МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА КОМП’ЮТЕРНИХ ТЕХНОЛОГІЙ

**ЗВІТ**

лабораторної роботи №2

з дисципліни «Дискретна математика»

на тему «Прості методи сортування»

**ВИКОНАВ:**

Студент 1-го курсу

Групи ФЕП-14

Іванський Остап Юрійович

**ПЕРЕВІРИВ:**

Асистент кафедри РКТ

Баран Микола Орестович

Львів 2024

**МЕТА:** Вивчити основні методи сортування.

**ТЕОРИТИЧНІ ВІДОМОСТІ:**

Сортування (упорядкування) є однією з найбільш поширених операцій обробки даних. Розрізняють методи внутрішнього та зовнішнього сортування. Методи внутрішнього сортування застосовують до невеликих об’ємів даних і вони не вимагають додаткової пам’яті. Методи зовнішнього сортування застосовують до великих масивів даних, що зберігаються на зовнішніх носіях.

Серед методів внутрішнього сортування розрізняють прості та удосконалені методи.

Найбільш відомими простими методами сортування є:

* сортування обміном (бульбашкове сортування);
* сортування вставкою;
* сортування вибором.

Серед удосконалених методів найчастіше вживаються [3, 4]:

* швидке сортування (метод Хоара);
* сортування злиттям;
* сортування включенням зі спадним приростом (метод Шелла);
* сортування за допомогою дерева (пірамідальне сортування).

Зауваження. Надалі, розглядаючи різні методи сортування, не втрачаючи загальності, будемо розглядати сортування *за* *зростанням*. При сортуванні *за* *спаданням* алгоритми є аналогічними з точністю до знаку порівняння елементів.

Зауваження. Нижченаведений опис алгоритмів буде здійснюватись на прикладі сортування *одномірного* *масиву* (вектора), кожен елемент якого характеризується *індексом* (порядковим номером) та *ключем* (даними певного типу для яких можна застосувати операції порівняння). Звичайно, ці ж алгоритми сортування можна застосовувати і для інших структур даних, зокрема, *лінійних* *зв’язаних* *списків* (див. лабораторну роботу № 4).

**Сортування** **обміном** **(бульбашкове** **сортування)**

Бульбашкове сортування (англ. *bubble* *sort*) полягає у порівнянні двох сусідніх елементів і їх подальшій перестановці місцями, якщо їхній порядок не є вірний. Застосування цієї процедури до всіх пар елементів масиву, від останньої пари до першої, дозволяє виявити елемент з найменшим ключем і перемістити його в першу позицію. Цей процес нагадує спливання бульбашки (звідки і походить назва методу). За кожну ітерацію черговий елемент з найменшим ключем потрапляє на потрібну позицію у лівій частині масиву, збільшуючи таким чином розмір відсортованої ділянки. З огляду на це, при кожній новій ітерації не потрібно розглядати вже відсортовані елементи у лівій частині масиву, зменшуючи розмір правої частини на один елемент.

У вдосконаленому алгоритмі вводиться індикатор (прапорець) який відслідковує факт перестановки елементів. Перед кожною ітерацією цей індикатор встановлюється у стан false, а якщо перестановка відбулася – у стан true. Таким чином, якщо після завершення ітерації прапорець залишився у стані false, то всі пари елементів розміщенні у вірному порядку і сортування можна припинити. У випадку частково впорядкованих масивів такий підхід дозволяє скоротити кількість операцій.

Зауваження. У сортуванні обміном (а також і в багатьох інших алгоритмах) використовується операція перестановки місцями двох елементів. Для цієї мети служить стандартна (std::swap(…)) чи користувацька функція swap(a,b) яка реалізується наступним кодом:

temp = a;

a = b;

b = temp;

Параметрами цієї функції повинні служити вказівники чи посилання на елементи, які потрібно переставити [5, 6]

*Алгоритм* *сортування* *обміном* *(функція* SortBubble(…)*).*

1. Встановити лічильник ітерацій рівний одиниці. Поки лічильник не досяг значення індексу останнього елементу масиву (цикл за зростанням індексу):
   1. Для елементів масиву: від останнього елементу до елементу з індексом, рівному значенню лічильника ітерацій (цикл за спаданням індексу):
      1. Якщо поточний елемент є менший за наступний, то поміняти ці елементи місцями.

