МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра інформаційних систем та технологій

Звіт

з комп'ютерного практикуму 6(9-10)

«Алгоритми сортування. Методи сортування малих обсягів даних»

з дисципліни

«Теорія алгоритмів»

Бригада – 10

Варіант № 1

Перевірила:
ст. вик. Солдатова М.О.

Виконали:
Бойко Катерина,
Гоголь Софія,
Павлова Софія,
Хіврич Володимир

Київ 2022

Комп'ютерний практикум 9-10

Тема: Алгоритми сортування. Методи сортування малих обсягів даних.

Мета роботи: Дослідження та порівняння алгоритмів сортування.

Завдання

Постановка задачі:

Варіант 1:

Жадібний леприкон зібрав всі можливі скарби в своїх печерах та вирішив їх розкласти по скриня в тій же кількості, що і знаходив.. Майстер по виготовленню скринь робить їх поступово від найменшої до найбільшої, тому вимагає надати йому відповідний перелік розмірів скарбів. Леприкон дав йому відповідний перелік. Але поки майстер розмірковував, гном пішов до сусіда та взяв частину його скарбів в мішечках та додав до своїх і знову пішов до майстра. Сформував новий список скарбів. Майстер прийняв замовлення. Але сусідам сподобалась така ідея і вони принесли свої скарби в мішечках леприкону, щоб розкласти по скринях. Він побачив, що скарбів в мішечках часто повторювалася. Але знову пішов до майстра з остаточним списком. Допоможіть йому дати майстру інформацію для кожного з трьох випадків.

Мета задачі – надати майстру відсортований список скарбів.

Використані структури даних: масиви, покажчики, цілочисельні типи.

Вибір алгоритму

Сортування підрахунком

Сортування підрахунком (Counting sort) — алгоритм впорядкування, що застосовується при малій кількості різних елементів (ключів) у масиві даних.

Час його роботи лінійно залежить як від загальної кількості елементів у масиві, так і від кількості різних елементів. Ідея алгоритму полягає у тому, щоб підрахувати скільки разів кожен елемент зустрічається у 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. Сортування підрахунком є стабільним методом.

Принцип роботи

Підраховуємо скільки разів в масиві зустрічається кожне значення, і заповнюємо масив підрахованими елементами у відповідних кількостях. Лічильники для всього діапазону чисел створюються заздалегідь (спочатку дорівнюють нулю).

Складність алгоритму сортування підрахунком за часом

Найгірший випадок – **O (n + k)**

Середній випадок – **O (n + k)**

Кращий випадок – **O (n)**

де n — кількість елементів у вхідному масиві, а k — діапазон вводу

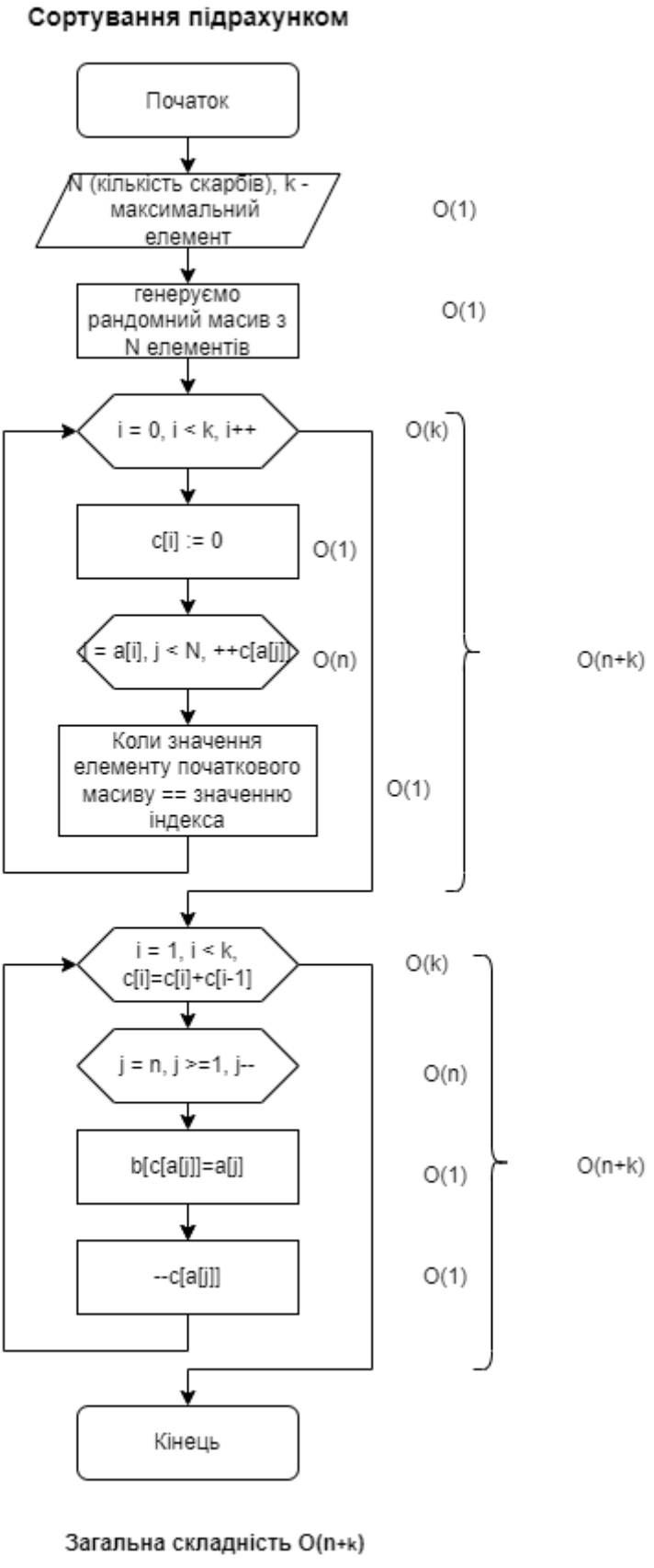


Рис. 1. Блок-схема алгоритму сортування підрахунком

Сортування Шелла

Сортування Шелла (Shell sort) — це алгоритм сортування, що є узагальненням сортування включенням.

Алгоритм базується на двох тезах:

- Сортування включенням ефективне для майже впорядкованих масивів.
- Сортування включенням неефективне, тому що переміщує елемент тільки на одну позицію за раз.

Тому сортування Шелла виконує декілька впорядкувань включенням, кожен раз порівнюючи і переставляючи елементи, що розташовані на різній відстані один від одного. Сортування Шелла не є стабільним методом.

Принцип роботи

1. На початку обираються m-елементів: d1, d2, ... dm причому d1 > d2 > ··> dm = 1, .

2. Потім виконується m впорядкувань методом включення, спочатку для елементів, що стоять через d1, потім для елементів через d2 і т. д. до dm =1

3. Ефективність досягається тим, що кожне наступне впорядкування вимагає меншої кількості перестановок, оскільки деякі елементи вже стали на свої місця.

4. Оскільки dm = 1, то на останньому кроці виконується звичайне впорядкування включенням всього масиву, а отже кінцевий масив буде впорядкованим.

