Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №10
дисциплины «Алгоритмизация»

Выполнил:
Лейс Алексей Вячеславович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем»

_____
(подпись)

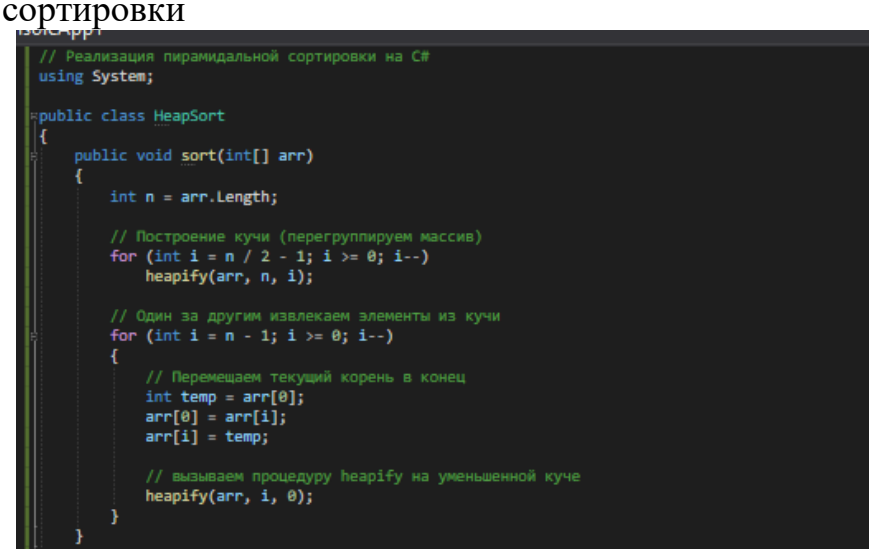
Руководитель практики кандидат тех.
наук доцент кафедры
инфокоммуникаций: Воронкин Р.А
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

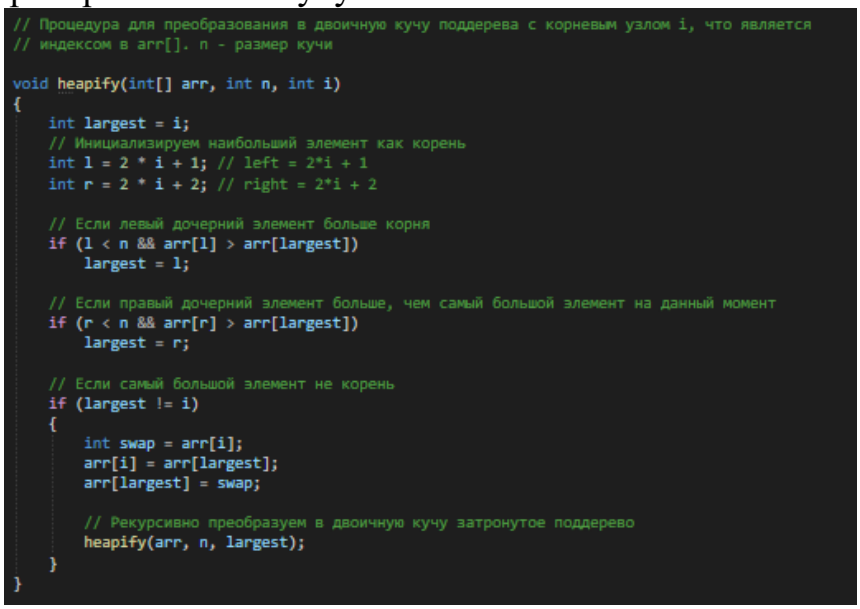
Порядок выполнения работы:

Алгоритм сортировки кучей Heap Sort:
Процедура сортировки



```
// Реализация пирамидальной сортировки на C#
using System;

public class HeapSort
{
    public void sort(int[] arr)
    {
        int n = arr.Length;

        // Построение кучи (перегруппируем массив)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // Один за другим 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 процедуру heapify на уменьшенной куче
            heapify(arr, i, 0);
        }
    }
```

Процедура преобразования в кучу



```
// Процедура для преобразования в двоичную кучу поддерева с корневым узлом i, что является
// индексом в arr[]. n - размер кучи

void heapify(int[] arr, int n, int i)
{
    int largest = i;
    // Инициализируем наибольший элемент как корень
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    // Если левый дочерний элемент больше корня
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // Если правый дочерний элемент больше, чем самый большой элемент на данный момент
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // Если самый большой элемент не корень
    if (largest != i)
    {
        int swap = arr[i];
        arr[i] = arr[largest];
        arr[largest] = swap;

        // Рекурсивно преобразуем в двоичную кучу затронутое поддерево
        heapify(arr, n, largest);
    }
}
```

