**СТРУКТУРА**

**звіту з лабораторної роботи**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені ІГОРЯ СІКОРСЬКОГО”**

**ЗВІТ**

**з лабораторної роботи №3**

**з навчальної дисципліни “Програмування складних алгоритмів”**

**Тема: Методи сортування**

**Варіант №1**

**Виконав студент групи ТР–15**

Руденко Владислав Ігорович

з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

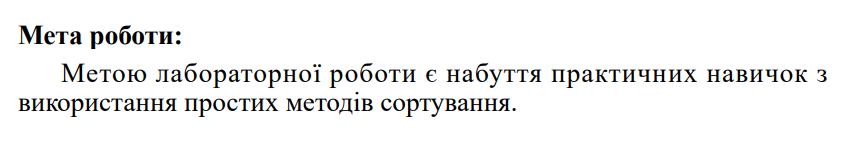
**Перевірив доцент кафедри**

Андрій ОНИСЬКО\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ року

**Київ 2022**

**І. Завдання + Мета:**

  
**Завдання:**1. Провести сортування масивів вказаним методом та у вказаному

порядку. Для тестування алгоритмів сортування масив (10x10, та

більше бажанням).

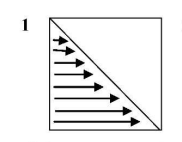
2. Самостійно обрати додатковий метод та провести сортування

того ж масиву.

3. Порівняти кількість перестановок (або час виконання) обох

методів. Спробувати порівняти час виконання сортування з

масивом більшого розміру, який створити за допомогою генератора

випадкових чисел  


Варіант Завдання

**Теоретична частина.**

*Алгоритм сортування* – це алгоритм упорядкування елементів у масиві. Якщо елемент у масиві має кілька полів, поле, що служить критерієм порядку, називається ключем сортування. На практиці як ключ часто виступає число, а в інших полях зберігаються будь-які дані, що не впливають на роботу алгоритму.  
  
Для алгоритму сортування (як і для будь-якого іншого сучасного алгоритму) основними характеристиками є:

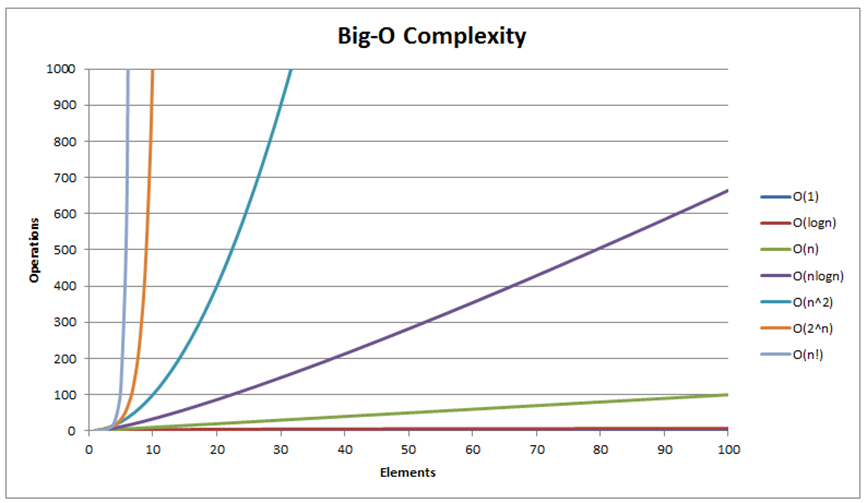
-Час, необхідний на впорядкування n-елементного масиву. Для значної кількості алгоритмів середній і найгірший час впорядкування n-елементного масиву є  — це пов'язано з тим, що в них передбачені перестановки елементів, що стоять поряд (різниця між індексами елементів не перевищує деякого заданого числа). Такі алгоритми зазвичай є стабільними, хоча і не ефективними для великих масивів. Інший клас алгоритмів здійснює впорядкування за час . В цих алгоритмах використовується можливість обміну елементів, що знаходяться на будь-якій відстані один від одного.

-Необхідність додаткової пам'яті для сортування. Зазвичай необхідно O(1) пам'яті.

-Стабільність — [стабільне сортування](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%B1%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Стабільне сортування) не змінює взаємного розташування елементів з однаковими ключами.

