МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет»

Кафедра инфокоммуникаций

Отчет по лабораторной работе №10 по дисциплине «Алгоритмизация»

Выполнил студент группы ИВТ-б-о-22-1
Пушкин Н.С. « »_____20__г.
Подпись студента_____
Работа защищена « »_____20__г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

Порядок выполнения работы:

Алгоритм сортировки кучей Heap Sort:
Процедура сортировки

```python
def heap_sort(arr):
    n = len(arr)

    # Построение max-heap
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    # Извлечение элементов из кучи по одному
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]   # меняем элементы местами
        heapify(arr, i, 0)
```

Процедура преобразования в кучу

```python
def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < n and arr[l] > arr[largest]:
        largest = l

    if r < n and arr[r] > arr[largest]:
        largest = r

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]  # меняем элементы местами
        heapify(arr, n, largest)
```

Результат работы программы:

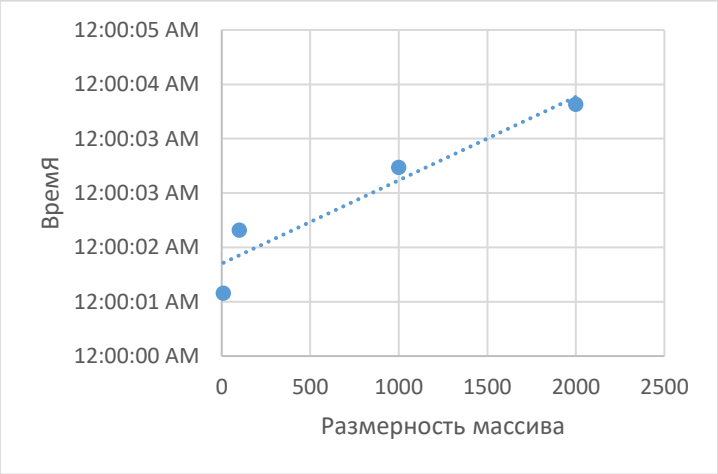
```
C:\Users\Никита\Desktop\alg_lab_10\venv\Script
Исходный массив:
[4, 12, 7, 13, 18, 15, 14, 4, 18, 17]

Отсортированный массив:
[4, 4, 7, 12, 13, 14, 15, 17, 18, 18]

Process finished with exit code 0
```

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.

```
def main():
    A = [1, 4, 7]
    B = [2, 3, 5]
    print_sorted_sums(A, B)

def print_sorted_sums(A, B):
    A.sort()
    B.sort()
    n = len(A)

    for i in range(n):
        a = A[i]
        j = 0

        while j < n and B[j] <= a:
            j += 1

        for k in range(j, n):
            print(a + B[k])

if __name__ == "__main__":
    main()
```