Складність алгоритму сортування Шелла за часом

Найгірший та середній випадок – **O (n^2)**

Кращий випадок – **O (n)**

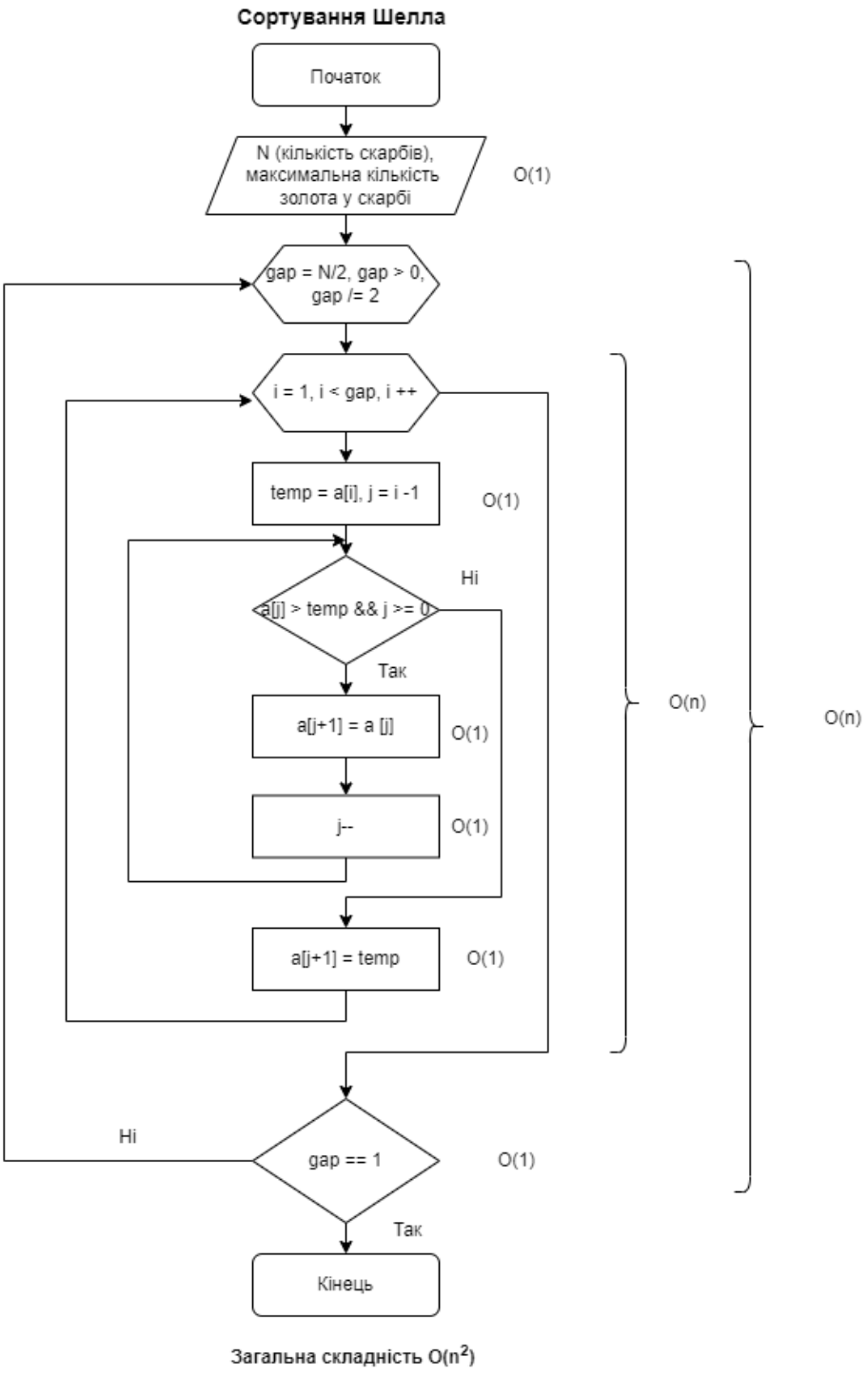


Рис. 2. Блок-схема алгоритму сортування методом Шелла

Швидке сортування

Швидке сортування (Quick Sort) — алгоритм сортування, який використовує дуже прості цикли й операції, він працює швидше за інші алгоритми, що мають таку ж асимптотичну оцінку складності. Наприклад, зазвичай більш ніж удвічі швидший у порівнянні з сортуванням злиттям. Швидке сортування не є стабільним.

Принцип роботи

Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини та кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин відбувається рекурсивно. Алгоритм швидкого сортування може бути реалізований як у масиві, так і у двозв'язному списку.

Алгоритм складається з трьох кроків:

1. Вибрати елемент з масиву. Має назву опорний.

2. Розбиття: перерозподіл елементів в масиві таким чином, що елементи менше опорного розміщаються перед ним, а більше або рівні після.

3. Рекурсивно застосувати перші два кроки до двох підмасивів зліва і праворуч від опорного елемента. Рекурсія не застосовується до масиву, в якому тільки один елемент або відсутні елементи.

Складність алгоритму швидкого сортування за часом:

Найгірший та середній випадок – **O (n^2)**

Кращий випадок – **O (n log(n))**

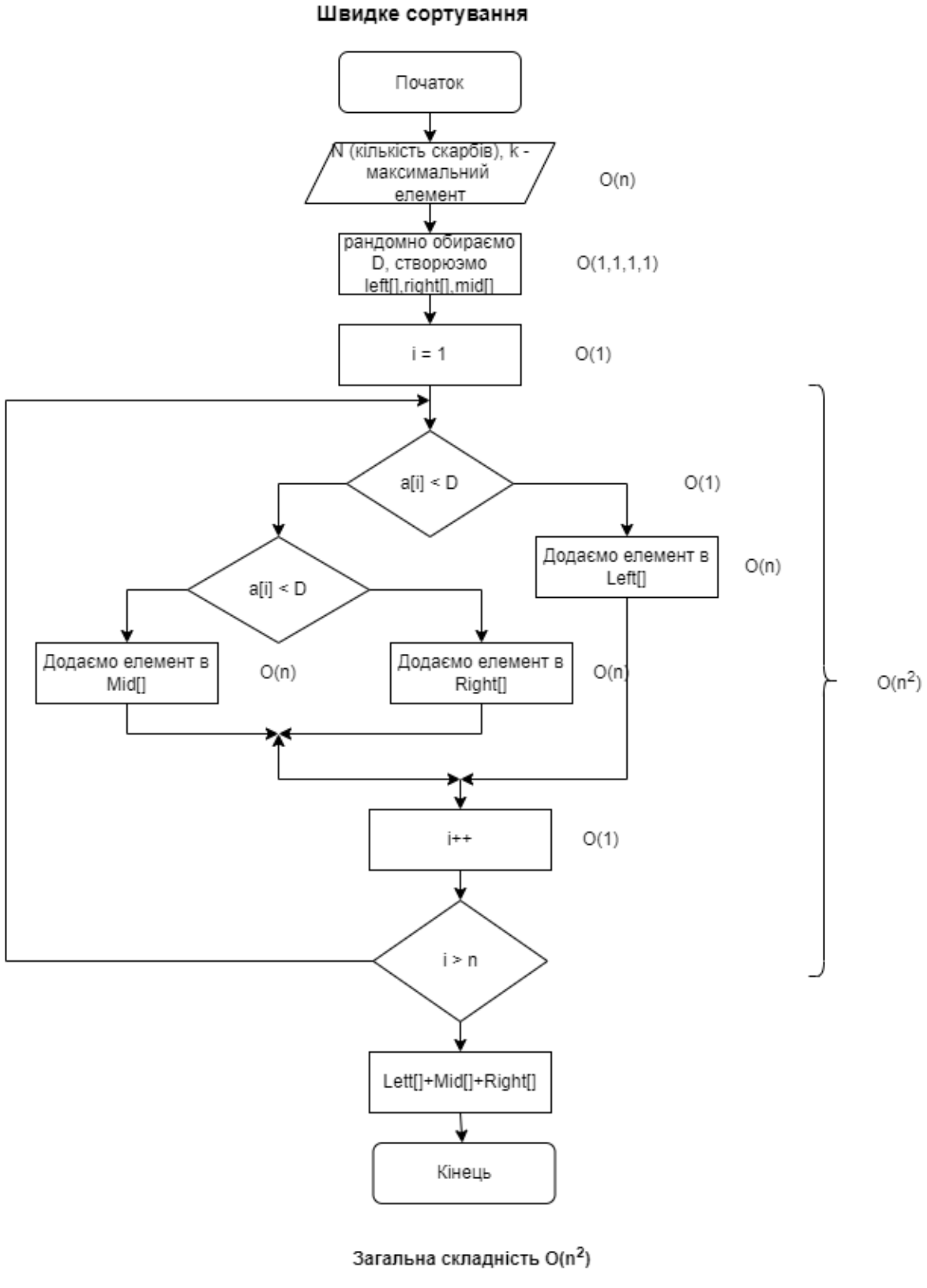


Рис. 3. Блок-схема алгоритму швидкого сортування

Вхідні дані

Назва змінної	Тип змінної	Значення змінної
length0	float	Кількість скарбів, що зібрав Леприкон
end0	float	Максимальна кількість золота
partlySorted0	string	Чи створити частково сортований масив
varik	int	Допомагає обрати тип сортування
addLength0	float	Кількість скарбів, що принесли друзі до Леприкона

Табл. 1 Таблиця вхідних даних

Модель:

Основні величини:

Назва змінної	Тип змінної	Значення змінної
length0	float	Кількість скарбів леприкона
end0	float	Максимальна кількість золота
addLength0	float	Кількість скарбів, що принесли друзі до Леприкона
length	int	Кількість скарбів Леприкона
arr	int*	масив, що буде сортуватися
end	int	Максимальна кількість золота
addLength	int	Кількість скарбів, що принесли друзі до Леприкона

Табл. 2. Таблиця основних величин

Допоміжні величини:

Назва змінної	Тип змінної	Значення змінної
array	int*	допоміжний масив, працює у тому випадку, коли друзі приносять скарби до Леприкона

Табл 3. Таблиця допоміжних величин

Лістинг програмної реалізації

```
#include <iostream>
#include <windows.h>
#include <vector>
#include <ctime>

using namespace std;

void shellSort(int arr[], int n)
{
   // Start with a big gap, then reduce the gap
   for (int gap = n / 2; gap > 0; gap /= 2)
   {
      // Do a gapped insertion sort for this gap size.
      // The first gap elements a[0..gap-1] are already in gapped order
      // keep adding one more element until the entire array is
      // gap sorted
      for (int i = gap; i < n; i += 1)
      {
         // add a[i] to the elements that have been gap sorted
         // save a[i] in temp and make a hole at position i
         int temp = arr[i];

         // shift earlier gap-sorted elements up until the correct
         // location for a[i] is found
         int j;
         for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
            arr[j] = arr[j - gap];

         //  put temp (the original a[i]) in its correct location
         arr[j] = temp;
      }
   }
}

void countSort(int arr[], int n)
{
   int max = arr[0];
   int min = arr[0];
   for (int i = 0; i < n; i++)
   {
      if (arr[i] > max) max = arr[i];
      if (arr[i] < min) min = arr[i];
   }
   int range = max - min + 1;
```

```
    vector<int> count(range), output(n);
    for (int i = 0; i < n; i++)
        count[arr[i] - min]++;

    for (int i = 1; i < count.size(); i++)
        count[i] += count[i - 1];

    for (int i = n - 1; i >= 0; i--)
    {
        output[count[arr[i] - min] - 1] = arr[i];
        count[arr[i] - min]--;
    }

    for (int i = 0; i < n; i++)
        arr[i] = output[i];
}

// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition(int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element and indicates the right position of pivot found so far

    for (int j = low; j <= high - 1; j++)
    {
        // If current element is smaller than the pivot
        if (arr[j] < pivot)
        {
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

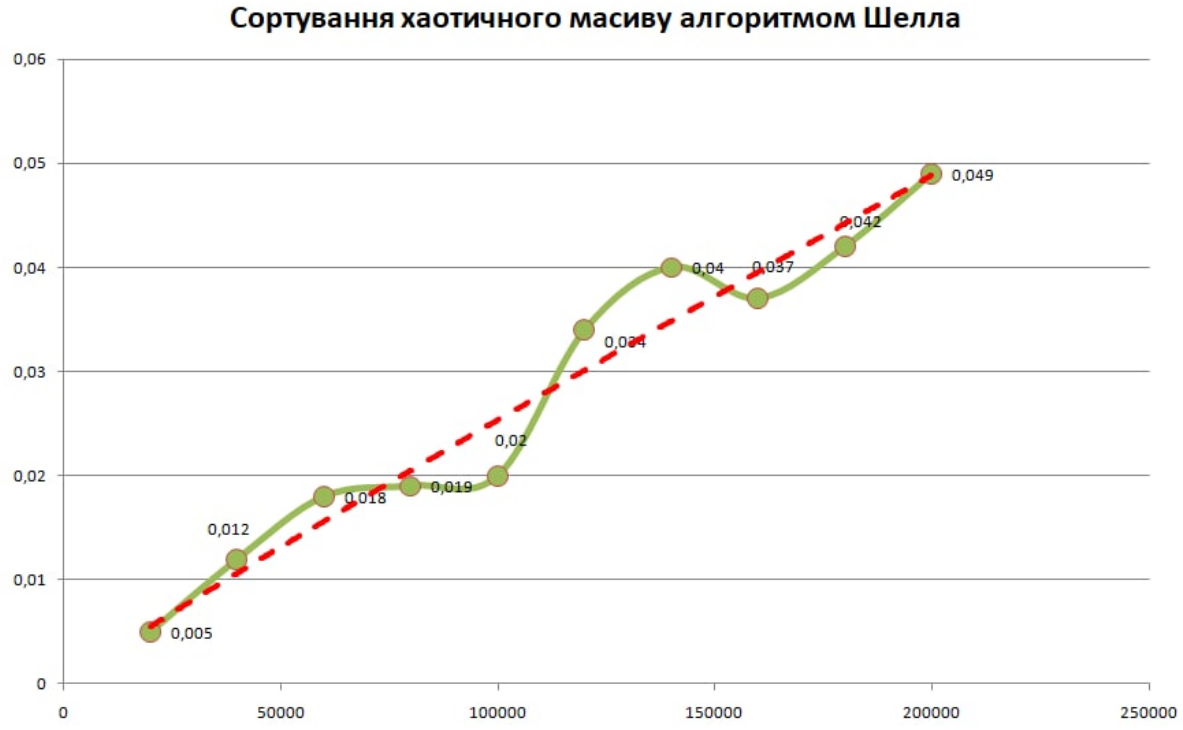
```
/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
   if (low < high)
   {
      /* pi is partitioning index, arr[p] is now
      at right place */
      int pi = partition(arr, low, high);