*Сортування бульбашкою* - один із найвідоміших алгоритмів сортування. Тут потрібно послідовно порівнювати значення сусідніх елементів і міняти числа місцями, якщо попереднє виявляється більшим за наступне. Таким чином, елементи з великими значеннями виявляються в кінці списку, а з меншими залишаються на початку.

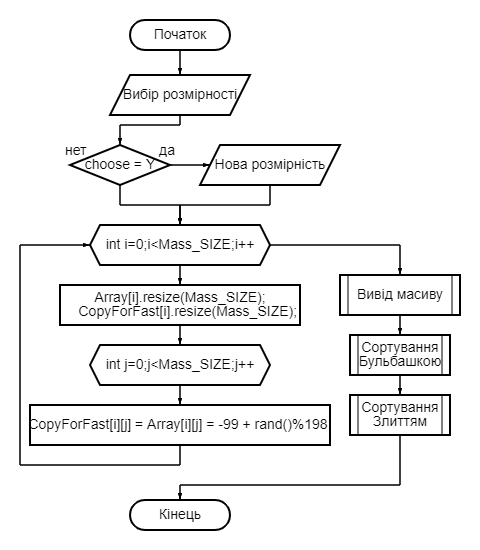
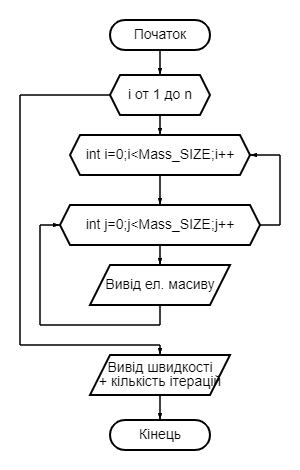
Цей алгоритм вважається навчальним і майже застосовується практично через низьку ефективність: він повільно працює на тестах, у яких маленькі елементи (їх називають «черепахами») стоять наприкінці масиву. Однак на ньому засновані багато інших методів, наприклад, шейкерне сортування та сортування гребінцем.  
  
*Сортування злиттям* — алгоритм сортування, який упорядковує списки (або інші структури даних, доступ до елементів яких можна отримувати лише послідовно, наприклад потоки) в певному порядку. Це сортування – гарний приклад використання принципу «розділяй і владарюй». Спочатку завдання розбивається на кілька підзадач меншого розміру. Потім ці завдання вирішуються за допомогою рекурсивного виклику або безпосередньо, якщо їхній розмір досить малий. Нарешті, їх вирішення комбінуються, і виходить вирішення вихідного завдання.

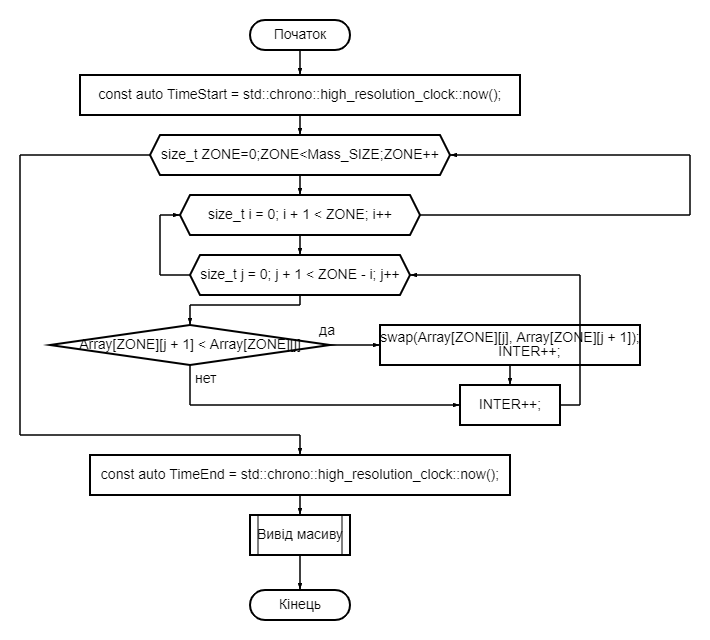
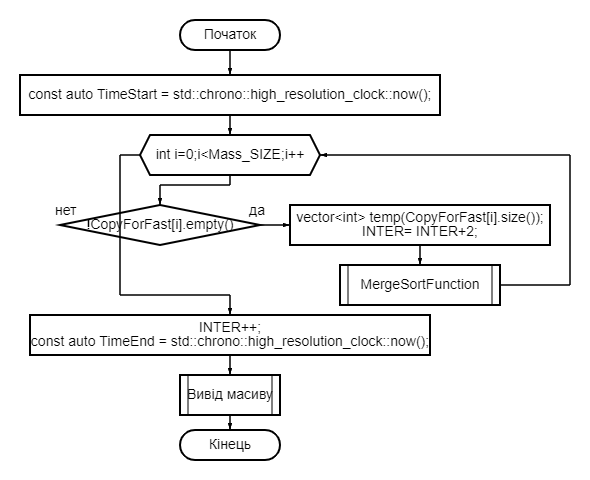


