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»

Выполнила:
Кубанова Ксения Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: алгоритм heap sort

Цель: проанализировать алгоритм heap sort и сравнить его с работой других алгоритмов

Порядок выполнения работы

1. **Реализация алгоритма Heap Sort:**

Спроектируйте и реализуйте алгоритм сортировки кучей (Heap Sort) на любом удобном для вас языке программирования. После этого протестируйте вашу реализацию на различных видах входных данных, таких как отсортированный массив, массив в обратном порядке и случайный массив. Оцените алгоритм в каждом случае.

Отсортированный массив:

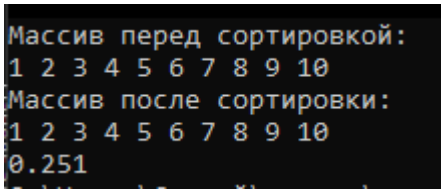

Рисунок 1 – heapsort отсортированный массив

Массив в обратном порядке:

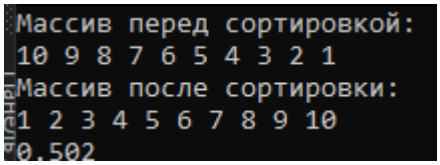

Рисунок 2 – heapsort массив в обратном порядке

Массив в случайном порядке:

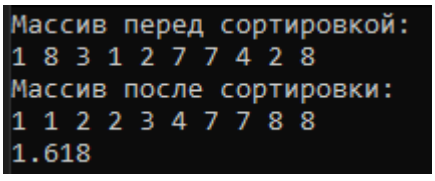

Рисунок 3 – heapsort случайный массив

Реализация алгоритма предоставлена в файле *heapsort10.cpp*.

2. Сравнение с другими сортировками:

Сравните производительность Heap Sort с другими известными алгоритмами сортировки, такими как Quick Sort и Merge Sort. Проведите

анализ времени выполнения, используя различные размеры входных данных. Сделайте выводы о преимуществах и недостатках каждого метода.

Quick Sort

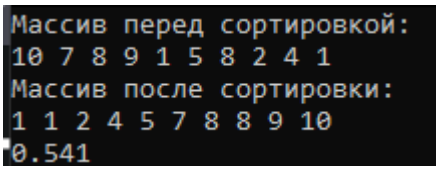

Рисунок 4 – quick sort 10 элементов

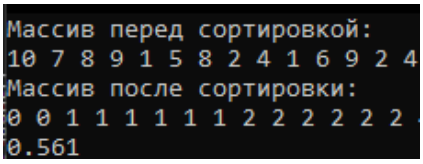

Рисунок 5 – quick sort 50 элементов

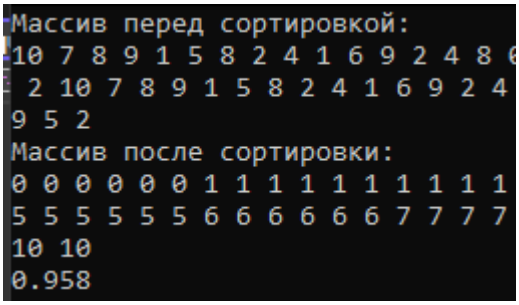

Рисунок 6 – quick sort 100 элементов

Реализация алгоритма предоставлена в файле *quicksort10.cpp*.

Merge Sort

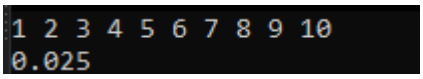

Рисунок 7 – merge sort 10 элементов

Рисунок 8 – merge sort 50 элементов

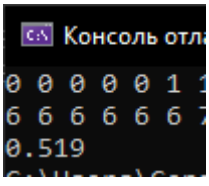

Рисунок 9 – merge sort 100 элементов

Реализация алгоритма предоставлена в файле *mergesort10.cpp*.

3. **Оптимизация алгоритма:**

Попробуйте оптимизировать алгоритм Heap Sort. Исследуйте возможности улучшения производительности, например, путем использования оптимизированных структур данных или алгоритмических подходов. Сравните вашу оптимизированную реализацию с базовой версией.

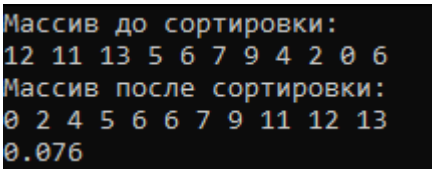

Рисунок 10 – heap sort улучшенный 10 элементов

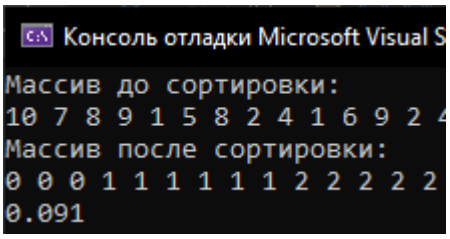

Рисунок 11 – heap sort улучшенный 50 элементов

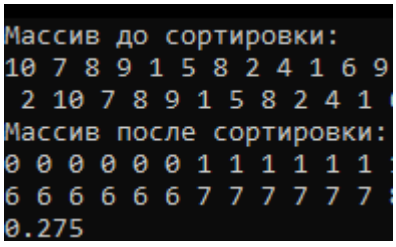

Рисунок 12 – heap sort улучшенный 100 элементов

Реализация алгоритма предоставлена в файле *heapsortup10.cpp*.

