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.4338

Бусов В.Р.

Принял

Преподаватель Валова П.А.

Казань 2024

**Цель работы**

Научиться оценивать сложность алгоритмов.

**Ход работы**

**Вариант 3**

**Задание**

1. Для каждого класса сложности реализовать алгоритм на любом языке программирования согласно своему варианту. В данном задании вы будете реализовывать сортировку. В этот алгоритм необходимо добавить метод, который будет рассчитывать количество перестановок, количество сравнений и время выполнения работы алгоритма.

2. Для каждого реализованного алгоритма нарисовать блок схему.

3. Описать работу каждого алгоритма и объяснить почему ему присваивается конкретная сложность.

**Константная сложность**

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

using System;

Class Program

{

Static void Main()

{

Int number = 12345;

Int lastDigit = GetLastDigit(number);

Console.WriteLine($»Последняя цифра числа {number} – {lastDigit}»);

}

Static int GetLastDigit(int num)

{

Return num % 10;

}

}

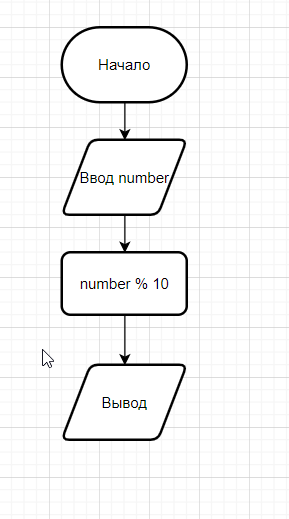


Рисунок 1 – блок схема к программе

**Логарифмическая сложность**

Поиск позиции в отсортированном массиве. Реализовать функцию, которая находит позицию, куда можно вставить элемент в отсортированном массиве, сохранив порядок.

using System;

class Program

{

static void Main()

{

int[] sortedArray = { 1, 3, 5, 6 };

int target = 5;

int position = SearchInsertPosition(sortedArray, target);

Console.WriteLine($"Элемент {target} можно вставить на позицию: {position}");

}

static int SearchInsertPosition(int[] nums, int target)

{

int left = 0, right = nums.Length;

while (left < right)

{

int mid = left + (right - left) / 2;

if (nums[mid] < target)

left = mid + 1;

else

right = mid;

}

return left;

}

}

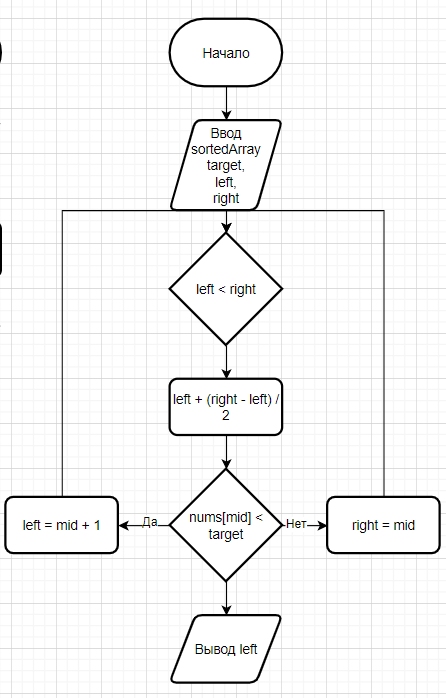


Рисунок 2 – блок схема к программе

**Линейная сложность**

Найти минимальное значение в массиве

using System;

class Program

{

static void Main()

{

int[] numbers = { 3, 5, 7, 2, 8, 1 };

int minValue = FindMinValue(numbers);

Console.WriteLine($"Минимальное значение в массиве: {minValue}");

}

static int FindMinValue(int[] nums)

{

int min = nums[0];

foreach (int num in nums)

{

if (num < min)

{

min = num;

}

}

return min;

}

}

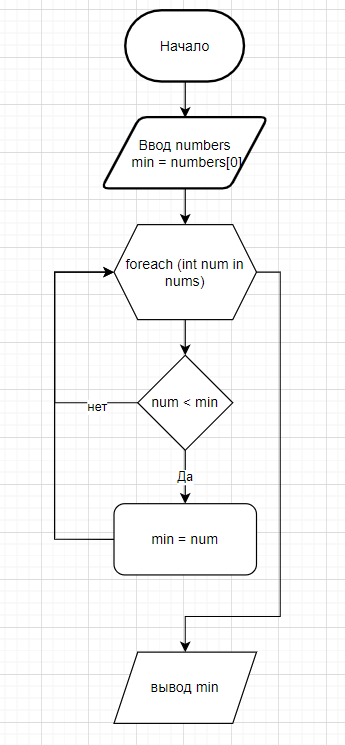


Рисунок 3 – блок схема к программе

**Линейно-логарифмическая сложность**

Реализуйте сортировку кучей

using System;

class Program

{

static void Main()