*Найпоширеніші варіанти складності алгоритмів*

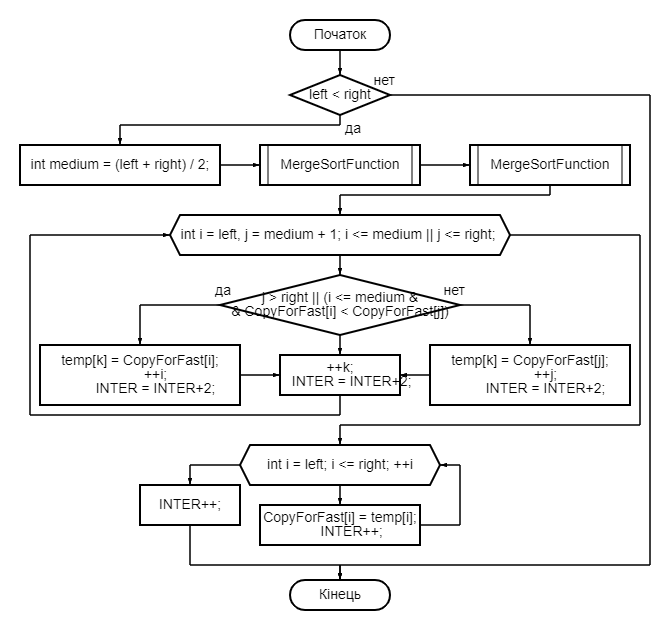
**ІІ. Результати виконання лабораторної роботи.**

**Блок Схеми:**

** **

Блок-Схема 1 (Main - Основна частина) Блок-Схема 2 (Visualization - Вивід маисву)  
  
 

Блок-Схема 3 (Метод бульбашки) Блок-Схема 4 (Метод Злиття - Основна частина)

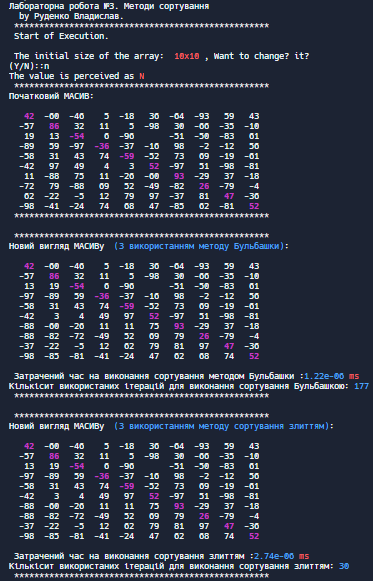


Блок-Схема 5 (Метод Злиття - Функція)

**Опис програми (написана на мові Сі++):** Для виконання завдання було підібрано 2 методи сортування : Бульбашкою та злиттям, що мають різну складність виконання () для Бульбашки та . Основна частина включає в себе декілька основних елементів. Першим є ініціалізація масивів та заповнення їх однаковими значеннями, зроблено це що б поставити обидва методи в однакові умови. Другим елементом є визов необхідних методів. Для економії розміру коду виведення масивів на екран було зроблено окремим методом, що дозволяє методам сортування самостійно викликати та виводити свої результати. Метод злиття включає в себе ще одну підпрограму що виконує обчислення за допомогою рекурсії.