      // Separately sort elements before
      // partition and after partition
      quickSort(arr, low, pi - 1);
      quickSort(arr, pi + 1, high);
   }
}

void createArray(int arr[], int n, int end)
{
   for (int i = 0; i < n; i++)
      arr[i] = rand() % end;
}

void addElementsToArray(int arr[], int start, int n, int end)
{
   for (int i = start; i < start + n; i++)
      arr[i] = rand() % end;
}

void printArray(int arr[], int n)
{
   for (int i = 0; i < n; i++)
      cout << arr[i] << " ";
}

int ErorCheck(float a)
{
   if (a <= 0 || (a - int(a) != 0))
   {
      cout << "\n * -------- * sќћ»Ћ j * -------- * \n";
      cout << " ”пс! ”ведено не натуральне число! \n\n";
      return 0;
   }
}

int main()
```

```
{
   SetConsoleCP(1251);
   SetConsoleOutputCP(1251);

   cout << "\t * -------- *  омп'ютерний практикум 6 * -------- *\n";
   cout << "\t\t\t Sригада 10. ¬ар≥ант 1  \n";
   cout << "\t     –еал≥зац≥¤ простих алгоритм≥в сортуванн¤\n\n";

   float length0, end0, addLength0;
   int i, length, end, varik = 0, partlySorted, addLength;
   string partlySorted0;

   cout << "\nЋеприкон з≥брав N скарб≥в\nN = ";
   cin >> length0;
   ErorCheck(length0);
   length = int(length0);
   int* arr = new int[length];
   int* array = new int[length];

   cout << "\nћаксимальна к≥льк≥сть золота = ";
   cin >> end0;
   ErorCheck(end0);
   end = int(end0);
   createArray(arr, length, end);

   cout << "\n—творити частково сортований масив? (0/1): ";
   cin >> partlySorted0;
   if (partlySorted0 == "0" || partlySorted0 == "1") partlySorted = atoi(partlySorted0.c_str());
   else
   {
      cout << "\n    * -------- * sќЋ»Ћ j * -------- * \n";
      cout << " Ќеправильний вар≥ант виконанн¤ програми!\n";
   }

   cout << "\nсерел≥к скарб≥в леприкона у пор¤дку њх отриманн¤: \n";
   printArray(arr, length);
   cout << "\n";

   cout << "\n-----------Ћенюшка----------";
   cout << "\n0 - —ортуванн¤ Ўелла\n1 - —ортуванн¤ п≥драхунком\n2 - Ўвидке сортуванн¤";
   cout << "\n----------------------------";
   cout << "\nќбер≥ть алгоритм сортуванн¤ (0/1/2): ";
   cin >> varik;

   float start = clock();

   switch (varik)
   {
```

```
case 0:
   if (partlySorted == 0) cout << "\n—ќ–“”¬jЌЌя Ў≈ЋЋj:";
   shellSort(arr, length);
   break;
case 1:
   if (partlySorted == 0) cout << "\n—ќ–“”¬jЌЌя s≤ƒ–j’”Ќ ќЋ:";
   countSort(arr, length);
   break;
case 2:
   if (partlySorted == 0) cout << "\nЎ¬»ƒ ≈ —ќ–“”¬jЌЌя:";
   quickSort(arr, 0, length - 1);
   break;
default:
   cout << "\n    * -------- * sќЋ»Ћ j * -------- * \n";
   cout << " Ќеправильний вар≥ант виконанн¤ програми!\n";
   return 0;
}

if (partlySorted == 1)
{
   cout << "\nƒруз≥ до леприкона принесли ще N скарб≥в\nN = ";
   cin >> addLength0;

   ErorCheck(addLength0);
   addLength = int(addLength0);
   int* array = new int[length + addLength]();
   for (int i = 0; i < length; i++)
      array[i] = arr[i];

   addElementsToArray(array, length, addLength, end);
   cout << "\nЌовий перел≥к скарб≥в леприкона: \n";
   printArray(array, length + addLength);

   //float start = clock();

   if (varik == 0)
   {
      cout << "\n\n—ќ–“”¬jЌЌя Ў≈ЋЋj:";
      shellSort(array, length + addLength);
   }
   else if (varik == 1)
   {
      cout << "\n\n—ќ–“”¬jЌЌя s≤ƒ–j’”Ќ ќЋ:";
      countSort(array, length + addLength);
   }
   else
   {
      cout << "\n\nЎ¬»ƒ ≈ —ќ–“”¬jЌЌя:";
```

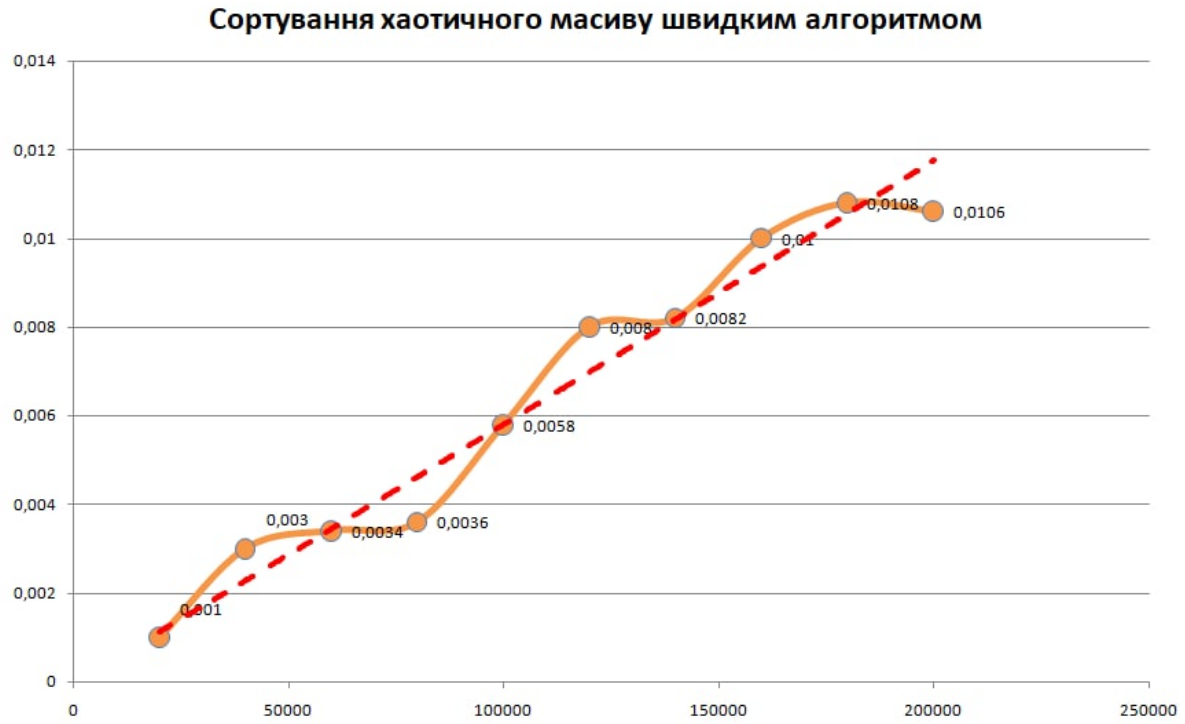
```
            quickSort(array, 0, length + addLength - 1);
        }