{

int[] array = { 12, 11, 13, 5, 6, 7 };

HeapSort(array);

Console.WriteLine("Отсортированный массив:");

foreach (var item in array)

{

Console.Write(item + " ");

}

}

static void HeapSort(int[] arr)

{

int n = arr.Length;

// Строим кучу (перегруппируем элементы массива)

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

{

Heapify(arr, n, i);

}

// Один за одним извлекаем элементы из кучи

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

{

// Перемещаем текущий корень в конец

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

// Вызываем Heapify на уменьшенной куче

Heapify(arr, i, 0);

}

}

// Функция для построения кучи

static void Heapify(int[] arr, int n, int i)

{

int largest = i; // Инициализируем largest как корень

int left = 2 \* i + 1; // Левый дочерний элемент

int right = 2 \* i + 2; // Правый дочерний элемент

// Если левый дочерний элемент больше корня

if (left < n && arr[left] > arr[largest])

{

largest = left;

}

// Если правый дочерний элемент больше largest

if (right < n && arr[right] > arr[largest])

{

largest = right;

}

// Если largest не корень, меняем местами и продолжаем Heapify

if (largest != i)

{

int swap = arr[i];

arr[i] = arr[largest];

arr[largest] = swap;

// Рекурсивный вызов Heapify для затронутого поддерева

Heapify(arr, n, largest);

}

}

}

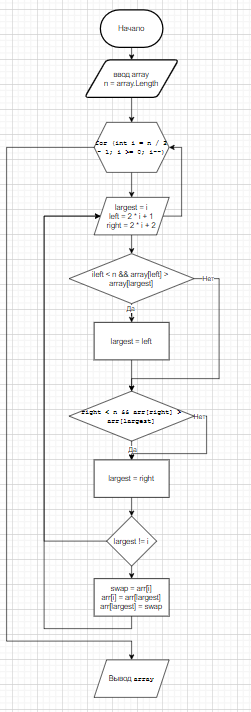


Рисунок 4 – блок схема к программе

**Квадратичная сложность**

Реализовать алгоритм, который найдет все пары чисел в квадратном массиве, сумма которых равна заданному значению

using System;

class Program

{

static void Main()

{

int[,] matrix = {

{ 1, 2, 3 },

{ 4, 5, 6 },

{ 7, 8, 9 }

};

int targetSum = 10;

FindPairsWithSum(matrix, targetSum);

}

static void FindPairsWithSum(int[,] matrix, int targetSum)

{

int rows = matrix.GetLength(0);

int cols = matrix.GetLength(1);

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

{

for (int j = 0; j < cols; j++)

{

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

{

for (int l = (k == i) ? j + 1 : 0; l < cols; l++)

{

if (matrix[i, j] + matrix[k, l] == targetSum)

{

Console.WriteLine($"Пара найдена: ({matrix[i, j]}, {matrix[k, l]})");

}

}

}

}

}

}

}

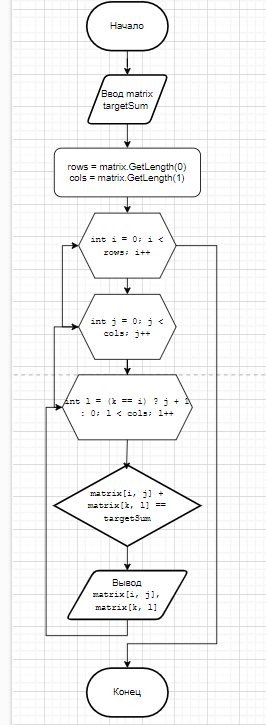


Рисунок 5 – блок схема к программе

**Экспоненциальная сложность**

Найти все подмножества множества.

static void Main()

{

Console.WriteLine("Введите элементы множества через запятую (например, 1,2,3):");

string input = Console.ReadLine();

int[] set = Array.ConvertAll(input.Split(','), int.Parse);

List<List<int>> subsets = FindSubsets(set);

Console.WriteLine("Подмножества:");

foreach (var subset in subsets)

{

Console.WriteLine("{" + string.Join(", ", subset) + "}");

}

Console.ReadLine();

}

static List<List<int>> FindSubsets(int[] set)

{

List<List<int>> allSubsets = new List<List<int>>();

int subsetCount = (int)Math.Pow(2, set.Length);

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

{

List<int> subset = new List<int>();

for (int j = 0; j < set.Length; j++)

{

if ((i & (1 << j)) != 0)

{

subset.Add(set[j]);

}

}

allSubsets.Add(subset);

}

return allSubsets;

