МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Казанский национальный исследовательский технический университет

им. А.Н. Туполева – КАИ»

Институт компьютерных технологий и защиты информации

Отделение СПО в ИКТЗИ (Колледж информационных технологий)

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

Теория алгоритмов

Тема: ВЫЧИСЛЕНИЕ СЛОЖНОСТИ РАБОТЫ АЛГОРИТМОВ

Работу выполнил

Студент гр.4332

Салахов К.И.

Принял

Преподаватель Валова П.А.

Казань 2024

**Цель**

Научиться оценивать сложность алгоритмов.

**Задание на лабораторную работу**

Вариант 16:

* Константная сложность

3. Получение последней цифры числа. Напишите функцию, которая принимает целое число и возвращает его последнюю цифру.

* Логарифмическая сложность

1. Поиск медианы. Реализовать функцию, которая находит медиану двух отсортированных массивов различной длины.

* Линейная сложность

1. Найти индекс первого вхождения определенного элемента в массив.

* Линейно-логарифмическая сложность

1. Реализуйте сортировку пирамидой.

* Квадратичная сложность

1. Реализовать алгоритм, который найдет наибольший элемент в каждой строке квадратной матрицы.

Алгоритмы **экспоненциальной** и **факториальной** сложности необходимо придумать самостоятельно.

**Результат работы программы**

Константная сложность.

Функция LastNumber имеет константную сложность, потому что время выполнения не зависит от размера входного числа. Это означает, что независимо от того, является ли число однозначным или состоит из миллионов цифр, время выполнения функции будет одинаковым и очень маленьким (Рисунок 1 – 2).

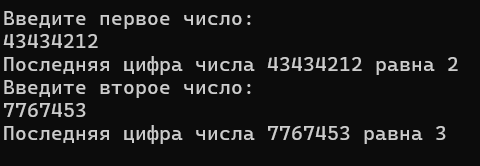


Рисунок 1 – Результат работы программы для константной сложности

Изображение выглядит как текст, диаграмма, снимок экрана, круг

Автоматически созданное описание

Рисунок 2 – Блок схема для константной сложности

Логарифмическая сложность.

Данный код имеет логарифмическую сложность, потому что он использует бинарный поиск для нахождения правильного разделения массивов. Это помогает эффективно находить медиану двух отсортированных массивов даже для больших размеров массивов (Рисунок 3–4).

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 3 – Результат работы программы для логарифмической сложности

Изображение выглядит как снимок экрана, Графика, текст, черно-белый

Автоматически созданное описание

Рисунок 4 – Блок схема для логарифмической сложности

Линейная сложность.

Данный код имеет линейную сложность это означает, что время выполнения алгоритма растет линейно с увеличением размера входных данных. Если массив содержит n элементов, то алгоритм выполнит n сравнений. В худшем случае (когда искомый элемент находится в самом конце массива или отсутствует), алгоритм будет проходить по всем элементам массива (Рисунок 5–6).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5 - Результат работы программы для линейной сложности

Изображение выглядит как диаграмма, зарисовка, текст, рисунок

Автоматически созданное описание

Рисунок 6 – Блок схема для линейной сложности

Линейно-логарифмическая сложность.

В коде используется пирамидальная сортировка. Общая сложность алгоритма пирамидальной сортировки составляет O(n log n), что является линейно-логарифмической сложностью. Это означает, что время выполнения алгоритма растет пропорционально произведению размера массива на логарифм этого размера (Рисунок 7–8)

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 7 - Результат работы программы для линейно-логарифмическая сложности

Изображение выглядит как текст, снимок экрана, графический дизайн, Графика

Автоматически созданное описание

Рисунок 8 – Блок схема для линейно-логарифмической сложности.

Квадратичная сложность.

Данный код имеет квадратичную сложность, потому что он выполняет вложенные циклы, которые зависят от размеров матрицы. Сложность алгоритма определяется как O (rows \* cols), что эквивалентно O(n^2), где n — размер матрицы (количество элементов). Это означает, что время выполнения алгоритма растет квадратично с увеличением размера матрицы (Рисунок 9–10).

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рисунок 9 – Результат работы программы для квадратичной сложности

Изображение выглядит как текст, снимок экрана, Шрифт, круг

Автоматически созданное описание

Рисунок 10 – Блок схема для константной сложности

Экспоненциальная сложность.

