# Лабораторна робота № 4

# Алгоритми сортування.

Філіпенко Д.

* **Постановка задачі.**

***Завдання:*** Розробити програму на мові С/С++, яка реалізує три алгоритми сортування масивів:

1. за методом «поганого» сортування;
2. за методом «ефективного» сортування;
3. за методом пірамідального сортування (з використанням структури даних «піраміда»)

Третій метод є обов’язковим для всіх варіантів. Вибір перших двох алгоритмів для реалізації – за варіантами в таблиці:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Шейкерне сортування | Сортування вибором | Сортування вставками | Сортування бульбашкою |
| Швидке сортування | 1 | 4 | 7 | 10 |
| Сортування злиттям | 2 | 5 | 8 | 11 |
| Сортування підрахунком | 3 | 6 | 9 | 12 |

Варіант обирається за номером студента у списку групи.

Вихідний масив – масив цілих чисел, який завантажується із текстового файлу. Для формування текстового файлу написати окрему програму (або функцію у меню програми). Числа для запису в файл обираються випадковим чином з діапазону, який вводиться по запиту з клавіатури.

Результуючий відсортований масив записати в інший файл (на екран результуючий масив виводити не треба).

Розглянути тестові приклади з розмірами масивів 100, 500, 1000, 5000, 10000, 30000, 50000 елементів. Програмно визначити час роботи кожного з алгоритмів на кожному з тестових прикладів з різними розмірами масивів. *Увага*: час має вимірюватися тільки для роботи алгоритму сортування, туди не має включатися час на введення або виведення масиву.

Для кожного з розмірів масиву розглянути три випадки вихідних даних:

1. масив випадкових чисел – середній випадок;
2. відсортований масив – найкращий випадок;
3. масив, відсортований у зворотному порядку (за спаданням) – найгірший випадок.

Результати навести в трьох таблицях (для середнього, найкращого та найгіршого випадків) за таким зразком:

*Примітка*. Для коректного порівняння на вхід кожного з трьох реалізованих алгоритмів має поступати один і той самий масив.

За кожною з трьох таблиць побудувати графіки залежності часу роботи алгоритмів від розміру масивів. Навести три рисунки з графіками – для середнього, найкращого та найгіршого випадків. На кожному рисунку – три графіки, що відповідають реалізованим алгоритмам.

Зробити порівняльний аналіз роботи кожного з алгоритмів в середньому, найкращому та найгіршому випадках та порівняльний аналіз реалізованих алгоритмів між собою (в середньому, найкращому та найгіршому випадках вихідних даних). Навести висновки.

* **Опис алгоритмів.**

**Сортування бульбашкою**

 Алгоритм працює таким чином — у масиві порівнюються два сусідні елементи. Якщо один з елементів є більшим, або ж, навпаки, меншим за свого сусіда то ці два елементи міняються місцями

Середній випадок = найгіршому =найкращому= квадратична складність по часу

Найкращий випадок, за умови Айверсона може бути лінійною по часовій складністю

Складність по пам’яті – константна

Алгоритм стійкий

**Сортування підрахунком**

Ідея алгоритму полягає в наступному: спочатку підрахувати скільки разів кожен елемент (ключ) зустрічається в вихідному масиві. Спираючись на ці дані можна одразу вирахувати на якому місці має стояти кожен елемент, а потім за один прохід поставити всі елементи на свої місця.

алгоритм впорядкування, що застосовується при малій кількості різних елементів у масиві даних. Час його роботи лінійно залежить як від загальної кількості елементів у масиві так і від кількості *різних* елементів.

Завжди має складність O(k+n),як по часу так і по пам’яті

Алгоритм стійкий, можна використовувати лише цілі додатні числа.  
Не має операторів порівняння, швидкий

**Пірамідальне сортування**

Алгоритм пірамідального сортування HeapSort використовує представлення масиву у виді дерева. Цей алгоритм не вимагає допоміжних масивів, сортуючи “на місці”

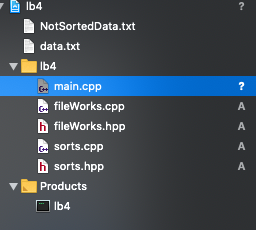
працює в найгіршому, в середньому і в найкращому випадку (тобто гарантовано) за  *О(n* log *n)* операцій при сортуванні *n* елементів. Кількість застосовуваної службової пам'яті не залежить від розміру масиву (тобто, O (1)).