        //cout << (clock() - start) / CLOCKS_PER_SEC;

        cout << "\n¬≥дсортований список, що леприкон в≥днесе майстру: \n";
        printArray(array, length + addLength);
    }
    else
    {
        //cout << (clock() - start) / CLOCKS_PER_SEC;

        cout << "\n¬≥дсортований список, що леприкон в≥днесе майстру: \n";
        printArray(arr, length);
    }

    cout << "\n\n\n";
    return 0;
}
```

Перевірка обчислювальної складності програми:



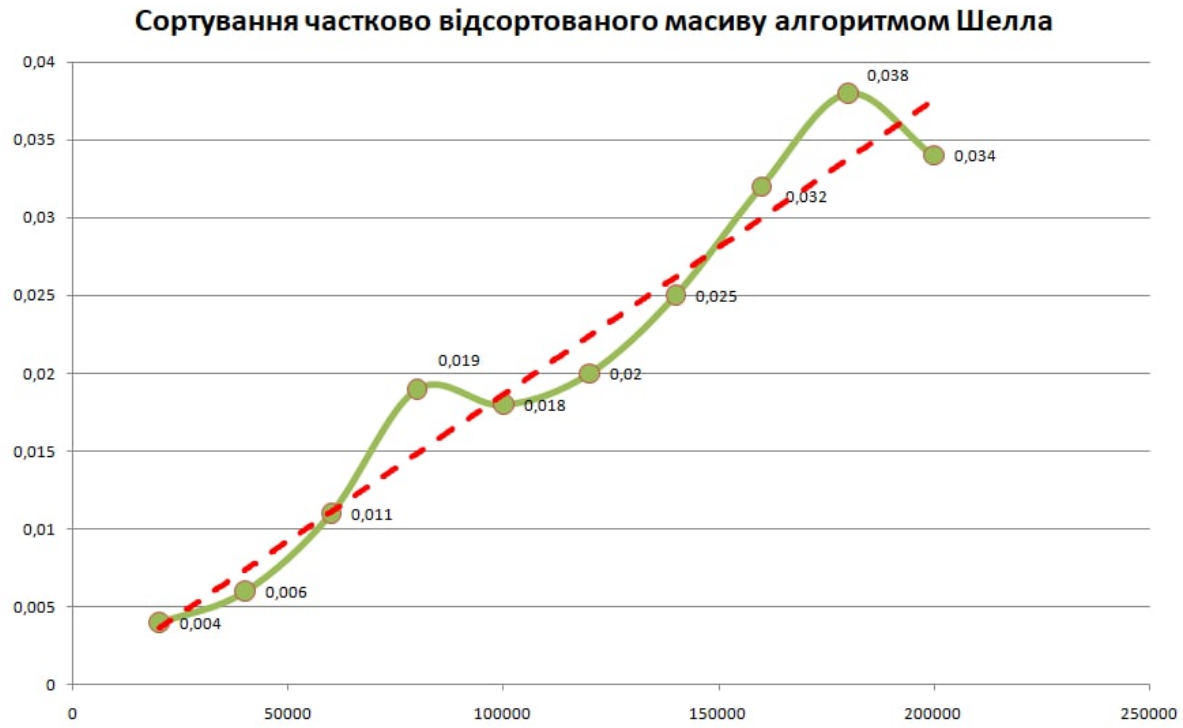
Діагр. 1. Сортування хаотичного масиву алгоритмом Шелла

Діагр. 2. Сортування хаотичного масиву алгоритмом підрахунку

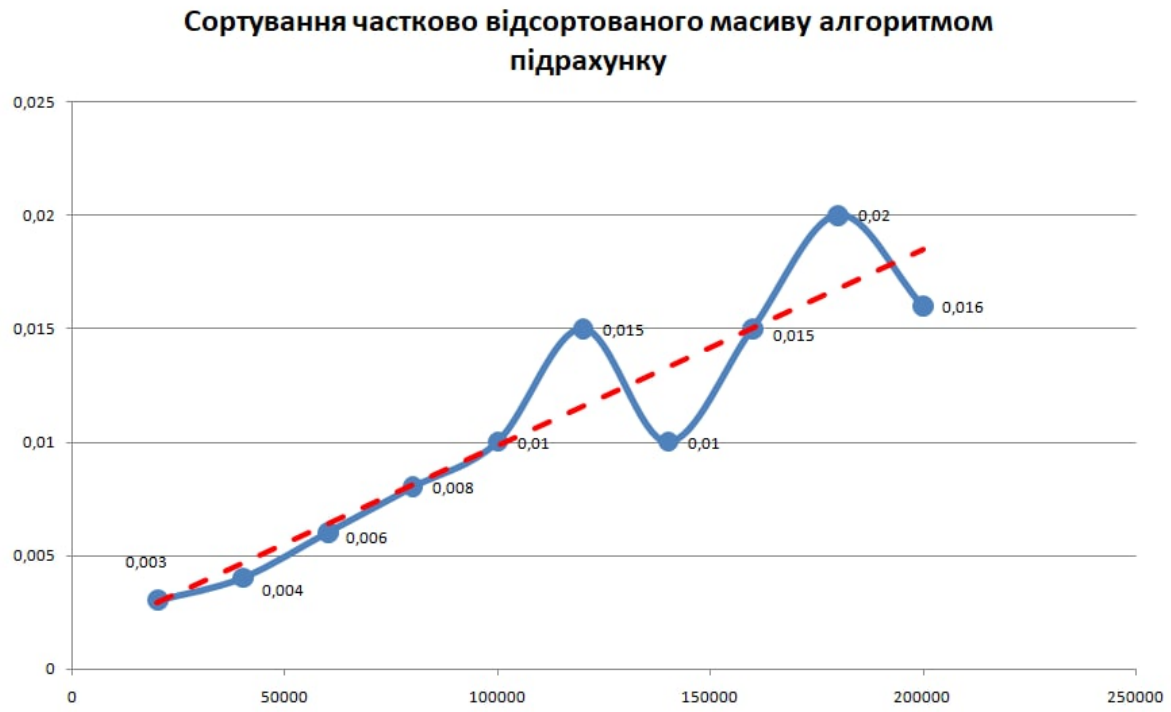


Діагр. 3. Сортування хаотичного масиву алгоритмом швидкого сортування

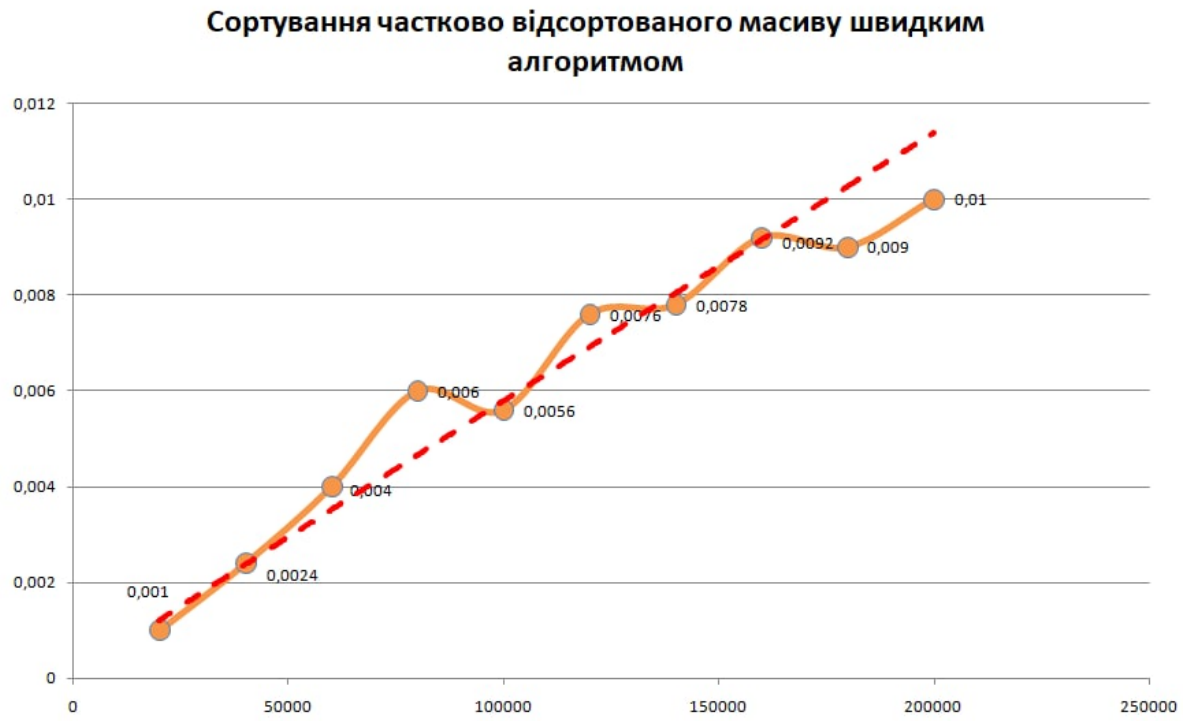
Діагр. 4. Порівняльна діаграма сортування хаотичного масиву різними алгоритмами



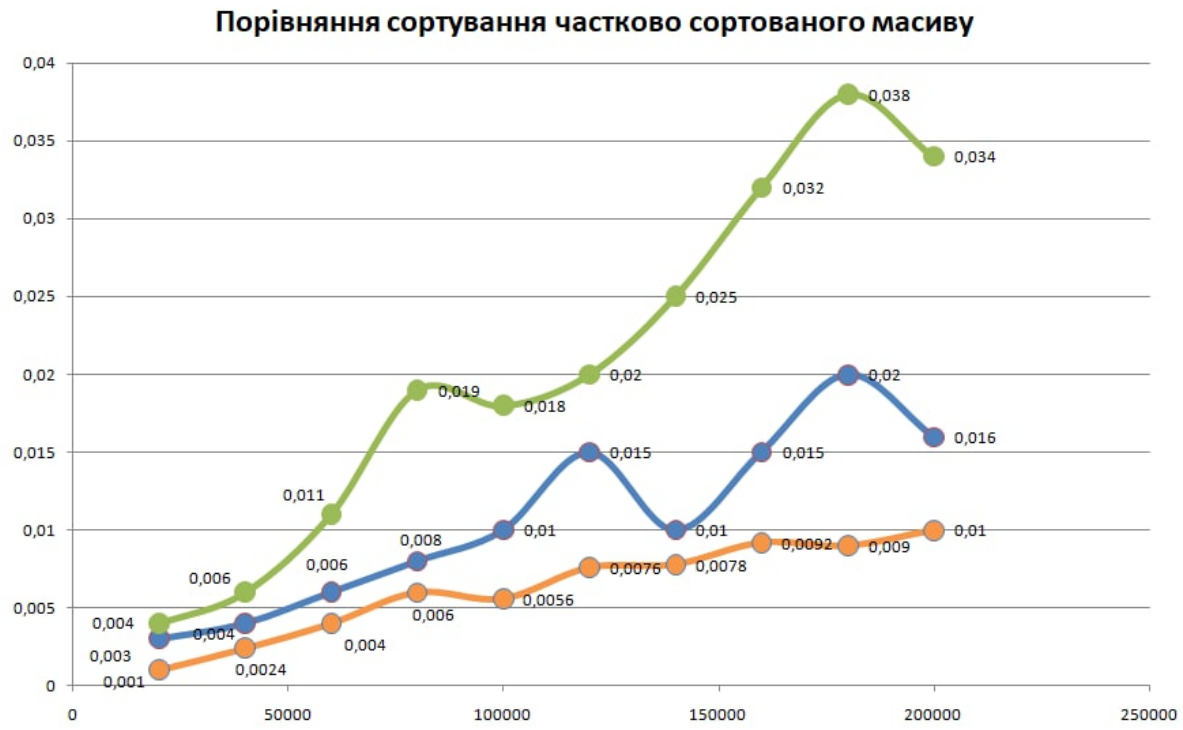
Діагр. 5. Сортування частково відсортованого масиву алгоритмом Шелла



Діагр. 6. Сортування частково відсортованого масиву алгоритмом підрахунку



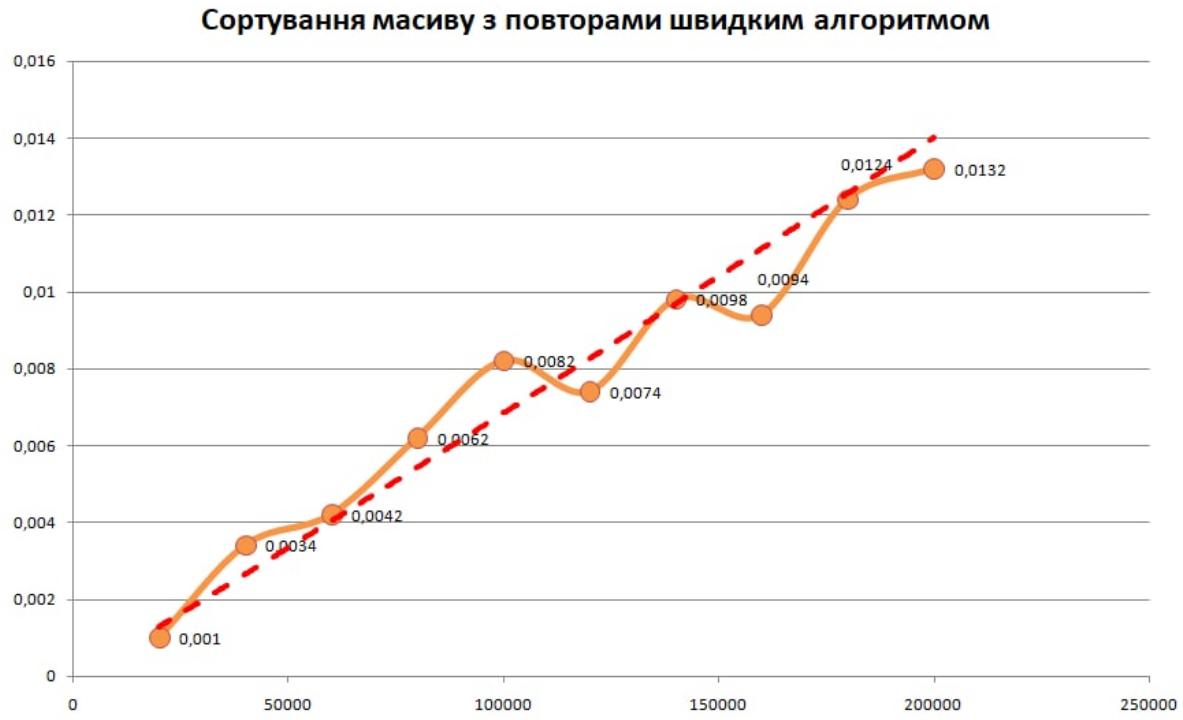
Діагр. 7. Сортування частково відсортованого масиву швидким алгоритмом



Діагр. 8. Порівняльна діаграма сортування частково сортованого масиву різними алгоритмами

Діагр. 9. Сортування масиву з повторами алгоритмом Шелла

Діагр. 10. Сортування масиву з повторами алгоритмом підрахунку



Діагр. 11. Сортування масиву з повторами швидким алгоритмом

Діагр. 12. Порівняльна діаграма сортування масиву з повторами різними алгоритмами

Скріншоти роботи програми

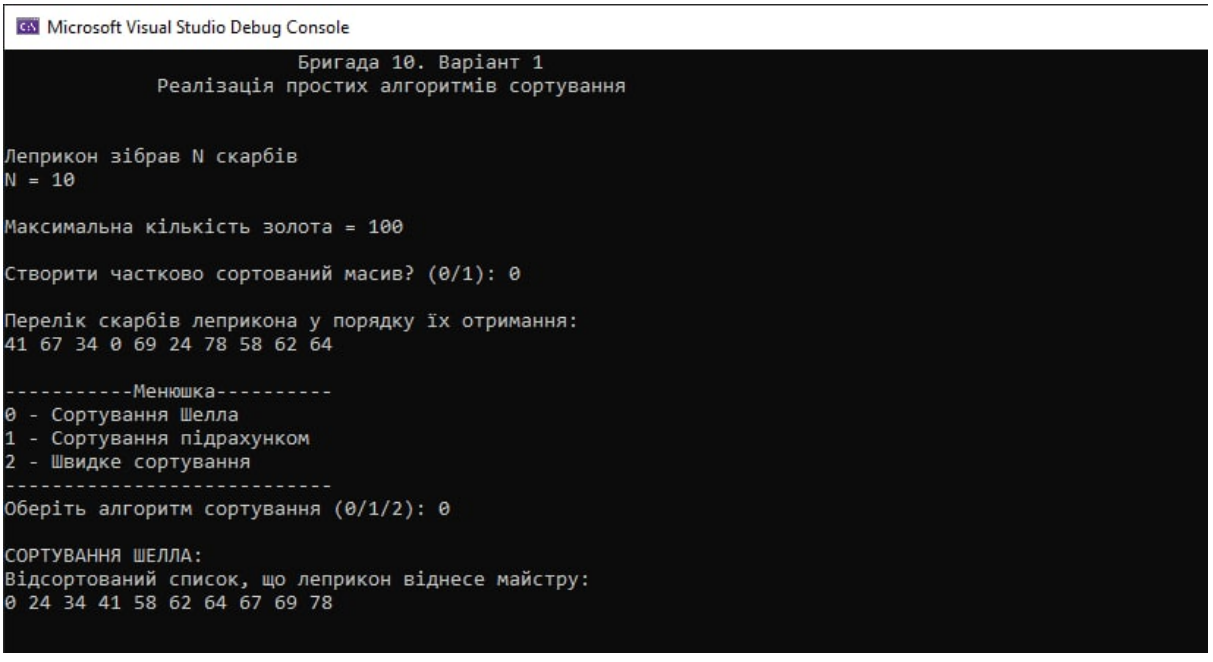


Рис. 4. Сортування Шелла з невідсортованим масивом

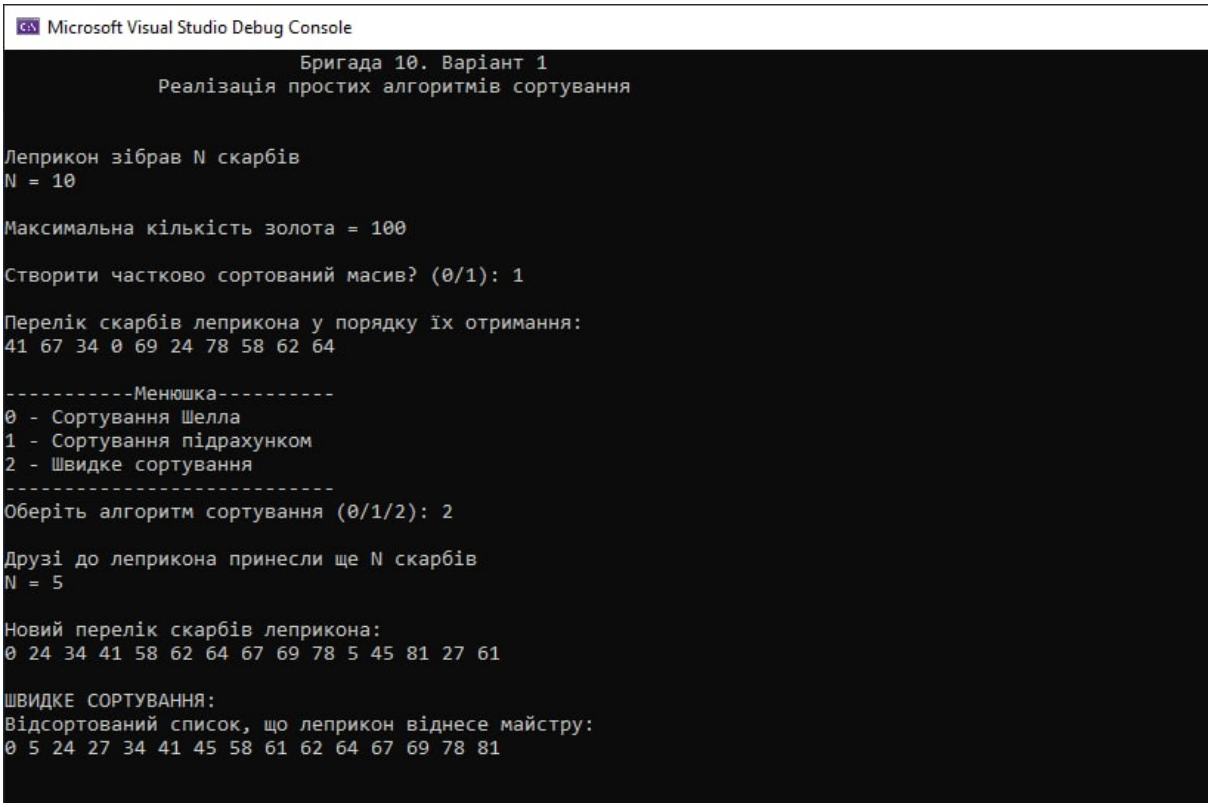


Рис. 5. Швидке сортування з частково відсортованим масивом

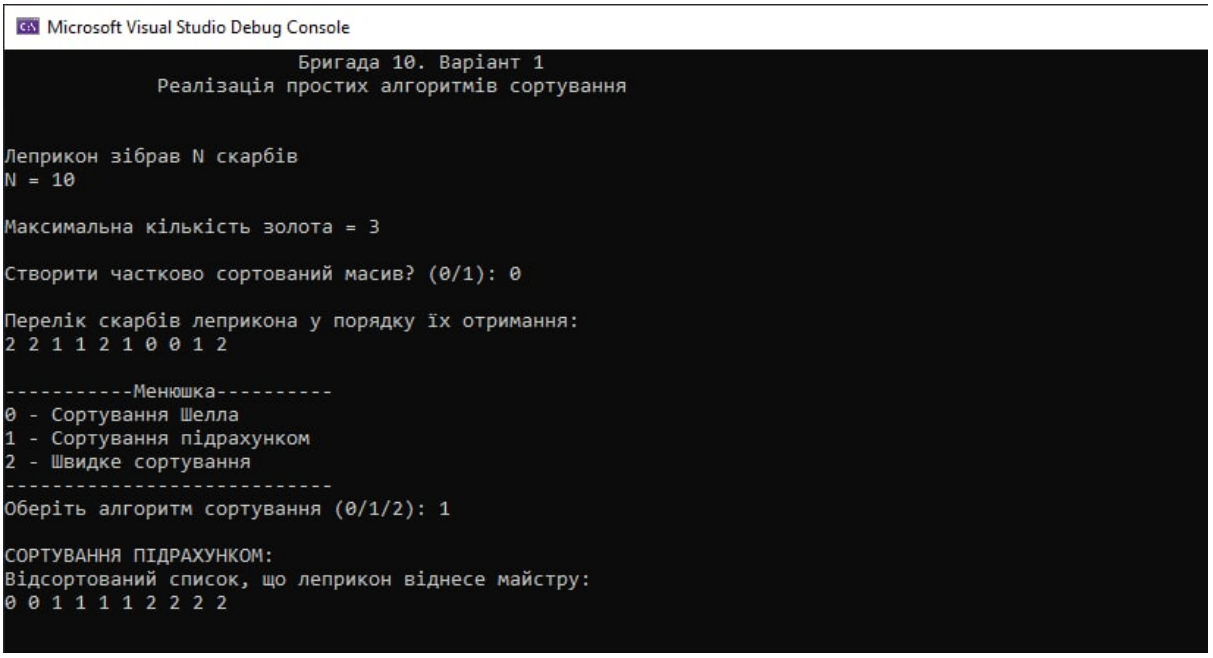


Рис. 6. Сортування підрахунком масиву з повтореннями

Висновок щодо доцільності використання алгоритму

Зазначимо, що для кожної з окремих варіантів постановки задачі та вхідних даних, за допомогою відомих теоретичних відомостей та аналізу складності, було обрано, на нашу думку найвлучніші методи для тестування.
