt()) {  
 arr[i] = sc.nextInt();  
 }  
 }  
  
 System.*out*.println(  
 "The elements of the array are: ");  
 for (int i = 0; i < arr\_size; i++) {  
 System.*out*.print(arr[i] + " ");  
 }  
 sc.close();  
 // int[] arr = {53,423,234,423,234,23,423,42,234,2342,34};  
 System.*out*.println();  
 System.*out*.println("Result after applying Shell sort: ");  
 int low = 0;  
 int high = arr.length - 1;  
  
 *quickSort*(arr, low, high);  
 System.*out*.println(Arrays.*toString*(arr));  
 }  
}

# 2.2 Результат работы



# 3 Контрольные вопросы

Контрольные вопросы по тексту лабораторной 1:

1. Под сортировкой понимается процесс упорядочивания данных (элементов массива, списка) по заданному критерию (возрастанию, убыванию).

2. Особенности различных методов сортировки:

- Сортировка вставками: устойчивая сортировка, эффективна для небольших массивов.

- Сортировка выбором: неустойчивая сортировка, не требует дополнительной памяти.

- Сортировка обменом (пузырьковая): неустойчивая сортировка, проста в реализации, эффективна для почти отсортированных массивов.

- Сортировка Шелла: быстрая сортировка, эффективна для средних массивов.

- Быстрая сортировка (Хоара): быстрая сортировка с рекурсивным разбиением на подмассивы.

- Турнирная сортировка: устойчивая сортировка, эффективна для больших массивов.

- Сортировка пирамидой: устойчивая сортировка, эффективна для больших массивов.

3. Понятие сложности алгоритма включает в себя оценку количества операций, которые выполняет алгоритм при обработке данных в зависимости от размера входных данных.

4. Методика анализа сложности алгоритмов сортировки заключается в оценке количества операций для каждого алгоритма при различных размерах входных данных и сравнении полученных значений. Для этого можно использовать замер времени выполнения, общее число сравнений и перестановок элементов массива.

5. Линейная динамическая структура данных - это структура, которая состоит из элементов, каждый из которых имеет ссылку на следующий элемент. Примеры линейных динамических структур данных: связный список, стек, очередь. Характеристики таких структур данных: удобство изменения размера, эффективный доступ к первому и последнему элементу.

6. Принцип топологической сортировки заключается в упорядочивании элементов графа в порядке, который гарантирует, что все ребра будут направлены от меньших к большим элементам. Примеры использования: составление графика выполнения процессов, нахождение порядка зависимых задач. Для реализации этого метода часто используется структура данных "стек".

Контрольные вопросы по тексту лабораторной 2:

1. Под поиском понимается процесс нахождения нужной информации, элемента или решения в некотором наборе данных или структуре.
2. Методика анализа сложности алгоритмов поиска включает в себя изучение зависимости времени выполнения или количества операций от размера входных данных. Это включает в себя оценку временной сложности алгоритма и оценку его пространственной сложности.
3. Особенности поиска последовательного (линейного) заключаются в последовательном проходе через все элементы набора данных для нахождения искомого элемента. Временная сложность такого поиска составляет O(n), где n - количество элементов.  
   Особенности бинарного поиска заключаются в использовании предположения о том, что элементы отсортированы в упорядоченном порядке. Поиск основан на делении набора данных пополам на каждом шаге, и сравнении искомого элемента с элементом в середине. Временная сложность бинарного поиска составляет O(log n), где n - количество элементов.
4. Особенности интерполяционного поиска заключаются в использовании линейной интерполяции для приближенного определения положения искомого элемента в упорядоченном наборе данных. Временная сложность интерполяционного поиска в среднем составляет O(log log n), где n - количество элементов.  
   Особенности Фибоначчиевого поиска заключаются в использовании чисел Фибоначчи для определения доли прогресса при делении набора данных. Это позволяет эффективно перемещаться по набору данных. Временная сложность Фибоначчиевого поиска составляет O(log n), где n - количество элементов.
5. Особенности поиска по бинарному дереву заключаются в использовании свойств бинарного дерева для эффективного поиска элементов. Каждый узел бинарного дерева имеет левого и правого потомка, которые образуют отсортированную структуру данных. Поиск происходит путем сравнения искомого элемента с элементами в узлах и движения влево или вправо по дереву. Временная сложность поиска в сбалансированном бинарном дереве составляет O(log n), где n - количество элементов.
6. Особенности поиска по бору (Trie) заключаются в использовании префиксного дерева для хранения и поиска строковых данных. Каждый узел бора представляет символ, и путь от корня до узла образует строку. Поиск происходит путем последовательного перехода от узла к узлу в соответствии с символами искомой строки. Временная сложность поиска по бору составляет O(k), где k - длина искомой строки.  
   Особенности поиска с использованием хеширования заключаются в использовании хеш-функции для преобразования ключа (искомого элемента) в индекс в массиве, называемом хеш-таблицей. Поиск происходит путем вычисления хеша ключа и поиска соответствующего индекса в хеш-таблице. Временная сложность поиска с использованием хеширования в среднем составляет O(1), но может быть O(n) в худшем случае, если возникают коллизии.
7. Методика анализа сложности алгоритмов поиска аналогична методике анализа сложности других алгоритмов и включает оценку временной сложности (количество операций или время выполнения) и пространственной сложности (используемая память) в зависимости от размера входных данных.
8. Алгоритмы поиска словесной информации отличаются от алгоритмов поиска числовых данных тем, что они работают с текстовыми данными, например, в документах, базах данных или веб-страницах. В таких алгоритмах используются методы обработки текста, такие как токенизация, сравнение строк, поиск шаблонов и т. д., чтобы найти искомые фрагменты информации в тексте.
9. Коллизия в контексте хеширования означает ситуацию, когда два разных ключа (элемента) приводят к одному и тому же индексу в хеш-таблице. Это может привести к потере данных или снижению производительности алгоритма.
10. Для разрешения коллизий в хеш-таблицах используются различные методы, включая метод цепочек (открытое связывание), метод открытой адресации (линейное пробирование, двойное хеширование) и метод косвенной адресации (использование дополнительной структуры данных для хранения элементов с одинаковым индексом).
11. Показатель сбалансированности узла дерева (например, в сбалансированном бинарном дереве поиска) обычно определяется по разнице глубины его левого и правого поддеревьев. Сбалансированный узел имеет показатель баланса, равный 0, -1 или 1, что гарантирует эффективность операций поиска и вставки в дерево.
12. Понятия:

* Дерево: Иерархическая структура данных, состоящая из узлов, связанных ребрами, и имеющая корневой узел без входящих ребер.
* Двоичное дерево: Дерево, в котором каждый узел имеет не более двух потомков (левого и правого).
* Упорядоченное дерево: Двоичное дерево, в котором каждый узел имеет ключ (значение), и для каждого узла значения в его левом поддереве меньше его значения, а значения в его правом поддереве больше его значения.
* Дерево поиска: Упорядоченное дерево, в котором выполняются свойства двоичного дерева и упорядоченности ключей.

1. Способы задания дерева:

* Явное задание: Узлы и связи между ними задаются явно, например, с использованием списков, массивов или структур данных.
* Рекурсивное задание: Дерево задается рекурсивно через его поддеревья или подузлы.
* Генеративное задание: Дерево генерируется на основе некоторых правил или алгоритма, например, дерево разбора грамматики.