

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 ЛАБОРАТОРНОЙ РАБОТЕ №10
дисциплины «Алгоритмизация»
Вариант 29

Выполнил:
Саенко Андрей Максимович
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», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., канд. технических
наук, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____
Ставрополь, 2023 г.

Тема: Алгоритм сортировки кучей

Порядок выполнения работы:

1. Реализация алгоритма Heap Sort.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import timeit
import matplotlib.pyplot as plt
import random
import numpy as np
from scipy.optimize import curve_fit

def find_coeffs_bin(x, time):
    params, _ = curve_fit(n_log_n, np.array(x),
                          np.array(time))
    a, b = params
    return a, b

def n_log_n(x, a, b):
    return a * x * np.log(x) + b

def heapify(nums, n, i):
    largest = i
    left = 2*i + 1
    right = 2*i + 2

    if left < n and nums[i] < nums[left]:
        largest = left

    if right < n and nums[largest] < nums[right]:
        largest = right

    if largest != i:
        nums[i], nums[largest] = nums[largest], nums[i]
        heapify(nums, n, largest)

def heap_sort(nums):
    n = len(nums)

    for i in range(n // 2 - 1, -1, -1):
        heapify(nums, n, i)
```

```

for i in range(n - 1, 0, -1):
    nums[i], nums[0] = nums[0], nums[i]
    heapify(nums, i, 0)

```

```

def random_list(n):
    arr = []

    for i in range(1,n):
        arr.append(random.randint(0,500))

    return arr

```

```

def good_list(n):
    return [i for i in range(1,n)]

```

```

def bad_list(n):
    return [i for i in range(n-1,0, -1)]

```

```

if __name__ == "__main__":
    x = []
    heap_sort_bad = []
    heap_sort_rnd = []
    heap_sort_good = []

    for i in range(1,301):
        x.append(i)
        arr = random_list(i)
        sort_time = (timeit.timeit(lambda: heap_sort(arr),
                                   number=50))/50
        heap_sort_rnd.append(sort_time)

        arr = bad_list(i)
        sort_time = (timeit.timeit(lambda: heap_sort(arr),
                                   number=50))/50
        heap_sort_bad.append(sort_time)

        arr = good_list(i)
        sort_time = (timeit.timeit(lambda: heap_sort(arr),
                                   number=50))/50
        heap_sort_good.append(sort_time)

    plt.figure(1)
    a, b = find_coeffs_bin(x, heap_sort_rnd)
    y = n_log_n(np.array(x), a, b)
    plt.plot(x, y, color='red')
    plt.title("Время сортировки при\n"+
              "заполнении массива рандомными элементами")
    plt.scatter(x,heap_sort_rnd,s=3)

```

```

plt.xlabel('Размер массива')
plt.legend(['f"y = {a}*x*log(x)" +
           f" + ({b})"]])
plt.ylabel("Время сортировки массива")

plt.figure(2)
a, b = find_coeffs_bin(x, heap_sort_bad)
y = n_log_n(np.array(x), a, b)
plt.plot(x, y, color='red')
plt.title("Время сортировки в\n"+
          "худшем случае")
plt.scatter(x, heap_sort_bad, s=3)
plt.xlabel('Размер массива')
plt.legend(['f"y = {a}*x*log(x)" +
           f" + ({b})"]])
plt.ylabel("Время сортировки массива")

plt.figure(3)
a, b = find_coeffs_bin(x, heap_sort_good)
y = n_log_n(np.array(x), a, b)
plt.plot(x, y, color='red')
plt.title("Время сортировки\n"+
          " в лучшем случае")
plt.scatter(x, heap_sort_good, s=3)
plt.xlabel('Размер массива')
plt.legend(['f"y = {a}*x*log(x)" +
           f" + ({b})"]])
plt.ylabel("Время сортировки массива")