}

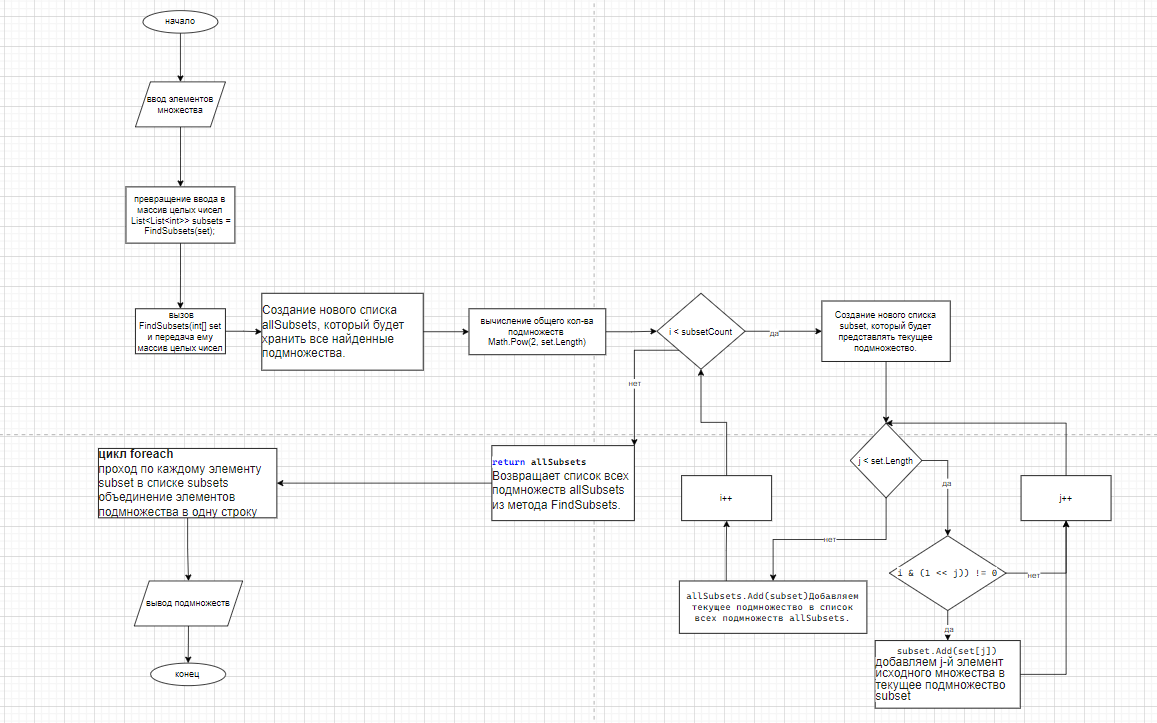


Рисунок 6 – блок схема к программе

**Факториальная сложность**

static void Main()

{

Console.WriteLine("Введите элементы массива через запятую (например, 1,2,3):");

string input = Console.ReadLine();

int[] array = Array.ConvertAll(input.Split(','), int.Parse);

List<List<int>> permutations = GetPermutations(array, 0);

Console.WriteLine("Перестановки:");

foreach (var perm in permutations)

{

Console.WriteLine("{" + string.Join(", ", perm) + "}");

}

Console.ReadLine();

}

static List<List<int>> GetPermutations(int[] array, int start)

{

List<List<int>> result = new List<List<int>>();

if (start == array.Length - 1)

{

result.Add(new List<int>(array));

return result;

}

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

{

Swap(ref array[start], ref array[i]);

result.AddRange(GetPermutations(array, start + 1));

Swap(ref array[start], ref array[i]); // backtrack

}

return result;

}

static void Swap(ref int a, ref int b)

{

int temp = a;

a = b;

b = temp;

}

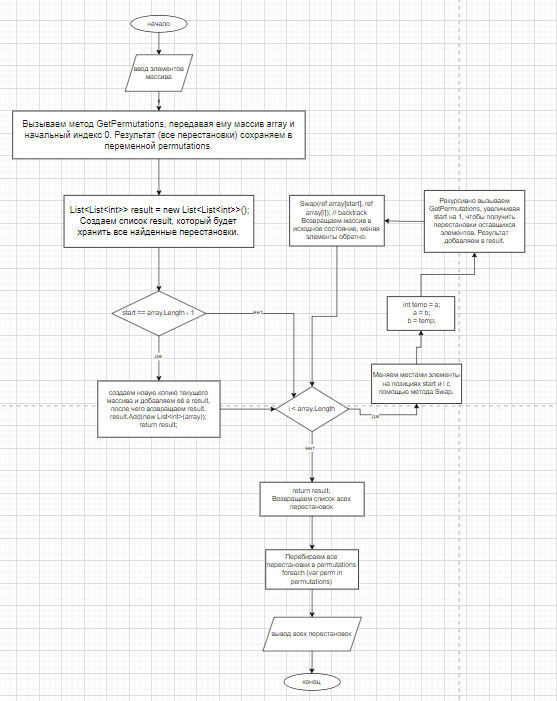


Рисунок 7 – блок схема к программе