**Результати роботи:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поріврнянн часу виконання двох варіантів розв’язку | | | | |
| Розмір Масиву | Кількість ітерацій (*Бульбашка*)  Складність: | Кількість ітерацій (сортування злиттям) Складність: | Час виконання *(Бульбашка) ms* | Час виконання *(*сортування злиттям*) ms* |
| **10x10** | **178** | **682** | **1.131e-06** | **3.191e-06** |
| **50х50** | **29561** | **32952** | **0.000108809** | **4.877e-05** |
| **100х100** | **243043** | **157977** | **0.00062458** | **0.00018112** |
| **500х500** | **30998090** | **5427237** | **0.068776** | **0.00593139** |
| **1000x1000** | **248807346** | **24228877** | **0.763384** | **0.0250288** |
| **1500x1500** | **840565867** | **57845377** | **2.4322** | **0.103762** |

**Результати роботи у вигляді скріншотів:**

Результати роботи - 1

**Посилання на repl.it:**

**<https://replit.com/join/vhlpaocnfv-hetik>**

**Ш. Висновки.**

В ході виконання лабораторної роботи було ознайомлено з концепцією сортування та розроблено програму що виконує сортування масиву за допомогою 2х методів. При низьких розмірах матриці метод Бульбашкою виявився швидшим та економічнішим в кількості ітерацій, але при збільшені розміру матриці метод Бульбашки програє методу злиття в швидкості та к.ітерацій в значиму зількість раз. Що робить неефективним метод Бульбашки при роботі з великими обсягами даних. Також було виявлено та виправлено декілька помилок.

**Програмний код:***#include <iostream>*

*#include <chrono>*

*#include <vector>*

*using namespace std;*

*using milliseconds = chrono::duration<long long, micro>;*

*#define RED "\x1b[31m"*

*#define BLUE "\x1b[34m"*

*#define PINK "\033[1;35m"*

*#define RESET "\x1b[0m"*

*void Visualization(int Mass\_SIZE, vector<vector<int>>& Array, int lock, float UsedTime,int INTER);*

*void BubbleSort(int Mass\_SIZE, vector<vector<int>>& Array, int lock, int INTER);*

*void MergeSortFunction(vector<int>& CopyForFast, vector<int>& temp, int left, int right,int INTER) ;*

*void MergeSortHEAD(int Mass\_SIZE, vector<vector<int>>& CopyForFast, int lock, int INTER) ;*

*int main() {*

*srand(time(NULL));*

*int Mass\_SIZE = 10, lock = 0, INTER = 0;*

*float UsedTime = 0;*

*char choose;*

*cout << "Лабораторна робота №3. Методи сортування\n by Руденко Владислав."<< "\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n Start of Execution.\n\n The initial size of the array: "<< RED << " 10х10 " << RESET << ", Want to change? it?\n(Y/N)::";*

*cin >> choose;*

*if (choose == 'Y' || choose == 'y')*

*{*

*cout << "Enter a new value"<< RED <<" (ROW length =COLUMN length) "<<RESET<<":\n:: ";*

*cin >> Mass\_SIZE;*

*}*

*else if (choose != 'N' || choose != 'n')*

*cout << "The value is perceived as"<<RED<<" N"<< RESET << endl;*

*vector<std::vector<int> > Array(Mass\_SIZE);*

*vector<std::vector<int> > CopyForFast(Mass\_SIZE);*

*for(int i=0;i<Mass\_SIZE;i++){*

*Array[i].resize(Mass\_SIZE);*

*CopyForFast[i].resize(Mass\_SIZE);*

*for(int j=0;j<Mass\_SIZE;j++)*

*CopyForFast[i][j] = Array[i][j] = -99 + rand()%198;*

*}*

*Visualization(Mass\_SIZE, Array, lock, UsedTime, INTER);*

*BubbleSort(Mass\_SIZE, Array, lock+1, INTER);*

*MergeSortHEAD(Mass\_SIZE, Array, lock+2, INTER);*

*}*

*void Visualization(int Mass\_SIZE, vector<vector<int>>& MASS, int lock, float UsedTime,int INTER)*