plt.show()

```

Результат работы программы:

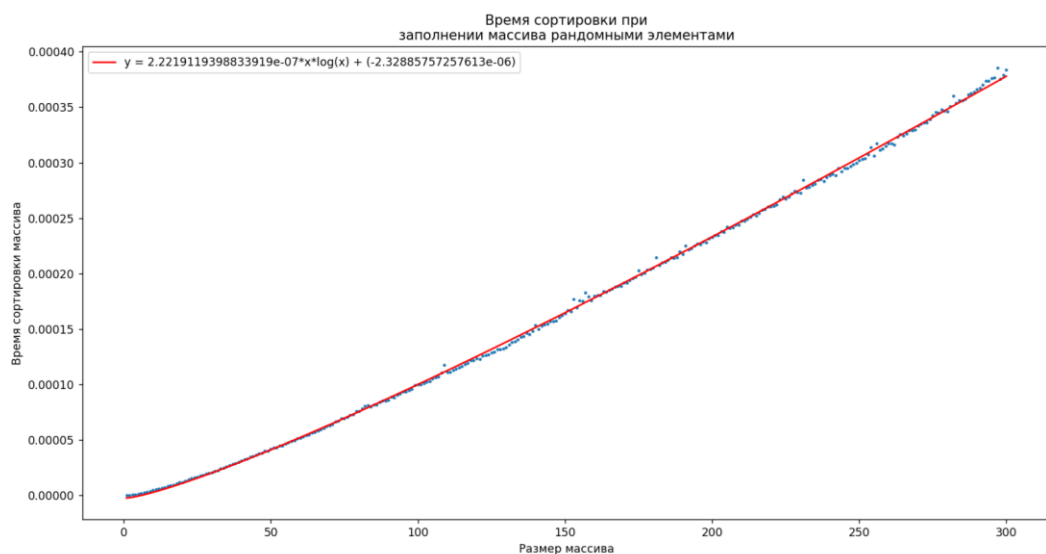


Рисунок 1 – Время работы программы при заполнении массива случайными числами

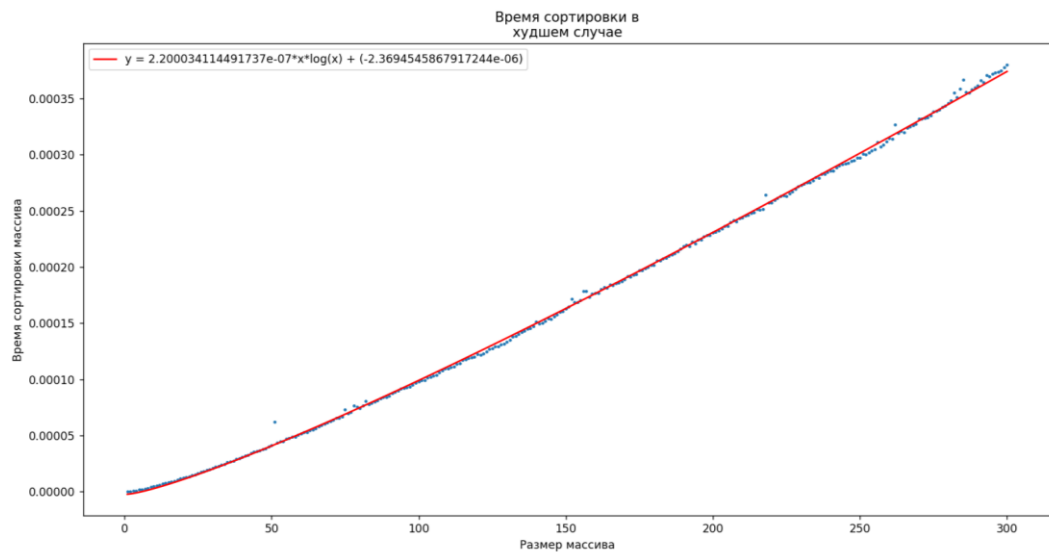


Рисунок 2 – Время работы программы в худшем случае

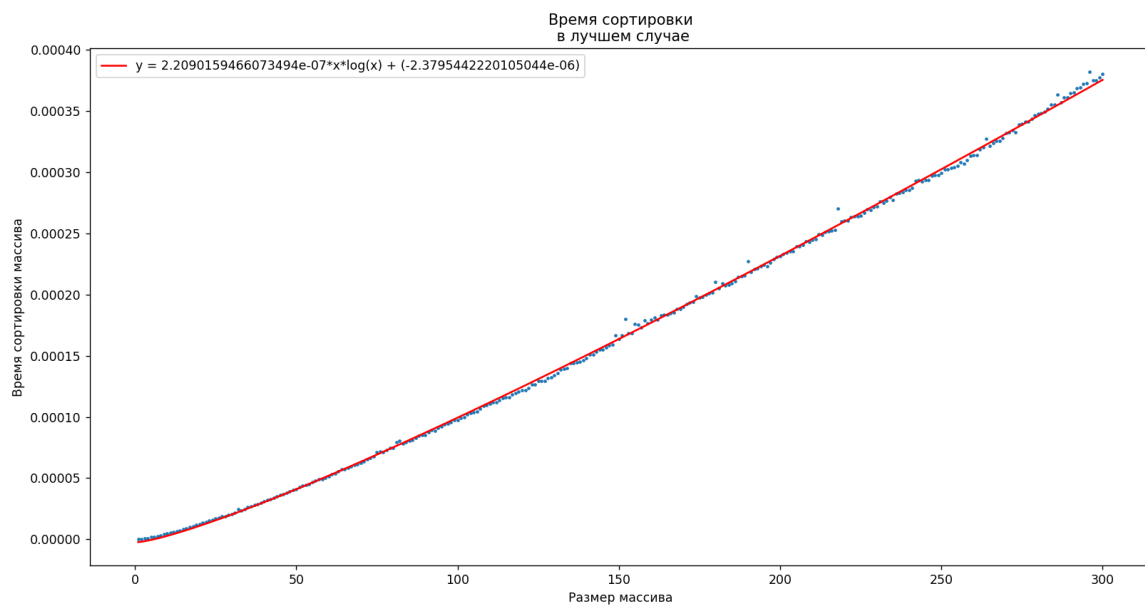


Рисунок 3 – Время работы программы в лучшем случае

2. Сравнение с другими сортировками:

Случай	Худший	Средний	Лучший
Heap Sort	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$
Quick Sort	$O(n^2)$	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$
Merge Sort	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$

Алгоритм Heap Sort не является стабильным, но обладает сложностью $O(n \cdot \log(n))$ во всех случаях, что позволяет эффективно сортировать большие объёмы данных. Этот алгоритм использует относительно небольшое количество памяти.

Алгоритм Merge Sort является стабильным и обладает сложностью $O(n \cdot \log(n))$ во всех случаях, но требует относительно много памяти, что является явным недостатком при сортировке больших объемов данных.

Алгоритм Quick Sort в лучшем и среднем случае обладает сложностью $O(n \cdot \log(n))$, но в худшем случае его сложность равна $O(n^2)$, что делает его неэффективным в некоторых случаях.

3. Оптимизация алгоритма

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import timeit
import matplotlib.pyplot as plt
import random
import numpy as np