После анализа полученных данных выводы были занесены в таблицу для сравнения:

Таблица 1 – сравнение алгоритма Heap Sort с базовой и оптимизированной версией

Алгоритм/кол-во элементов	**Heap Sort (базовая версия)**	**Heap Sort (оптимизированная версия)**
10	0,251	0,076
50	0,285	0,091
100	0,3	0,275

Алгоритм был улучшен за счёт использования вектора из библиотеки STL для хранения элементов массива. Функции heapify и heapSort

остаются неизменными, за исключением параметра, который теперь передается как ссылка на вектор (vector<int>&). Функция printArray также принимает const vector<int>&, чтобы избежать изменения входного вектора.

Таблица 2 – сравнение алгоритмов

Алгоритм / Кол-во элементов	**Quick Sort сек.**	**Merge Sort сек.**	**Heap Sort (оптим.) сек.**
10	0,541	0,025	0,076
50	0,561	0,117	0,091
100	0,958	0,519	0,275

Исходя из данных, приведённых в таблице, можно сделать вывод, что среди всех сравниваемых алгоритмов, алгоритм Heap Sort является оптимально лучшим вариантом.

4. **Применение в реальной жизни:**

Рассмотрите практические применения Heap Sort в реальных сценариях. Например, как этот алгоритм может быть использован для оптимизации работы баз данных, событийной обработки или других вычислительных задач. Объясните, почему Heap Sort может быть предпочтительным выбором в некоторых ситуациях.

Heap Sort может быть полезным во многих сценариях и предоставляет несколько преимуществ, приводящих к его предпочтительному выбору в определенных ситуациях. Например:

- Сортировка большого объёма данных за счёт его временной сложности O(nLogn);
- Ограниченное использование памяти, так как данный алгоритм не требует дополнительного хранения данных в процессе сортировки;
- Эффективность при сортировки случайных данных, опять-таки, за счёт его временной сложности, что делает его стабильным выбором в ситуации со случайными данными.

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.

Как выяснилось ранее, алгоритм Heap Sort имеет временную сложность O(nlogn) в лучшем, среднем и худшем случаях. Пространственная сложность составляет O(1), так как сортировка выполняется на месте, без дополнительных структур данных.

Зависимость времени выполнения от размера входных данных (n) обусловлена построением и реконструкцией кучи. В худшем случае алгоритм Heap Sort несколько медленнее, чем алгоритмы с линейной временной сложностью, такие как Quick Sort или Merge Sort. Однако, Heap Sort имеет преимущество в пространственной сложности, так как не требует дополнительной памяти для хранения временных данных, в отличие от Quicksort, например.

Heap Sort может быть менее эффективным по сравнению с другими алгоритмами сортировки, такими как Quick Sort или Merge Sort, в случаях:

- Когда массив уже предварительно отсортирован или близок к отсортированному состоянию, так как Heap Sort все равно выполняет построение и реконструкцию кучи (видно в первом задании при вводе уже отсортированного массива);
- Когда требуется сортировать небольшие массивы данных, так как у Heap Sort имеется некоторая накладная расходы на построение и реконструкцию кучи.

6. Использование алгоритма с линейной памятью:

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.

Для реализации алгоритма с линейной памятью будет использован модифицированный алгоритм Merge Sort со временем работы O(n2 Log n):

Вывод кода, реализованного в файле *mergesort610.cpp*:

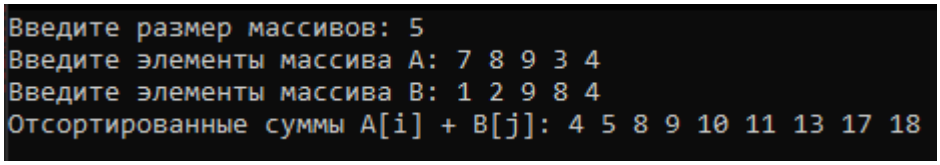

Рисунок 13 – вывод кода

Вывод: Heap Sort, Merge Sort и Quick Sort являются эффективными алгоритмами сортировки, каждый из которых имеет свои преимущества и недостатки. Однако, существует несколько аспектов, в которых Heap Sort может быть предпочтительным выбором перед Merge Sort и Quick Sort.