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) | |
|  | |  | |

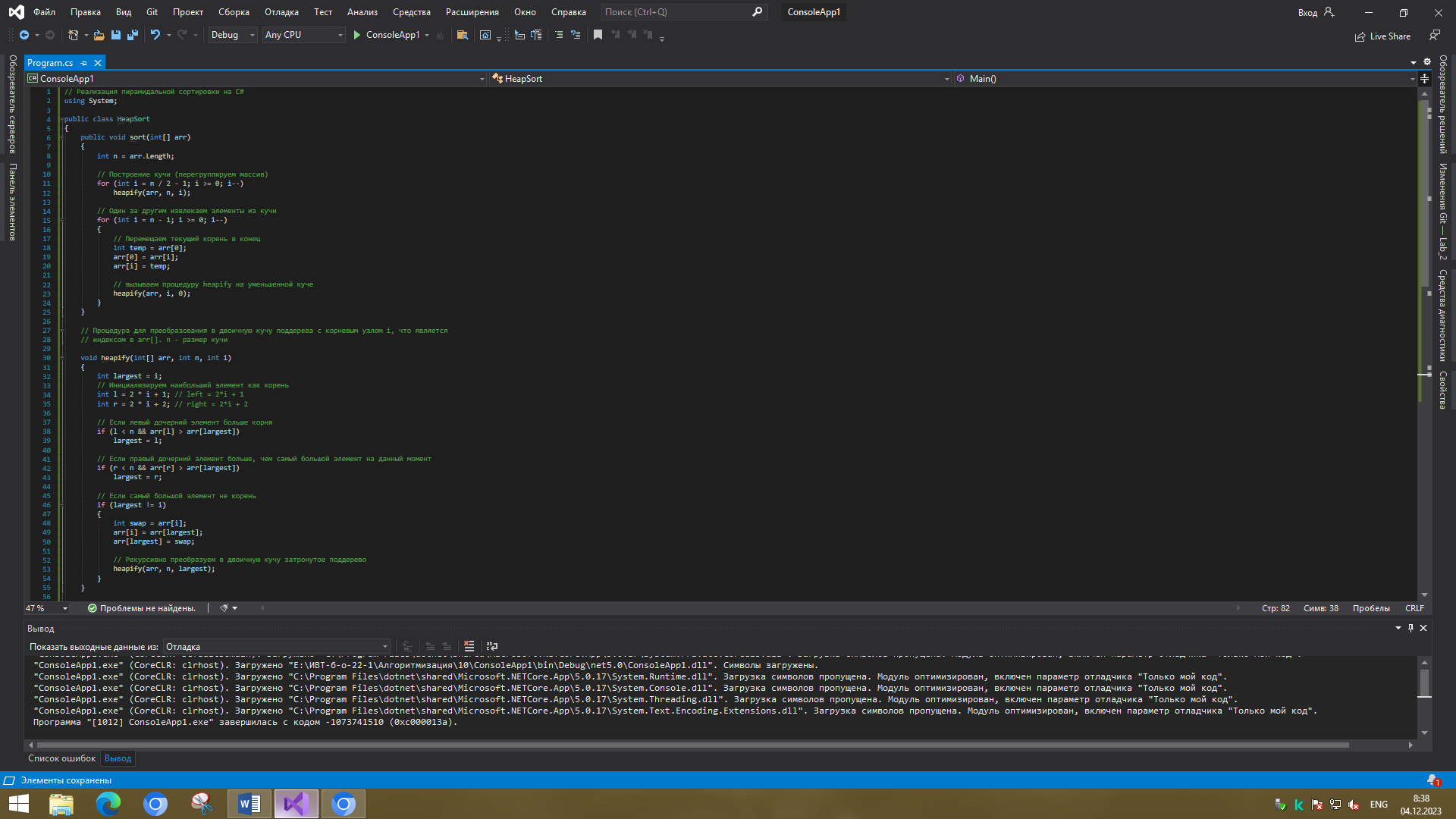
Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_ Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ставрополь, 2023 г.