Складність по часу = О(n log n)

Алгоритм не забезпечує стійкості. На упорядкованих даних працює так само як і на випадкових

* **Опис структури програми та особливостей реалізації.**

Програма розділена на 3 файли (головний, функції для роботи за файлами, і функції сортування)



***Реалізовано 6 головних функцій***

1.Заполнить массив

2.Сортировка Подсчетом

3.Сортировка Пузырьком

4.Пирамидальная сортировка

5.Сортировка в обратном порядке (Пузырьком)

6.Замер времени для таблички

В кожній функції присутні функції з підключених файлів

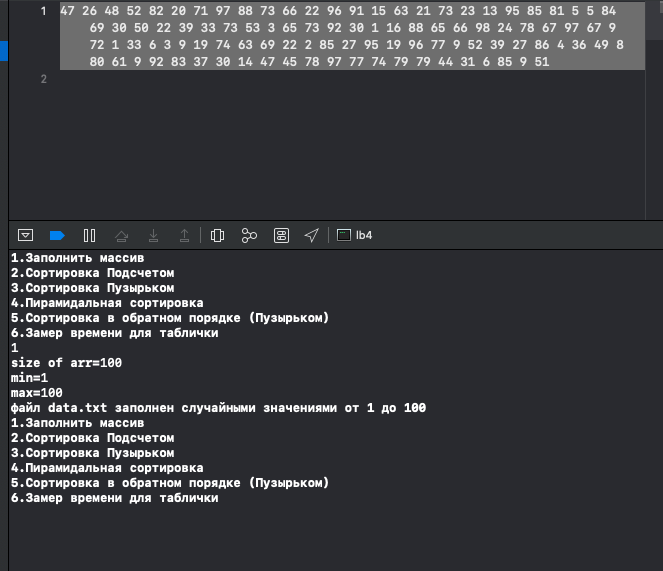
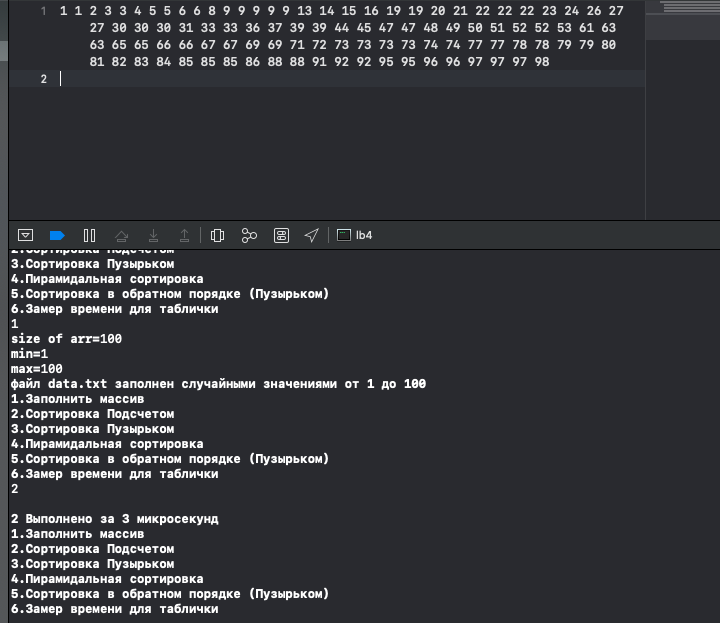
В файлі fileWorks.cpp 3 функції :