**Сортування** **вставкою**

В сортуванні методом вставки (англ. *insertion* *sort*) масив даних розділяється на дві частини: ліву – відсортовану і праву – невідсортовану. Перший елемент невідсортованої частини вставляється у відсортовану частину таким чином, щоб вона залишалася відсортованою. У результаті відсортована частина збільшується на один елемент, а невідсортована – зменшується. Таким чином, на кожному кроці потрібно знайти позицію у відсортованій частині для вставки нового елемента та здійснити саму вставку. Для забезпечення місця під вставку, елементи відсортованого підмасиву, ключі яких більші за ключ елемента, що вставляється, повинні бути зсунуті на одну позицію вправо. Оскільки такий зсув зітре перший елемент невідсортованої частини (власне той, що вставляєтся), перед вставкою його потрібно запам’ятати в допоміжній змінній.

На першій ітерації відсортована частина містить тільки перший елемент масиву, решту елементів масиву відносимо до невідсортованої частини.

Зауваження. Існує інша реалізація алгоритму, коли поточний елемент з невідсортованої частини переставляється місцями з попереднім елементом до тих пір, поки ключ цього елементу є більшим за його ключ. Така версія не вимагає введення допоміжної змінної, але потребує більшу кількість операцій (для здійснення перестановок).

*Алгоритм* *сортування* *вставкою* *(функція* SortInsertion(…)*).*

1. Встановити лічильник ітерацій рівний одиниці. Поки лічильник не досяг значення індексу останнього елементу масиву (цикл за зростанням індексу):
   1. Занести елемент, що вставляється (перший в невідсортованій ділянці – його індекс рівний лічильнику ітерацій) в допоміжну змінну.
   2. Для елементів масиву: від останнього елементу відсортованої частини допоки не знайдено місце для вставки, або не досягнуто початку масиву (цикл за спаданням індексу):
      1. Якщо ключ поточного елементу є більший за ключ елементу, що вставляється, то зсунути поточний елемент на одну позицію вправо (пошук місця для вставки).
   3. Вставити на місце останнього поточного елементу значення елементу, що вставляється з додаткової змінної.

**Сортування** **вибором**

При сортуванні вибором (англ. *selection* *sort*), як і в попередньому випадку, масив даних складається з відсортованої і невідсортованої частини. На кожному кроці у невідсортованій частині шукається елемент з мінімальним ключем і вставляється на її початок. Таким чином відсортована частина збільшується на один елемент, а невідсортована – зменшується. Процедура вставки здійснюється, як правило, шляхом обміну елементу з мінімальним ключем з першим елементом у невідсортованій частині.

На першій ітерації невідсортована частина складає весь масив.

Зауваження. Пошук мінімального (максимального) елементу у послідовності (масиві) є самостійною задачею і може бути реалізований у вигляді окремої функції(напр. MinItem(…) або MaxItem(…)). Ця функція повинна повертати індекс елемента з мінімальним (максимальним) ключем та містити як параметри сам масив, у якому здійснюється пошук, та індекси елементів, які визначають ділянку пошуку (оскілки пошук відбувається не в усьому масиві, то потрібно задати перший та останній елементи підмасиву пошуку).

*Алгоритм* *пошуку* *мінімального* *елементу* *в* *масиві* *(функція*

MinItem(…)*).*

1. Присвоїти змінній min значення індексу першого елементу підмасиву пошуку. Для елементів масиву: від другого до останнього елементів підмасиву (цикл за зростанням індексу):
   1. Якщо значення ключа поточного елементу є меншим за значення ключа елементу з індексом min, то присвоїти змінній min значення індексу поточного елементу.
2. Повернути значення min як результат функції.

Зауваження. Алгоритм пошуку максимального елементу в масиві є аналогічним попередньому.

*Алгоритм* *сортування* *вибором* *(функція* SortSelection(…)*).*

1. Для елементів масиву: від першого до останнього елементів (цикл за зростанням індексу):
   1. Знайти елемент з мінімальним ключем (функція MinItem(…)) серед елементів від поточного до останнього.
   2. Переставити місцями знайдений елемент з мінімальним ключем з поточним.

Обчислювальна складність простих алгоритмів сортування становить

*O*(*n*2 ) , де *n* – кількість елементів масиву, що обумовлено наявністю двох вкладених циклів. Окрім того, у зв’язку з великою кількістю перестановок, метод обміну вважається найгіршим і на практиці не використовується.

Серед вищеописаних алгоритмів найкращим вважається метод вставки,

однак для великих наборів даних доцільніше використовувати удосконалені методи [3].