В данном коде вычисляется n-ое число Фибоначчи с помощью рекурсии. Экспоненциальная сложность, часто наблюдается в алгоритмах, которые используют рекурсию без каких-либо оптимизаций. Временная сложность данного алгоритма составляет O(2^n), что и означает экспоненциальную сложность. Это означает, что время выполнения алгоритма будет резко расти с увеличением входного параметра n (Рисунок 11–12).

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

Рисунок 11 – Результат работы программы для экспоненциальной сложности

Изображение выглядит как диаграмма, текст, План, линия

Автоматически созданное описание

Рисунок 10 – Блок схема для экспоненциальной сложности

Факториальная сложность.

Алгоритм решения задачи коммивояжёра в данном коде следует из-за его глубокой рекурсии и полного перебора всех возможных маршрутов, что приводит к факториальной сложности O(n!). Это делает простой переборный метод неэффективным для больших значений n. Временная сложность такого рекурсивного алгоритма оценивается как O(n!). Это означает, что количество операций растет факториально с увеличением n (Рисунок 11-12).

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 11 – Результат работы программы для факториальной сложности

Изображение выглядит как диаграмма, текст, План, Технический чертеж

Автоматически созданное описание

Рисунок 12 – Блок схема для факториальной сложности

**Листинг**

Константная сложность

using System;

class Program

{

public static int LastNumber(int number)

{

return Math.Abs(number) % 10;

}

public static void Main()

{

Console.WriteLine("Введите первое число:");

int number = int.Parse(Console.ReadLine());

int lastNum = LastNumber(number);

Console.WriteLine($"Последняя цифра числа {number} равна {lastNum}");

Console.WriteLine("Введите второе число:");

int negativeNumber = int.Parse(Console.ReadLine());

int lastNum2 = LastNumber(negativeNumber);

Console.WriteLine($"Последняя цифра числа {negativeNumber} равна {lastNum2}");

}

}

Логарифмическая

int[] mas1 = new int[10];

int[] mas2 = new int[12];

Random rand = new Random();

for (int i = 0; i < mas1.Length; i++)

{

mas1[i] = rand.Next(-100, 100);

}

for (int i = 0; i < mas2.Length; i++)

{

mas2[i] = rand.Next(-100, 100);

}

Array.Sort(mas1);

Array.Sort(mas2);

Console.WriteLine("Первый отсортированный массив: " + string.Join(" ", mas1));

Console.WriteLine("Второй отсортированный массив: " + string.Join(" ", mas2));

double med = Median(mas1, mas2);

Console.WriteLine("Медиана двух массивов: " + med);

static double Median(int[] arr1, int[] arr2)

{

int n = arr1.Length;

int m = arr2.Length;

if (n > m)

{

return Median(arr2, arr1);

}

int low = 0, high = n;

while (low <= high)

{

int mid1 = (low + high) / 2;

int mid2 = (n + m + 1) / 2 - mid1;

int l1;

if (mid1 == 0)

{

l1 = int.MinValue;

}

else

{

l1 = arr1[mid1 - 1];

}

int r1;

if (mid1 == n)

{

r1 = int.MaxValue;

}

else

{

r1 = arr1[mid1];

}

int l2;

if (mid2 == 0)

{

l2 = int.MinValue;

}

else

{

l2 = arr2[mid2 - 1];

}

int r2;

if (mid2 == m)

{

r2 = int.MaxValue;

}

else

{

r2 = arr2[mid2];

}

if (l1 <= r2 && l2 <= r1)

{

if ((n + m) % 2 == 0)

{

return (Math.Max(l1, l2) + Math.Min(r1, r2)) / 2.0;

}

else

{

return Math.Max(l1, l2);

}

}

else if (l1 > r2)

{

high = mid1 - 1;

}

else

{

low = mid1 + 1;

}

}

return -1;

}

Линейная сложность

using System;

class Program

{

static void Main()

{

Console.WriteLine("Введите элементы массива через пробел:");

string[] inputArray = Console.ReadLine().Split(' ');

int[] array = Array.ConvertAll(inputArray, int.Parse);

Console.WriteLine("Введите искомый элемент:");

int target = int.Parse(Console.ReadLine());

int index = FirstOccurrence(array, target);

if (index != -1)