Так, за нашими припущеннями: алгоритм підрахунку буде найбільш ефективний на типу даних “Масив з повтореннями”, тим часом алгоритм Шелла — на решті.
У реальності маємо таку картину: серед обраних нами алгоритмів, найефективнішим виявився алгоритм сортування підрахунком у трьох випадках вхідних даних.
Проаналізувавши отриманні результати ми знайшли пояснення цієї розбіжності. Через те, що було обрано для реалізації класичний варіант алгоритму Шелла, його складність склала O(n^2), через що, прогноз щодо ефективності цього алгоритму не справдився.

Перевірка правильності програми:

Вхідні дані	Результат	Призначення тесту
length0, end0, addLength0		
-1	Відомий	Перевірка, завершення некоректні вхідні дані (додавання повідомлень про помилки, завершення роботи програми)
2	Відомий	Дані коректні, програма продовжує виконувати свою роботу
b	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки, завершення роботи програми)

*	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки, завершення роботи програми)
1.7	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки, завершення роботи програми)

Табл. 4. Таблиця тестування програми

Висновок

Під час виконання лабораторної роботи ми ознайомились з різними алгоритмами сортування малих обсягів даних різного варіанту подання. Ми розглянули та визначили найефективніші способи використання того чи іншого алгоритму, в залежності від вхідних даних та умов задачі. Теоретична складність підтвердила практичну під час тестування. Результатами роботи є аналіз вибору алгоритму після порівняння практичної складності.

Відповіді на контрольні запитання

1. Сформулюйте задачу сортування.

Задача сортування полягає у впорядковуванні послідовності записів таким чином, щоб значення ключового поля складали монотонну послідовність.

2. Поясніть, виходячи з чого обирається алгоритм сортування для розв'язку конкретної задачі.

По-перше, з того що потрібно досягти у цій задачі. Якщо справа здебільшого складається з необхідности зробити швидкий та ефективний продукт, то й алгоритм повинен бути таким, щоб реалізувати ці вимоги. У деяких проєктах швидкість обробки інформації може не бути головною вимогою, тому, можна підібрати й інший алгоритм.