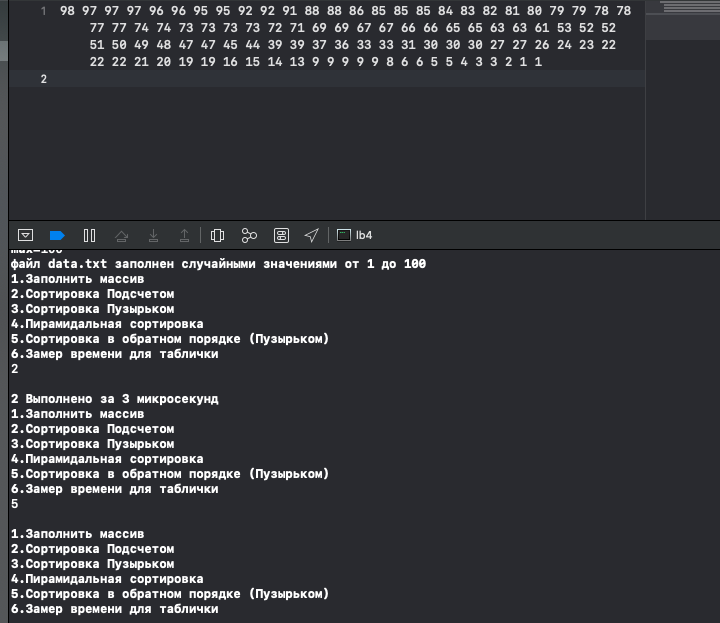
* Заповнення файлу випадковими числами у діапазоні від min до max
* Зчитування елементів з фалу до масиву
* Вивід всіх елементів масиву у файл

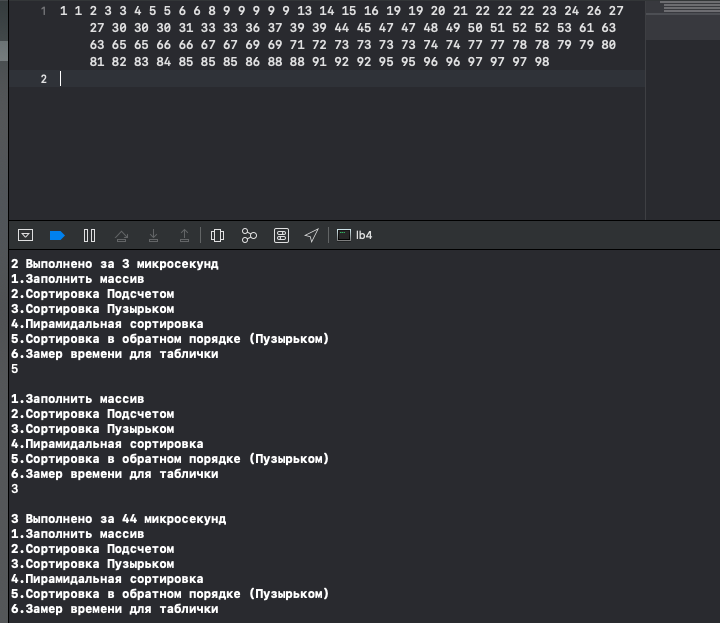
В файлі sorts.cpp 3 основні функції і 2 допоміжні :

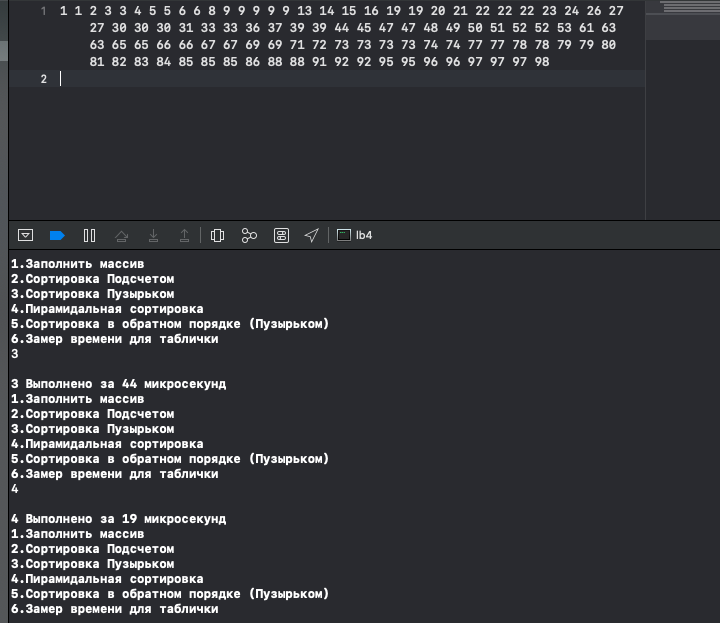
* Сортування підрахунком
* Сортування бульбашкою
* Пірамідальне сортування
* Сортування бульбашкою (обернене)
* Підтримка основної властивості купи у вершині і