from scipy.optimize import curve_fit
import heapq

def find_coeffs_bin(x, time):
    params, _ = curve_fit(n_log_n, np.array(x),
                          np.array(time))
    a, b = params
    return a, b

def n_log_n(x, a, b):
    return a*x*np.log(x) + b

def heapify(nums, n, i):
    end = n
    new = nums[i]
    child = 2*i + 1

    while child < end:
        right = child + 1
```

```

        if right < end and not nums[right] < nums[child]:
            child = right

        if nums[i] < nums[child]:
            nums[i] = nums[child]
            i = child
            child = 2*i + 1

        else:
            break

    nums[i] = new

def heap_sort_upgrade(nums):
    n = len(nums)
    heapq._heapify_max(nums)

    for i in range(n-1, 0, -1):
        nums[i], nums[0] = nums[0], nums[i]
        heapify(nums, i, 0)

def random_list(n):
    arr = []

    for i in range(1,n):
        arr.append(random.randint(0,500))

    return arr

def good_list(n):
    return [i for i in range(1,n)]

def bad_list(n):
    return [i for i in range(n-1,0, -1)]

if __name__ == "__main__":
    x = []
    heap_sort_bad = []
    heap_sort_rnd = []
    heap_sort_good = []

    for i in range(1,301):
        x.append(i)
        arr = random_list(i)
        sort_time = (timeit.timeit(lambda: heap_sort_upgrade(arr),
                                    number=50))/50
        heap_sort_rnd.append(sort_time)