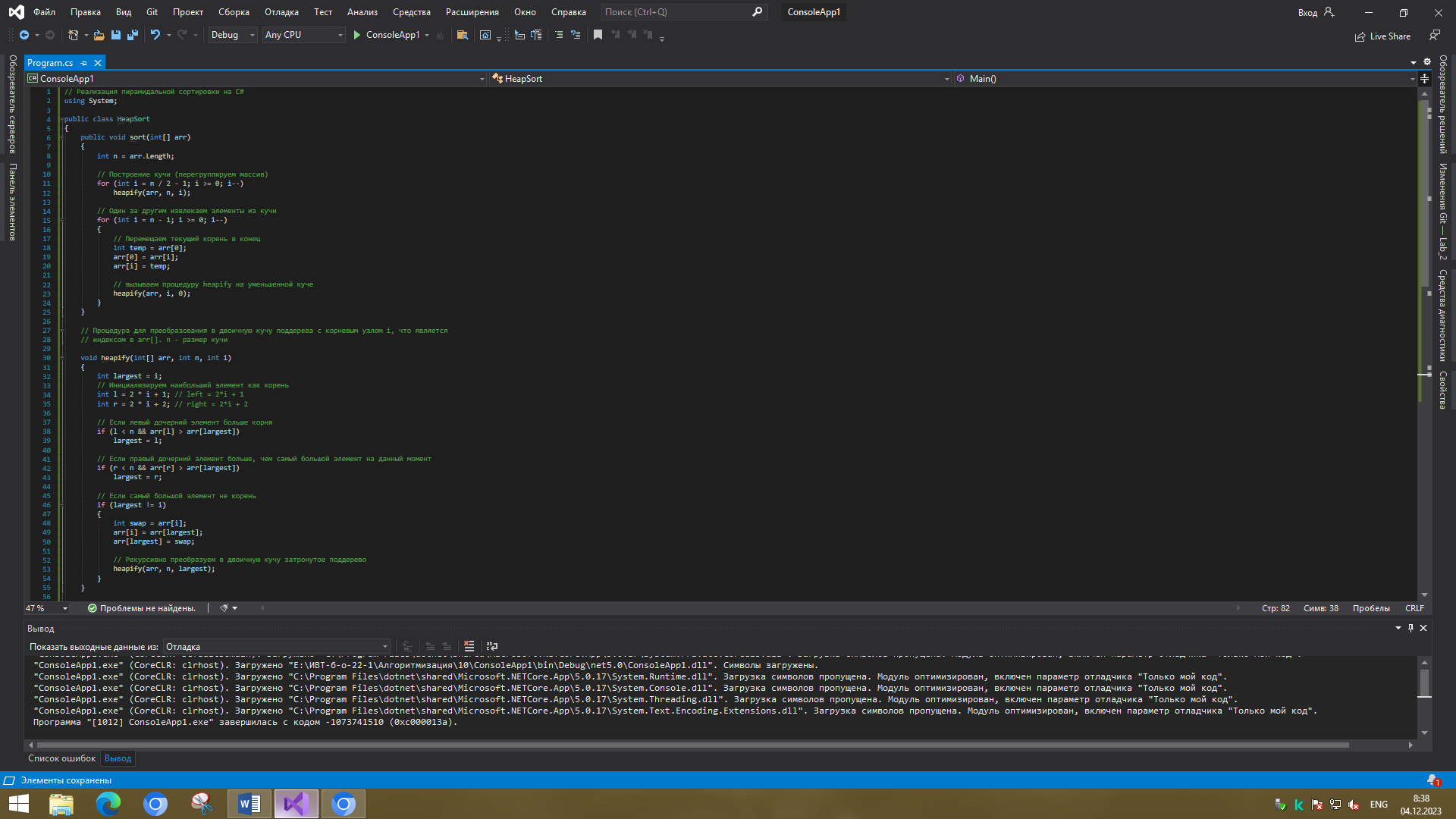
**Порядок выполнения работы:**

Алгоритм сортировки кучей Heap Sort:

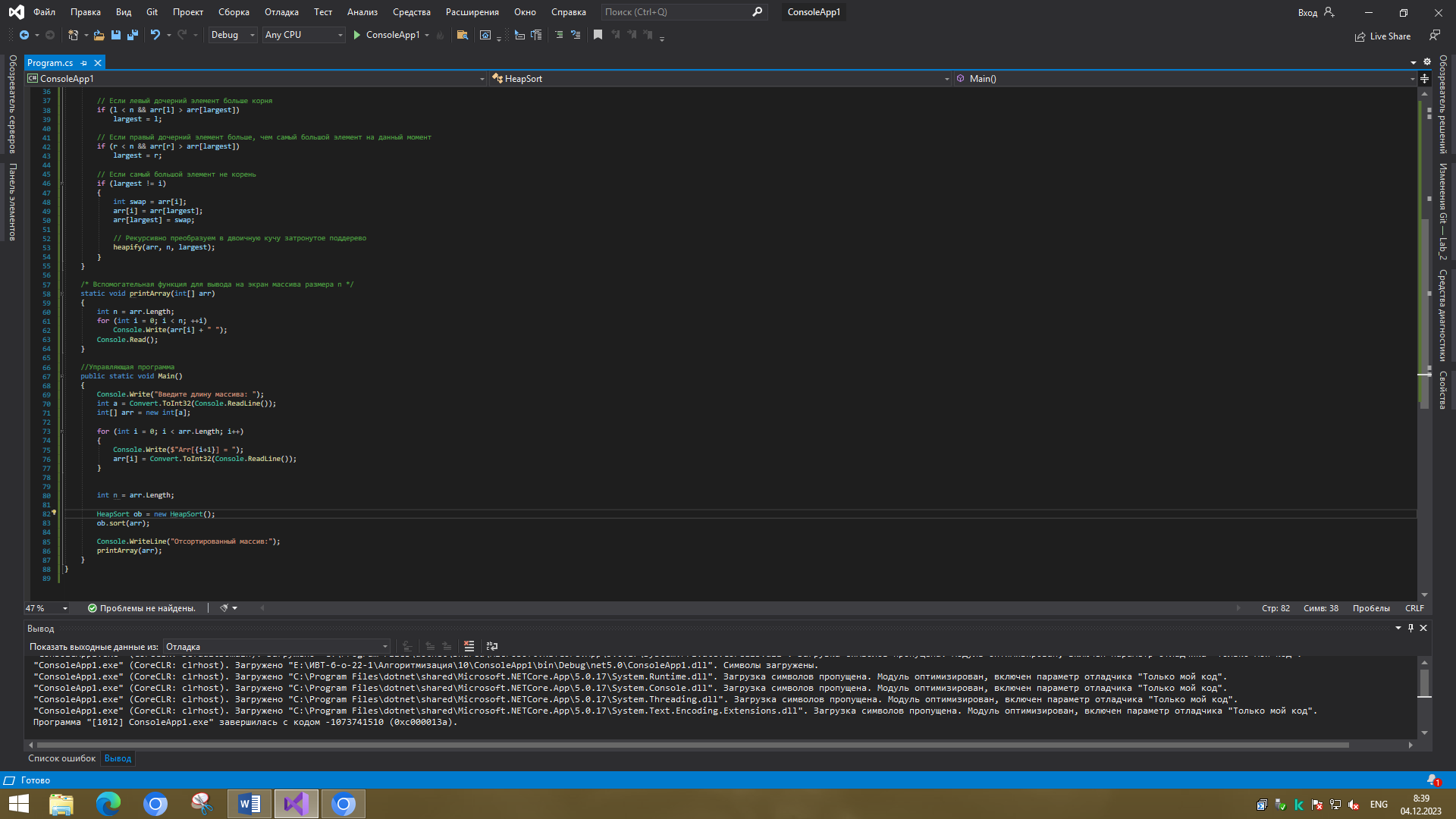
Процедура сортировки



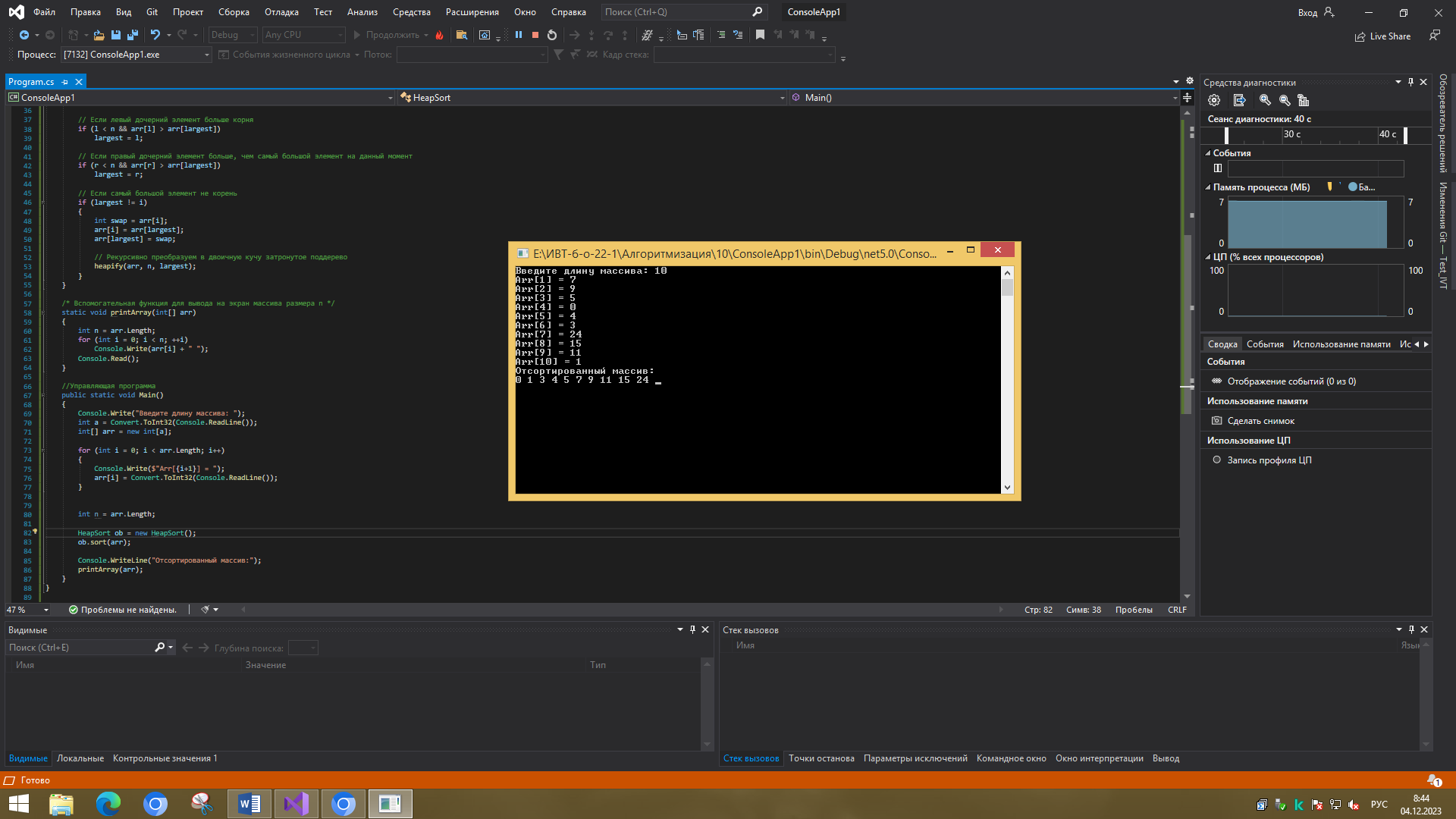
Процедура преобразования в кучу



Метод Main и вывод массива



Результат работы программы:



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]);

}

}

}

}