**ХІД РОБОТИ**

1. Створюємо нову бібліотеку Sort (файли Sort.h, Sort.cpp).

2. У файлі Sort.h за допомогою команди typedef зв’язуємо тип ключа елементу масиву datatype з типом даних.

3. У бібліотеці Sort, згідно описаних вище алгоритмів, реалізувуємо функції сортування методом обміну (бульбашки), вставки та вибору (функції SortBubble(…), SortInsertion(…), SortSelection(…)).

4. В цій же бібліотеці запрограмовуємо допоміжні функції для перестановки елементів масиву (swap(…)), відображення вмісту масиву (show(…)) та пошуку елементу з мінімальним та максимальним ключем у частині масиву (MinItem(…), MaxItem(…)).

5. Підключаємо бібліотеку Sort. Реалізувуємо меню для вибору методу сортування масиву з даними.

6. Модифікуємо алгоритми сортування таким чином, щоб вони дозволяли відсортовувати тільки задану частину масиву.

**Sort.h**

#ifndef SORT\_H  
#define SORT\_H  
typedef int datatype;  
  
void swap(datatype& a, datatype& b);  
void show(const datatype arr[], int size);  
int MinItem(const datatype arr[], int size, int start = 0, int end = -1);  
int MaxItem(const datatype arr[], int size, int start = 0, int end = -1);  
void SortBubble(datatype arr[], int size,int reverse = 0, int start = 0, int end = -1);  
void SortInsertion(datatype arr[], int size,int reverse = 0, int start = 0, int end = -1);  
void SortSelection(datatype arr[], int size,int reverse = 0, int start = 0, int end = -1);  
  
#endif *//SORT\_H*

**Sort.cpp**

#include "Sort.h"  
#include <iostream>  
#include <chrono>  
  
void swap(datatype& a, datatype& b) {  
 datatype temp = a;  
 a = b;  
 b = temp;  
}  
  
void show(const datatype arr[], int size) {  
 for (int i = 0; i < size; i++) {  
 std::cout << arr[i] << " ";  
 }  
 std::cout << std::endl;  
}  
  
int MinItem(const datatype arr[], int size, int start, int end) {  
 if (end == -1) end = size - 1;  
 int minIndex = start;  
 for (int i = start + 1; i <= end; i++) {  
 if (arr[i] < arr[minIndex]) {  
 minIndex = i;  
 }  
 }  
 return minIndex;  
}  
  
int MaxItem(const datatype arr[], int size, int start, int end) {  
 if (end == -1) end = size - 1;  
 int maxIndex = start;  
 for (int i = start + 1; i <= end; i++) {  
 if (arr[i] > arr[maxIndex]) {  
 maxIndex = i;  
 }  
 }  
 return maxIndex;  
}  
  
void SortBubble(datatype arr[], int size, int reverse, int start, int end) {  
 auto startTime = std::chrono::high\_resolution\_clock::*now*();  
 int counter = 0;  
 if (end == -1) end = size - 1;  
 for (int i = end; i > start; i--) {  
 for (int j = start; j < i; j++) {  
 counter++;  
 if (reverse==0) {  
 if (arr[j] > arr[j + 1]) {  
 swap(arr[j], arr[j + 1]);  
 }  
 }  
 else if (reverse==1) {  
 if (arr[j] < arr[j + 1]) {  
 swap(arr[j], arr[j + 1]);  
 }  
 }  
 }  
 }  
 auto endTime = std::chrono::high\_resolution\_clock::*now*();  
 auto duration = std::chrono::duration\_cast<std::chrono::nanoseconds>(endTime - startTime);  
 std::cout << "Time: " << duration.count() << " ns" << std::endl;  
 std::cout << "Iterations: "<<counter << std::endl;  
}  
  
void SortInsertion(datatype arr[], int size,int reverse, int start, int end) {  
 auto startTime = std::chrono::high\_resolution\_clock::*now*();  
 int counter = 0;  
 if (end == -1) end = size - 1;  
 for (int i = start + 1; i <= end; i++) {  
 datatype key = arr[i];  
 int j = i - 1;  
 if (reverse==0) {  
 while (j >= start && arr[j] > key) {  
 counter++;  
 arr[j + 1] = arr[j];  
 j--;  
 }  
 }  
 else if (reverse==1) {  
 while (j >= start && arr[j] < key) {  
 counter++;  
 arr[j + 1] = arr[j];  
 j--;  
 }  
 }  
 arr[j + 1] = key;  
 }  
 auto endTime = std::chrono::high\_resolution\_clock::*now*();  
 auto duration = std::chrono::duration\_cast<std::chrono::nanoseconds>(endTime - startTime);  
 std::cout << "Time: " << duration.count() << " ns" << std::endl;  
 std::cout << "Iterations: "<<counter << std::endl;  
}  
  