Метод Main и вывод массива

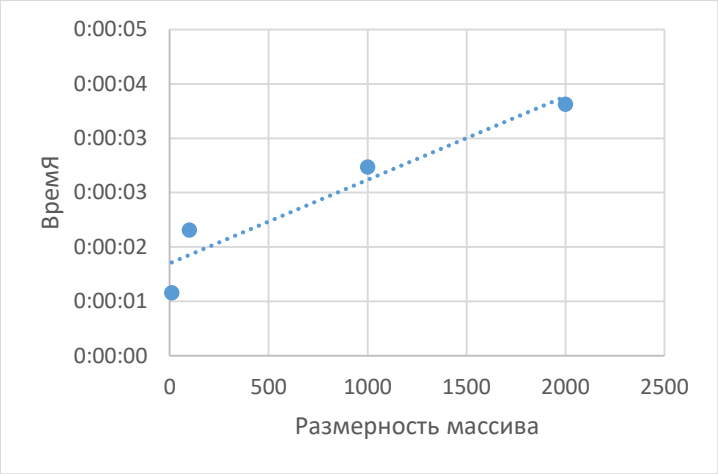
```
    /* Вспомогательная функция для вывода на экран массива размера n */
    static void printArray(int[] arr)
    {
        int n = arr.Length;
        for (int i = 0; i < n; ++i)
            Console.Write(arr[i] + " ");
        Console.Read();
    }

    //Управляющая программа
    public static void Main()
    {
        Console.Write("Введите длину массива: ");
        int a = Convert.ToInt32(Console.ReadLine());
        int[] arr = new int[a];

        for (int i = 0; i < arr.Length; i++)
        {
            Console.Write($"Arr[{i+1}] = ");
            arr[i] = Convert.ToInt32(Console.ReadLine());
        }


        int n = arr.Length;

        HeapSort ob = new HeapSort();
        ob.sort(arr);

        Console.WriteLine("Отсортированный массив:");
        printArray(arr);
    }
}
```

Результат работы программы:

```
Введите длину массива: 10
Arr[1] = 7
Arr[2] = 9
Arr[3] = 5
Arr[4] = 0
Arr[5] = 4
Arr[6] = 3
Arr[7] = 24
Arr[8] = 15
Arr[9] = 11
Arr[10] = 1
Отсортированный массив:
0 1 3 4 5 7 9 11 15 24 _
```

Сравнение анализ времени выполнения:

График для Heap Sort O(n log n)



Heap Sort

Heap Sort имеет асимптотическую сложность O(n log n) в худшем, среднем и лучшем случаях. Он эффективен и не требует дополнительной памяти, кроме константной.

Quick Sort

Quick Sort в среднем имеет асимптотическую сложность O(n log n), но в худшем случае может достигать O(n^2). Он обычно быстрее Heap Sort на практике, так как имеет меньший коэффициент скрытой константы. Однако Quick Sort требует дополнительной памяти для рекурсии.

Merge Sort

Merge Sort имеет асимптотическую сложность O(n log n) в худшем, среднем и лучшем случаях. Он эффективен и устойчив, но требует дополнительной памяти для хранения вспомогательных массивов.

Анализ времени выполнения:

1. **Маленький объем данных (несколько элементов):**

- Heap Sort: Быстр, но может быть немного медленнее Quick Sort из-за дополнительных операций.
- Quick Sort: Очень быстр для маленьких массивов.
- Merge Sort: Эффективен, но может быть избыточен для небольших наборов данных из-за дополнительной памяти.

2. **Средний объем данных (несколько тысяч элементов):**

- Heap Sort: Эффективен и стабилен, хорош для средних наборов данных.
- Quick Sort: Быстрый и обычно превосходит другие методы в этом диапазоне.
- Merge Sort: Стабилен, но может быть чуть медленнее Quick Sort.

3. **Большой объем данных (десятки тысяч и более элементов):**

- Heap Sort: Может быть менее эффективным из-за константного коэффициента, но всё равно приемлем для больших наборов данных.
- Quick Sort: Быстрый, но может столкнуться с худшим случаем.
- Merge Sort: Стабилен и эффективен, но требует дополнительной памяти.

Выводы:

Если важна стабильность сортировки и память не является критическим ресурсом, то лучше выбрать Merge Sort.

Quick Sort хорош для средних и маленьких наборов данных, но важно учесть возможность худшего случая.

Heap Sort обычно обеспечивает стабильную производительность, особенно на больших наборах данных, но может быть немного медленнее для небольших данных.

Даны массивы A[1…n] и B[1…n]. Мы хотим вывести все n2 сумм вида A[i]+B[j] в возрастающем порядке. Наивный способ — создать массив, содержащий все такие суммы, и отсортировать его. Соответствующий алгоритм имеет время работы O(n2logn) и использует O(n2) памяти. Приведите алгоритм с таким же временем работы, который использует линейную память.

```
using System;

class Program
{
   static void Main()
```

```
    {
        int[] A = { 1, 4, 7 };
        int[] B = { 2, 3, 5 };

        PrintSortedSums(A, B);
    }

    static void PrintSortedSums(int[] A, int[] B)
    {
        Array.Sort(A);
        Array.Sort(B);

        int n = A.Length;

        for (int i = 0; i < n; i++)
        {
            int a = A[i];
            int j = 0;

            while (j < n && B[j] <= a)
            {
                j++;
            }

            for (; j < n; j++)
            {
                Console.WriteLine(a + B[j]);
            }
        }
    }
}
```