По-друге, є важливим фактор того, що саме хоче побачити користувач. Кожен має свій смак, і алгоритми у цій ситуації не є винятком. Хоча це й стоїть більше «на другому плані», все одно потрібно брати це до уваги.

3. Поясніть чому задача сортування в програмуванні досі повністю не визначена.

Тому що хоча й існує велика кількість алгоритмів сортування, все ж таки метою програмування є не лише розробка алгоритмів сортування елементів, але й розробка саме ефективних алгоритмів сортування.

Ми знаємо, що одну й ту саму задачу можна вирішити за допомогою різних алгоритмів і кожен раз зміна алгоритму приводить до нових, більш або менш ефективних розв'язків задачі. Основними вимогами до ефективності алгоритмів сортування є перш за все ефективність за часом

та економне використання пам'яті. Згідно цих вимог, прості алгоритми сортування (такі, як сортування вибором і сортування включенням) не є дуже ефективними.

4. Що таке стійкість алгоритмів?

Стійкістю алгоритмів називається такий алгоритм сортування, що не змінює порядок елементів з однаковим ключем. Найпоширеніша модель представлення даних для сортування — масив структур, в якому кожен елемент має поля англ. key (ключ по якому відбувається впорядкування) і їх значення англ. data (інша інформація).

5. Які з простих алгоритмів стійкі, а які ні? Поясніть.

За час O(n^2): сортування вставкою, сортування обміном

За час O(n*log(n)): сортування злиттям

За час O(n) з використанням додаткової інформації про елементи: сортування підрахунком, сортування за розрядами, сортування комірками

За час O(n*log(n)^2): сортування злиттям модифіковане

6. Зробіть порівняльну характеристику методів сортування вибором, сортування вставками, бульбашкового сортування. Вкажіть, на яких вхідних даних ці методи працюють найкраще.

В результаті проведеного дослідження та отриманих даних для сортування невідсортованого масиву найбільш оптимальним з представлених алгоритмів для сортування масиву є бульбашкою. Незважаючи на триваліший час виконання алгоритм споживає менше пам'яті, що може бути важливим у великих проєктах. Однак такі алгоритми, як сортування вибором та вставками, можуть краще підійти для наукових цілей,