```

```

arr = bad_list(i)
sort_time = (timeit.timeit(lambda: heap_sort_upgrade(arr),
                        number=50))/50
heap_sort_bad.append(sort_time)

```

```

arr = good_list(i)
sort_time = (timeit.timeit(lambda: heap_sort_upgrade(arr),
                        number=50))/50
heap_sort_good.append(sort_time)

```

```

plt.figure(1)
a, b = find_coeffs_bin(x, heap_sort_rnd)
y = n_log_n(np.array(x), a, b)
plt.plot(x, y, color='red')
plt.title("Время сортировки при\n"+
          "заполнении массива рандомными элементами")
plt.scatter(x, heap_sort_rnd, s=3)
plt.xlabel('Размер массива')
plt.legend([f'y = {a}*x*log(x)" +
            f' + ({b})"'])
plt.ylabel("Время сортировки массива")

```

```

plt.figure(2)
a, b = find_coeffs_bin(x, heap_sort_bad)
y = n_log_n(np.array(x), a, b)
plt.plot(x, y, color='red')
plt.title("Время сортировки в\n"+
          "худшем случае")
plt.scatter(x, heap_sort_bad, s=3)
plt.xlabel('Размер массива')
plt.legend([f'y = {a}*x*log(x)" +
            f' + ({b})"'])
plt.ylabel("Время сортировки массива")

```

```

plt.figure(3)
a, b = find_coeffs_bin(x, heap_sort_good)
y = n_log_n(np.array(x), a, b)
plt.plot(x, y, color='red')
plt.title("Время сортировки\n"+
          "в лучшем случае")
plt.scatter(x, heap_sort_good, s=3)
plt.xlabel('Размер массива')
plt.legend([f'y = {a}*x*log(x)" +
            f' + ({b})"'])
plt.ylabel("Время сортировки массива")

```

```

plt.show()

```


Результат работы программы:

