Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу

Кафедра ІПЗ

**Лабораторна робота №8**

**Тема: “ АЛГОРИТМИ СОРТУВАННЯ ”**

Виконав

ст.гр. ІП-22-1

Токарєв С.О.

Перевірив:

Григорчук Л.І.

Івано-Франківськ

2023

Теоретичні відомості:

**QuickSort:**

QuickSort - це алгоритм сортування, який використовує стратегію "розділити і володарювати". Алгоритм був придуманий Тоні Хоаром у 1960 році. В основі лежить принцип розбиття масиву на менші частини, сортування кожної частини окремо, а потім об'єднання вже відсортованих частин.

Основні кроки алгоритму QuickSort:

1. **Вибір опорного елемента (Pivot):** Обирається один з елементів масиву як опорний елемент. Це може бути середній елемент, перший елемент, останній елемент або випадковий елемент.
2. **Розподіл елементів:** Решта елементів розподіляється так, щоб елементи, менші за опорний, знаходилися ліворуч від нього, а більші - праворуч. Опорний елемент вже знаходиться на своєму кінцевому місці у відсортованому масиві.
3. **Рекурсивне сортування підмасивів:** Застосовується той самий процес до лівого і правого підмасивів, поки весь масив не буде відсортований.

Алгоритм QuickSort є швидким і ефективним на практиці, а його часова складність у середньому випадку становить O(n log n). Однак, в найгіршому випадку, коли опорний елемент завжди обирається так, що одна з половин масиву порожня, часова складність може становити O(n^2).

**Binary Search:**

Бінарний пошук - це алгоритм пошуку, який працює на відсортованому масиві. Основна ідея полягає в тому, щоб порівнювати ключ для пошуку з середнім елементом масиву та виключати половину елементів, в залежності від результату порівняння.

Основні кроки алгоритму Binary Search:

1. **Вибір середини масиву:** Визначення середнього елемента у відсортованому масиві.
2. **Порівняння ключа з серединою:** Порівняння ключа для пошуку зі значенням середнього елемента.
3. **Визначення нового діапазону пошуку:** Якщо ключ менший за середній елемент, пошук ведеться в лівій половині масиву; якщо більший - в правій половині.
4. **Повторення процесу:** Повторення кроків 1-3 до тих пір, поки ключ не буде знайдений або діапазон пошуку не стане порожнім.

Бінарний пошук має часову складність O(log n), що робить його ефективним для великих відсортованих масивів.

Завдання:

**8.2. Завдання до лабораторної роботи 8**

Розробити програму, яка читає з клавіатури:

1) числа *N, M (1<N, M<256*)

2) послідовність N ключів (цілих або дійсних чисел залежно від варіанту завдання);

3) послідовність M ключів.

Програма зберігає першу послідовність до масиву та виконує сортуван ня. Потім програма виводить відсортовану послідовність на екран та виконує бінарний пошук кожного елемента другої послідовності : для кожного по відомити, чи є він у першій послідовності, а якщо є, то на якому місці.

**8.3. Варіанти завдань до лабораторної роботи 8**

***Варіант 3:*** швидке сортування.

**Код:**

package lab8;

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

// Input N and M

System.***out***.println("Enter N (1 < N < 256): ");

int N = scanner.nextInt();

System.***out***.println("Enter M (1 < M < 256): ");

int M = scanner.nextInt();

// Input sequence of N keys

System.***out***.println("Enter the sequence of N keys:");

double[] sequenceN = new double[N];

for (int i = 0; i < N; i++) {

sequenceN[i] = scanner.nextDouble();

}

// Input sequence of M keys

System.***out***.println("Enter the sequence of M keys:");

double[] sequenceM = new double[M];

for (int i = 0; i < M; i++) {

sequenceM[i] = scanner.nextDouble();

}

// Sorting the sequence of N keys using quicksort

*quickSort*(sequenceN, 0, N - 1);

// Output the sorted sequence

System.***out***.println("Sorted sequence of N keys: " + Arrays.*toString*(sequenceN));

// Binary search for each element in the second sequence

System.***out***.println("Binary search results:");

for (int i = 0; i < M; i++) {

int index = *binarySearch*(sequenceN, sequenceM[i]);

if (index != -1) {

System.***out***.println("Element " + sequenceM[i] + " is present in the sequence at position " + (index + 1));

} else {

System.***out***.println("Element " + sequenceM[i] + " is not found in the sequence");

}

}

}

// QuickSort

private static void quickSort(double[] arr, int low, int high) {

if (low < high) {

int pi = *partition*(arr, low, high);

*quickSort*(arr, low, pi - 1);

*quickSort*(arr, pi + 1, high);

}

}

private static int partition(double[] arr, int low, int high) {

double pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

double temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

double temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

// BinarySearch

private static int binarySearch(double[] arr, double key) {

int low = 0, high = arr.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

if (arr[mid] == key) {

return mid;

} else if (arr[mid] < key) {

low = mid + 1;

} else {

high = mid - 1;

}

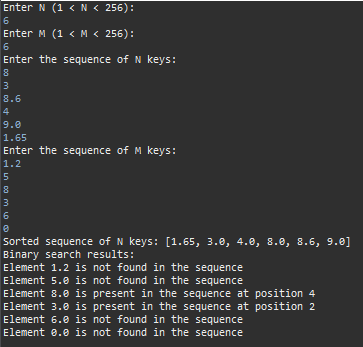
}

return -1;

}

}

Результат виконання:



Приклад з цілими і дробовими числами, які сходять і не сходяться

Висновок:

У цій лабораторній роботі я вивчив та реалізував два основних алгоритми: QuickSort для сортування масиву та Binary Search для ефективного пошуку елементів у відсортованому масиві.