*{*

*if(lock == 0)*

*cout <<" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nПочатковий МАСИВ:\n "<<endl;*

*else if (lock ==1)*

*cout <<" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nНовий вигляд МАСИВу "<<BLUE<<" (З використанням методу Бульбашки)"<< RESET <<":\n "<<endl;*

*else*

*cout <<" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nНовий вигляд МАСИВу "<<BLUE<<" (З використанням методу сортування злиттям)"<< RESET <<":\n "<<endl;*

*for(int i=0;i<Mass\_SIZE;i++){*

*for(int j=0;j<Mass\_SIZE;j++){*

*if (i==j)*

*printf(PINK"%5.i" RESET, MASS[i][j]);*

*else*

*printf("%5.i", MASS[i][j]);*

*}*

*cout << endl;*

*}*

*if(UsedTime !=0 ){*

*if (lock == 1)*

*cout <<"\n Затрачений час на виконання сортування методом Бульбашки :"<< BLUE << UsedTime << RED << " ms" << RESET << "\nКількісит використаних ітерацій для виконання сортування Бульбашкою: "<< BLUE << INTER << RESET <<endl;*

*else if(lock == 2)*

*cout <<"\n Затрачений час на виконання сортування злиттям :"<< BLUE<< UsedTime << RED << " ms" << RESET << "\nКількісит використаних ітерацій для виконання сортування злиттям: "<< BLUE << INTER << RESET <<endl;*

*}*

*cout <<" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n "<<endl;*

*}*

*void BubbleSort(int Mass\_SIZE, vector<vector<int>>& Array, int lock, int INTER)*

*{*

*const auto TimeStart = std::chrono::high\_resolution\_clock::now();*

*INTER = 0;*

*for(size\_t ZONE=0;ZONE<Mass\_SIZE;ZONE++)*

*for (size\_t i = 0; i + 1 < ZONE; i++)*

*for (size\_t j = 0; j + 1 < ZONE - i; j++)*

*{*

*if (Array[ZONE][j + 1] < Array[ZONE][j])*

*{*

*swap(Array[ZONE][j], Array[ZONE][j + 1]);*

*INTER++;*

*}*

*INTER++;*

*}*

*const auto TimeEnd = std::chrono::high\_resolution\_clock::now();*

*Visualization(Mass\_SIZE ,Array ,lock ,chrono::duration<float>(TimeEnd - TimeStart).count(), INTER);*

*}*

*void MergeSortHEAD(int Mass\_SIZE, vector<vector<int>>& CopyForFast, int lock, int INTER)*

*{*

*INTER=0;*

*const auto TimeStart = std::chrono::high\_resolution\_clock::now();*

*for (int i=0;i<Mass\_SIZE;i++)*

*{*

*if (!CopyForFast[i].empty()) {*

*vector<int> temp(CopyForFast[i].size());*

*MergeSortFunction(CopyForFast[i], temp, 0, i - 1, INTER);*

*INTER= INTER+2;*

*}*

*INTER++;*

*}*

*const auto TimeEnd = std::chrono::high\_resolution\_clock::now();*

*Visualization(Mass\_SIZE ,CopyForFast ,lock ,chrono::duration<float>(TimeEnd - TimeStart).count(), INTER);*

*}*

*void MergeSortFunction(vector<int>& CopyForFast, vector<int>& temp, int left, int right,int INTER)*

*{*

*if (left < right) {*

*int medium = (left + right) / 2;*

*INTER++;*

*MergeSortFunction(CopyForFast, temp, left, medium, INTER);*

*MergeSortFunction(CopyForFast, temp, medium + 1, right, INTER);*

*int k = left;*

*for (int i = left, j = medium + 1; i <= medium || j <= right; ) {*

*if (j > right || (i <= medium && CopyForFast[i] < CopyForFast[j])) {*

*temp[k] = CopyForFast[i];*

*++i;*

*} else {*

*temp[k] = CopyForFast[j];*

*++j;*

*INTER = INTER+2;*

*}*

*++k;*

*INTER = INTER+2;*

*}*

*for (int i = left; i <= right; ++i) {*

*CopyForFast[i] = temp[i];*

*INTER++;*

*}*

*INTER++;*

*}*

*}*