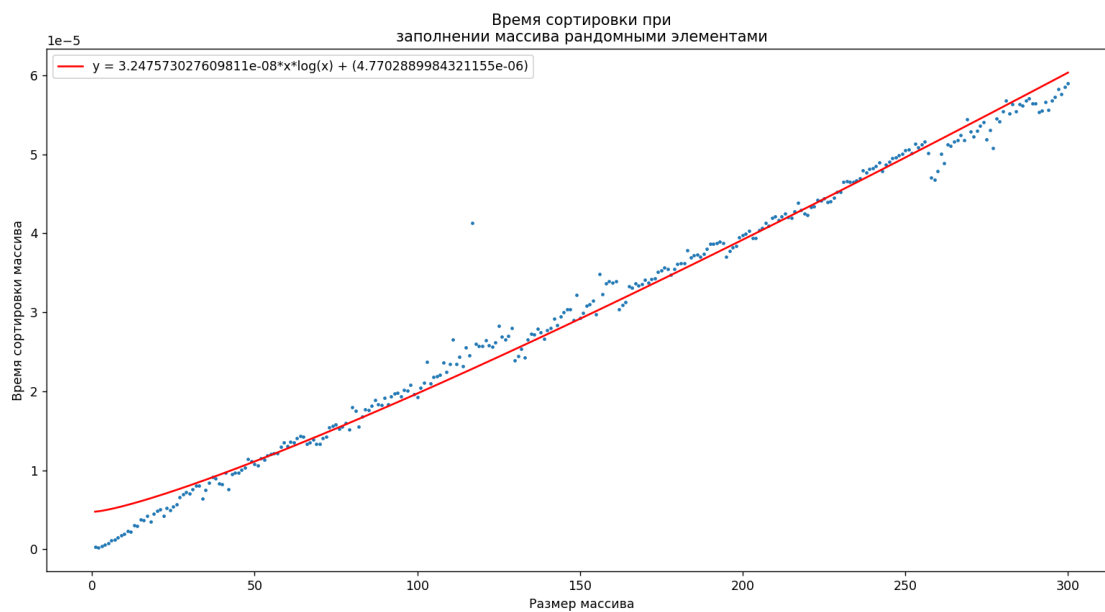


Рисунок 4 – Время работы программы при заполнении массива случайными числами

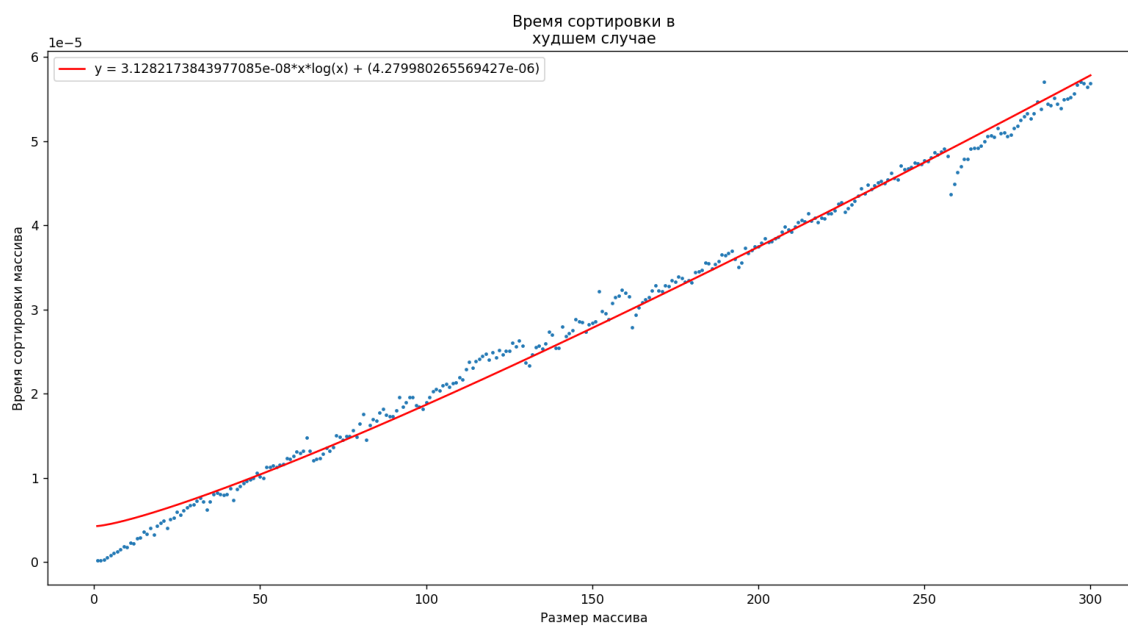


Рисунок 5 – Время работы программы в худшем случае

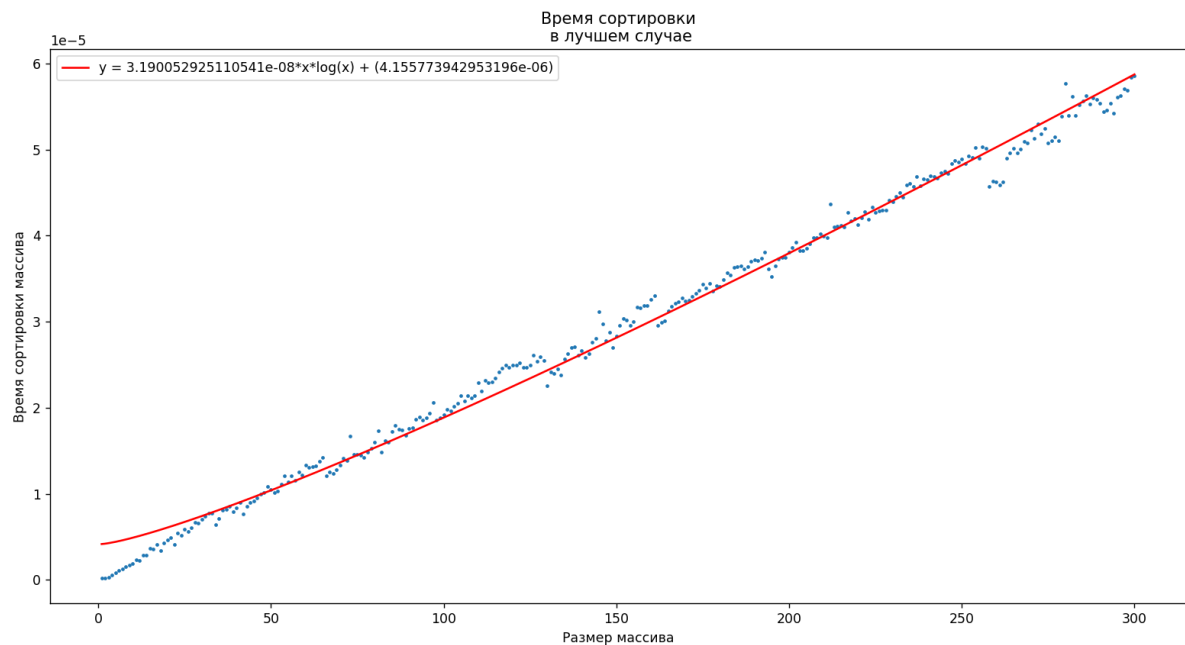


Рисунок 6 – Время работы программы в лучшем случае

4. Применение в реальной жизни.

Данный алгоритм сортировки может применяться при работе с базами данных, также сортировка кучей активно применяется в ядре Linux. Данный алгоритм может быть предпочтительным выбором в некоторых ситуациях из-за относительно высокой скорости работы и из-за малого использования памяти.

5. Анализ сложности: Проведите анализ времени выполнения и пространственной сложности алгоритма Heap Sort. Исследуйте, как эти характеристики зависят от размера входных данных. Сделайте выводы о том, в каких случаях Heap Sort может быть более или менее эффективным по сравнению с другими алгоритмами сортировки.

Алгоритм Heap Sort имеет временную сложность $O(n \cdot \log(n))$ в лучшем, худшем и среднем случаях. Пространственная сложность алгоритма составляет $O(1)$, т.к. он выполняется на месте, без использования дополнительной памяти.

Время выполнения алгоритма Heap Sort растет пропорционально $O(n \cdot \log(n))$, что делает его эффективным для больших наборов данных.

Также следует учитывать, что алгоритм Heap Sort имеет некоторые ограничения на типы данных, которые могут быть отсортированы им. Например, он не может работать со строками или другими несравнимыми типами данных.