наприклад, у навчанні, де не потрібно обробляти величезну кількість даних. При частково відсортованому масиві результати не сильно відрізняються, всі алгоритми сортування показують час приблизно на 2-3 мілісекунди менше.

7. Зробіть порівняльну характеристику методів сортування вибором та сортування методом підрахунку. Вкажіть, на яких вхідних даних ці методи працюють найкраще.

У кожного алгоритму сортування своя тимчасова та просторова складність. Використовувати можна будь-який із представлених алгоритмів залежно від поставлених завдань.

8. Порівняйте непряме (по індексам або покажчикам) та пряме сортування вибором. Вкажіть, на яких вхідних даних ці методи працюють найкраще.

Недоліки:

- час виконання сортування дуже мало залежить від того, наскільки впорядкований вхідний набір;
- процес знаходження мінімального елементу за один прохід набору не дає інформації про те, де може знаходитися мінімальний елемент на наступному проході цього набору.

Переваги:

- не зважаючи на всю простоту та очевидний примітивізм підходу, сортуванню вибором надається перевага у випадках, коли записи наборів великі, а ключі, по яким іде сортування, – малі.
- В цього типу задачах витрати ресурсів на переміщення елементів значно більші вартості операцій порівняння.

9. Перерахуйте та прокоментуйте переваги та недоліки сортування методом швидкого сортування для різних послідовностей даних. Вкажіть, на яких вхідних даних цей метод працює найкраще.

Що стосується складності швидкого сортування, то в найкращому випадку ми отримаємо складність Ω (n log n), а в найгіршому — O(n2). Окрім низької обчислювальної складності цьому алгоритму притаманні й інші переваги:

- на практиці це один з найбільш швидкодіючих алгоритмів внутрішнього сортування;
- алгоритм доволі простий як для розуміння, так і для реалізації;
- потребує лише O(n) пам’яті, для покращеної версії — O(1);
- дозволяє розпаралелювання для сортування підмасивів;
- працює на пов’язаних списках;
- є найефективнішим для сортування великої кількості даних.

Недоліками можна вважати:

- нестійкість;
- обчислювальна складність сильно деградує за умови невдалих вхідних даних (сортований масив).