{

Console.WriteLine($"Первое вхождение элемента {target} находится на позиции {index}.");

}

else

{

Console.WriteLine($"Элемент {target} не найден в массиве.");

}

}

static int FirstOccurrence(int[] array, int target)

{

for (int i = 0; i < array.Length; i++)

{

if (array[i] == target)

{

return i;

}

}

return -1;

}

}

Линейно-логарифмическая сложность

int[] mas = new int[15];

Random random = new Random();

int swaps = 0;

int comp = 0;

for (int i = 0; i < mas.Length; i++)

mas[i] = random.Next(-100,100);

Console.WriteLine("Исходный массив:" + string.Join(", ", mas));

PyramidSort(mas);

Console.WriteLine("Отсортированный массив" + string.Join(", ", mas));

Console.WriteLine("Количество замен: " + swaps);

Console.WriteLine("Количество сравнений: " + comp);

void PyramidSort(int[] mas)

{

int n = mas.Length;

for (int i = n / 2 - 1; i >= 0; i--)

Heapify(mas, n, i);

for (int i = n - 1; i > 0; i--)

{

Swap(mas, 0, i);

Heapify(mas, i, 0);

}

}

void Heapify(int[] mas, int n, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if(left < n)

{

comp++;

if (mas[left] > mas[largest])

{

largest = left;

}

}

if (right < n)

{

comp++;

if (mas[right] > mas[largest])

{

largest = right;

}

}

if(largest != i)

{

Swap(mas, i, largest);

Heapify(mas,n,largest);

}

}

void Swap(int[] mas, int a, int b)

{

int temp = mas[a];

mas[a] = mas[b];

mas[b] = temp;

swaps++;

}

Квадратичная сложность

using System;

class Program

{

static void Main()

{

int[,] mas = new int[3, 3] { { 1, 2, 3 }, { 5, 14, 4 }, { 3, 4, 11 } };

FindMaxInRows(mas);

}

static void FindMaxInRows(int[,] matrix)

{

int rows = matrix.GetLength(0);

int cols = matrix.GetLength(1);

for (int i = 0; i < rows; i++)

{

int max = matrix[i, 0];

for (int j = 1; j < cols; j++)

{

if (matrix[i, j] > max)

{

max = matrix[i, j];

}

}

Console.WriteLine($"Максимальный элемент в строке {i + 1}: {max}");

}

}

}

Экспоненциальная сложность.

using System;

class Program

{

static void Main()

{

Console.WriteLine("Введите число для вычисления числа Фибоначчи: ");

int n = int.Parse(Console.ReadLine());

Console.WriteLine($"{n}-ое число Фибоначчи равно {Fibonacci(n)}");

}

static int Fibonacci(int n)

{

if (n <= 1)

return n;

return Fibonacci(n - 1) + Fibonacci(n - 2);

}

}

Факториальная сложность

using System;

class Program

{

static void Main()

{

int[,] distanceMatrix = new int[,]

{

{ 0, 10, 15, 20 },

{ 10, 0, 25, 25 },

{ 15, 25, 0, 30 },

{ 20, 25, 30, 0 }

};

int result = TravelingSalesman(distanceMatrix);

Console.WriteLine($"Минимальная стоимость маршрута: {result}");

}

static int TravelingSalesman(int[,] distanceMatrix)

{

int n = distanceMatrix.GetLength(0);

bool[] visited = new bool[n];

int minCost = int.MaxValue;

FindPath(0, 0, visited, 1, distanceMatrix, ref minCost);

return minCost;

}

static void FindPath(int currentPosition, int currentCost, bool[] visited, int visitedCount, int[,] distanceMatrix, ref int minCost)

{

int n = distanceMatrix.GetLength(0);

if (visitedCount == n && distanceMatrix[currentPosition, 0] > 0)

{

int totalCost = currentCost + distanceMatrix[currentPosition, 0];

minCost = Math.Min(minCost, totalCost);

return;

}

for (int nextCity = 0; nextCity < n; nextCity++)

{

if (!visited[nextCity] && distanceMatrix[currentPosition, nextCity] > 0)

{

visited[nextCity] = true;

FindPath(nextCity, currentCost + distanceMatrix[currentPosition, nextCity], visited, visitedCount + 1, distanceMatrix, ref minCost);

visited[nextCity] = false;

}

}

}

}