***Приклади роботи***

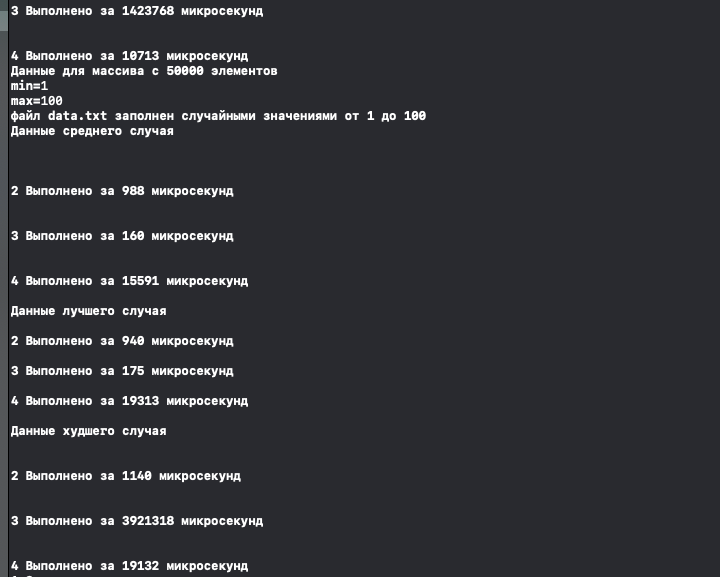
***Заповнення масиву******сортування підрахунком***

***Сортування в зворотному порядку ***

***сортування бульбашкою***

***пірамідальне сортування***

***перевірка функції для підрахунку часу***

******

* Порівняльний аналіз алгоритмів (таблиці, графіки)

***Час роботи алгоритмів сортування в середньому випадку (мікросекунди)***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Розмір  масиву  Алгоритм  сортування | 100 | 500 | 1000 | 5000 | 10000 | 30000 | 50000 |
| Сортування бульбашкою | 39 | 921 | 4819 | **97795** | **372499** | **3346176** | **9267883** |
| Сортування  підрахунком | 2 | 12 | 169 | **148** | **387** | **1202** | **2336** |
| Пірамідальне сортування | 20 | 100 | 298 | **1903** | **3831** | **26549** | **25300** |

***Час роботи алгоритмів сортування в найкращому випадку (мікросекунди)***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Розмір  масиву  Алгоритм  сортування | 100 | 500 | 1000 | 5000 | 10000 | 30000 | 50000 |
| Сортування бульбашкою | 0 | 3 | 3 | **31** | **31** | **90** | **140** |
| Сортування  підрахунком | 2 | 15 | 29 | **107** | **363** | **1293** | **2027** |
| Пірамідальне сортування | 20 | **187** | 345 | **3496** | **5154** | **10834** | **25059** |

***Час роботи алгоритмів сортування в найгіршому випадку (мікросекунди)***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Розмір  масиву  Алгоритм  сортування | 100 | 500 | 1000 | 5000 | 10000 | 30000 | 50000 |
| Сортування бульбашкою | 31 | **823** | **5822** | **96530** | **441567** | **4266532** | **12091773** |
| Сортування  підрахунком | 2 | **12** | 38 | **165** | **296** | **996** | **1782** |
| Пірамідальне сортування | 13 | **214** | **491** | **1512** | **9030** | **11214** | **21633** |

Висновки

Найшвидшим із цих 3 алгоритмів сортування є сортування підрахунком, бо в нього на будь яких входженнях швидкість – лінійна O(k+n), але є мінуси це працює лише на натуральних числах.

Пірамідальне сортування показало однакову швидкість на усіх наборах елементів, тобто це добре коли ми маємо найгірший або середній випадок, але не дуже добре коли ми маємо майже відсортований масив.

Сортування бульбашкою значно програє по часу попереднім алгоритмам через те що цей алгоритм має квадратичну складність і на великих наборах працює помітно довше. Але при найкращому випадку За умови Айверсона ми можемо отримати лінійну складність