void SortSelection(datatype arr[], int size,int reverse, int start, int end) {  
 auto startTime = std::chrono::high\_resolution\_clock::*now*();  
 int counter = 0;  
 if (end == -1) end = size - 1;  
 if (reverse==0) {  
 for (int i = start; i < end; i++) {  
 counter++;  
 int minIndex = MinItem(arr, size, i, end);  
 swap(arr[i], arr[minIndex]);  
 }  
 }else if (reverse==1) {  
 for (int i = start; i < end; i++) {  
 counter++;  
 int maxIndex = MaxItem(arr, size, i, end);  
 swap(arr[i], arr[maxIndex]);  
 }  
 }  
 auto endTime = std::chrono::high\_resolution\_clock::*now*();  
 auto duration = std::chrono::duration\_cast<std::chrono::nanoseconds>(endTime - startTime);  
 std::cout << "Time: " << duration.count() << " ns" << std::endl;  
 std::cout << "Iterations: "<<counter << std::endl;  
}

**Menu.h**

#ifndef MENU\_H  
#define MENU\_H  
#include "Sort.h"  
typedef int datatype;  
  
void MainMenu(datatype arr[],int size);  
  
#endif *//MENU\_H*

**Menu.cpp**

void ThirdSubMenu(datatype arr[],int size,int mainChoice,int secondChoice) {  
 int choice,start,end;  
 std::cout << "1. Full sort\n";  
 std::cout << "2. Partial sort\n";  
 std::cin >> choice;  
 if(choice == 1) {  
 switch (mainChoice) {  
 case 1:  
 SortBubble(arr,size,secondChoice);  
 break;  
 case 2:  
 SortInsertion(arr,size,secondChoice);  
 break;  
 case 3:  
 SortSelection(arr,size,secondChoice);  
 break;  
 }  
 }else if(choice == 2) {  
 std::cout << "Arr: \n";  
 show(arr,size);  
 std::cout << "Size: \n"<<size<<"\n";  
 std::cout << "Enter start and end index\n";  
 std::cin >> start>>end;  
 switch (mainChoice) {  
 case 1:  
 SortBubble(arr,size,secondChoice,start,end);  
 break;  
 case 2:  
 SortInsertion(arr,size,secondChoice,start,end);  
 break;  
 case 3:  
 SortSelection(arr,size,secondChoice,start,end);  
 break;  
 }  
 }  
}  
  
  
void SecondMenu(datatype arr[],int size,int mainChoice) {  
 int secondChoice;  
 std::cout << "1. Straight sort\n";  
 std::cout << "2. Reverse sort\n";  
 std::cin >> secondChoice;  
 ThirdSubMenu(arr,size,mainChoice,secondChoice-1);  
}  
  
void MainMenu(datatype arr[],int size) {  
  
 int mainChoice;  
  
 while (true) {  
 std::cout << "Chose a sort method:\n";  
 std::cout << "1. Bubble sort\n";  
 std::cout << "2. Insertion sort\n";  
 std::cout << "3. Selection sort\n";  
 std::cout << "4. Exit\n";  
 std::cin >> mainChoice;  
 if (mainChoice == 4) {  
 break;  
 }  
 SecondMenu(arr,size,mainChoice);  
  
 std::cout << "Array has been sorted: ";  
 show(arr, size);  
 std::cout << "MinItem - "<<MinItem(arr,size)<<"\n";  
 std::cout << "MaxItem - "<<MaxItem(arr,size)<<"\n";  
  
 }  
};

**Main.cpp**

int main() {  
 datatype arr[] = {5, 3, 8, 1,-10,1,9, 2, 7};  
 int size = sizeof(arr) / sizeof(arr[0]);  
 MainMenu(arr,size);  
   
 return 0;  
}

**ВИСНОВОК**

Під час виконання лабораторної роботи вивчив основи методи сортування такі як сортування бульбашкою, вставкою та вибором. Реалізували меню вибору сортування, також можливість сортування у напрямку спадання та сортування частини масиву.