В целом, алгоритм Heap Sort является эффективным для сортировки больших наборов данных, особенно если данные уже содержатся в виде двоичной кучи.

6. Даны массивы $A[1...n]$ и $B[1...n]$. Мы хотим вывести все n^2 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 $O(n^2 \log n)$ и использует $O(n^2)$ памяти. Приведите алгоритм с таким же временем работы, который использует линейную память.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def f(A, B):
    n = len(A)
    A.sort()
    B.sort()
    result = []
    heap = []
    heap.append((A[0] + B[0], 0, 0))

    for _ in range(n**2 - 1):
        sum, i, j = heap.pop(0)
        result.append(sum)

        if j < n - 1:
            heap.append((A[i] + B[j+1], i, j+1))

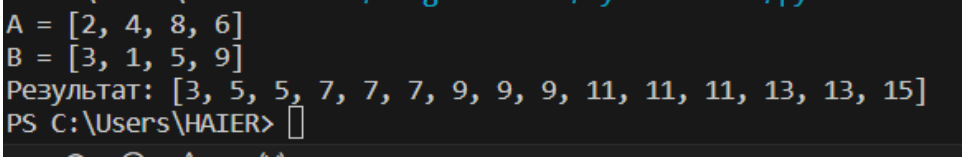
        if j == 0 and i < n - 1:
            heap.append((A[i+1] + B[j], i+1, j))

    heap.sort()

    return result
```

```
if __name__ == "__main__":  
    A = [2, 4, 8, 6]  
    B = [3, 1, 5, 9]  
    print(f'A = {A}\nB = {B}')  
    print(f'Результат: {f(A, B)}')
```

Результат работы программы:



```
A = [2, 4, 8, 6]  
B = [3, 1, 5, 9]  
Результат: [3, 5, 5, 7, 7, 7, 9, 9, 9, 11, 11, 11, 13, 13, 15]  
PS C:\Users\HAIER>
```

Рисунок 7 – Результат работы программы

Вывод

В ходе выполнения лабораторной работы был изучен алгоритм сортировки кучей, проведено сравнение времени работы обычного алгоритма и его улучшенной версии, проведено сравнение данного алгоритма с другими алгоритмами сортировки.