# Лабораторна робота №13. Розробка програми з реалізацією простих алгоритмів пошуку і сортування

**Мета:**. Засвоєння навичок розробки програм з використанням алгоритмів пошуків та сортування.

**Хід роботи**

1. Ознайомитися з теоретичними відомостями.
2. Написати програму, що містить 2 функції, які реалізують два простих методи сортування - сортування вставками та "бульбашкове сортування".
3. Згенерувати три масиви з випадковими елементами типу **Integer** довжиною 100, 1000 та 10000 елементів, відповідно.
4. Відсортувати одержані масиви за збільшенням елементів, визначивши при цьому такі параметри:
   1. кількість порівнянь;
   2. кількість обмінів;
   3. фактичний час роботи,

Швидкодія сучасних ЕОМ/ПК може привести до того, що час роботи процедури буде дорівнювати нулеві з точністю, яку забезпечує системний таймер. Тоді варто запустити її багато разів у циклі для різних масивів, а потім усереднити результат.

1. В першому рядку програми записати

*// Група № Прізвище Номер ЛР*

вказавши номер своєї групи та своє прізвище.

1. Оформити звіт, навівши одержані експериментальні дані та теоретичні оцінки у вигляді таблиць.

Файл повинен мати назву в такому форматі:

**ОП+АМ <Номер групи><Номер лабораторної><Прізвище англійською>**

Наприклад, 21-01Ivanov.cpp.

**Тему в заголовку листа записати**

**ОП+АМ <Номер групи>-><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ОП+АМ-Запитання-<Номер групи>-<Прізвище >**.

.

**Строк відсилки ЛР для ІПЗ-21 09.11.2022**

**ІПЗ-22 09.11.2022**

**Теоретичні відомості**

Сортування вставками

Даний метод сортування називається сортування вставками, так як на і-му етапі відбувається "вставка" і-ого елемента a[i] в потрібну позицію серед елементів a[1], a[2], …, a[i-1], які вже впорядковані. Після цієї вставки перші і елементів будуть впорядковані.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Масив до впорядкування** | **22** | **20** | **-1** | **-40** | **88** | **-75** | **-22** |
| Перший перегляд масиву | 20 | 22 | -1 | -40 | 88 | -75 | -22 |
| Другий перегляд масиву | -1 | 20 | 22 | -40 | 88 | -75 | -22 |
| Третій перегляд масиву | -40 | -1 | 20 | 22 | 88 | -75 | -22 |
| Четвертий перегляд масиву | -40 | -1 | 20 | 22 | 88 | -75 | -22 |
| П'ятий перегляд масиву | -75 | -40 | -1 | 20 | 22 | 88 | -22 |
| Шостий перегляд масиву | -75 | -40 | -22 | -1 | 20 | 22 | 88 |

Рис 1. Сортування вставками

Серед критеріїв, за якими порівнюються методи сортування масивів, розрізняють:

* **C** (від **compare**) - кількість порівнянь ключів між собою;
* **M** (від **move**) - кількість операцій перезапису елементів з місця на місце у оперативній пам'яті або кількість обмінів;
* **T** (від **time**) - загальний час роботи процедури.

Аналіз. Кількість перевірок на **i**-му кроці дорівнює щонайбільше **i-1**, щонайменше **1**, тому в середньому - **i/2**. Тому в середньому загальна кількість перевірок



При цьому

C_min=n-1; C_max = (n-1)n/2.

Кількість обмінів **M** дорівнює щонайбільше **i**, щонайменше **0** на **i**-му кроці, тобто **i/2** у середньому. Тому

M_ave = (n-1)n/4

**Метод бульбашкового сортування**

Метод "бульбашкового сортування" ґрунтується на перестановці сусідніх елементів. Для впорядкування елементів масиву здійснюються повторні проходи по масиву. Переміщення елементів масиву здійснюється таким чином: масив переглядається зліва направо, порівнюються два сусідніх елементи; якщо елементи в парі розміщені в порядку зростання, вони лишаються без змін, а якщо ні - міняються місцями.

В результаті першого проходу найбільше число буде поставлене в кінець масиву. У другому проході такі операції виконуються над елементами з першого до (N-1)-ого, у третьому - від першого до (N-2)-ого і т.д. Впорядкування масиву буде закінчено, якщо при проході масиву не виконається жодної перестановки елементів масиву. Факт перестановки фіксується за допомогою деякої змінної, яка на початку має значення 0 і набуває значення 1 тоді, коли виконається перестановка в якій-небудь парі.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Масив до впорядкування** | **22** | **20** | **-1** | **-40** | **88** | **-75** | **-22** |
| Перший перегляд масиву | 20 | -1 | -40 | 22 | -75 | -22 | 88 |
| Другий перегляд масиву | -1 | -40 | 20 | -75 | -22 | 22 | 88 |
| Третій перегляд масиву | -40 | -1 | -75 | -22 | 20 | 22 | 88 |
| Четвертий перегляд масиву | -40 | -75 | -22 | -1 | 20 | 22 | 88 |
| П'ятий перегляд масиву | -75 | -40 | -22 | -1 | 20 | 22 | 88 |

**Рис 1.** Бульбашкове сортування

Кількість порівнянь для "бульбашкового методу":

C = (n-1)n/2

обмінів:

M_ave = (n-1)n/4

**Приклади алгоритмів сортування масивів**

**Сортування масиву** — один з найбільш розповсюджених процесів обробки даних. Завдяки йому здійснюється розміщення об’єктів у визначеному порядку, наприклад, чисел за зростанням або за спаданням їх значень, прізвищ у алфавітному порядку тощо. Існують різні методи сортування, серед них — обмінне сортування (метод «пухирця», «шейкер-сортування), сортування вибором, сортування вставками, швидке сортування, сортування Шелла, пірамідальне сортування, сортування обчисленням адреси, сортування порозрядним групуванням тощо. Ці методи відрізняються швидкістю отримання результату, складністю і універсальністю.

Розглянемо три методи сортування: обміном, вибором та вставками. Названі методи легко описуються у формі чітких алгоритмiв і передбачають нескладну програмну реалізацію, крім того вони цікаві тим, що моделюють природну поведінку люди­ни, яка здійснює сортування вручну. З іншого боку, вказані ме­тоди не досить ефективні і використовуються у випадках, коли необхідно відсортувати масиви невеликого розміру.

**Обмінне сортування** проілюструємо простим сортуванням обмiном — **методом «пухирця»**, який здійснюється шляхом перестановки елементів за визначеним правилом. У загальному випадку алгоритм сортування за цим методом наведений у **прикладі 1.4**. Розглянемо його більш детально.

Нагадаємо головні складові методу «пухирця»:

* крок сортування містить перегляд елементів масиву з по­чатку до кінця, при цьому розглядаються пари сусідніх елементів;
* елементи деякої пари міняються місцями у випадку, коли їх послідовність розташування не відповідає умові сортуван­ня (за зростанням або за спаданням).

***Приклад 6.11.*** Упорядкувати за зростанням методом “пухирця” масив **xі (і = 0…n-1), n = 5**, що має значення:

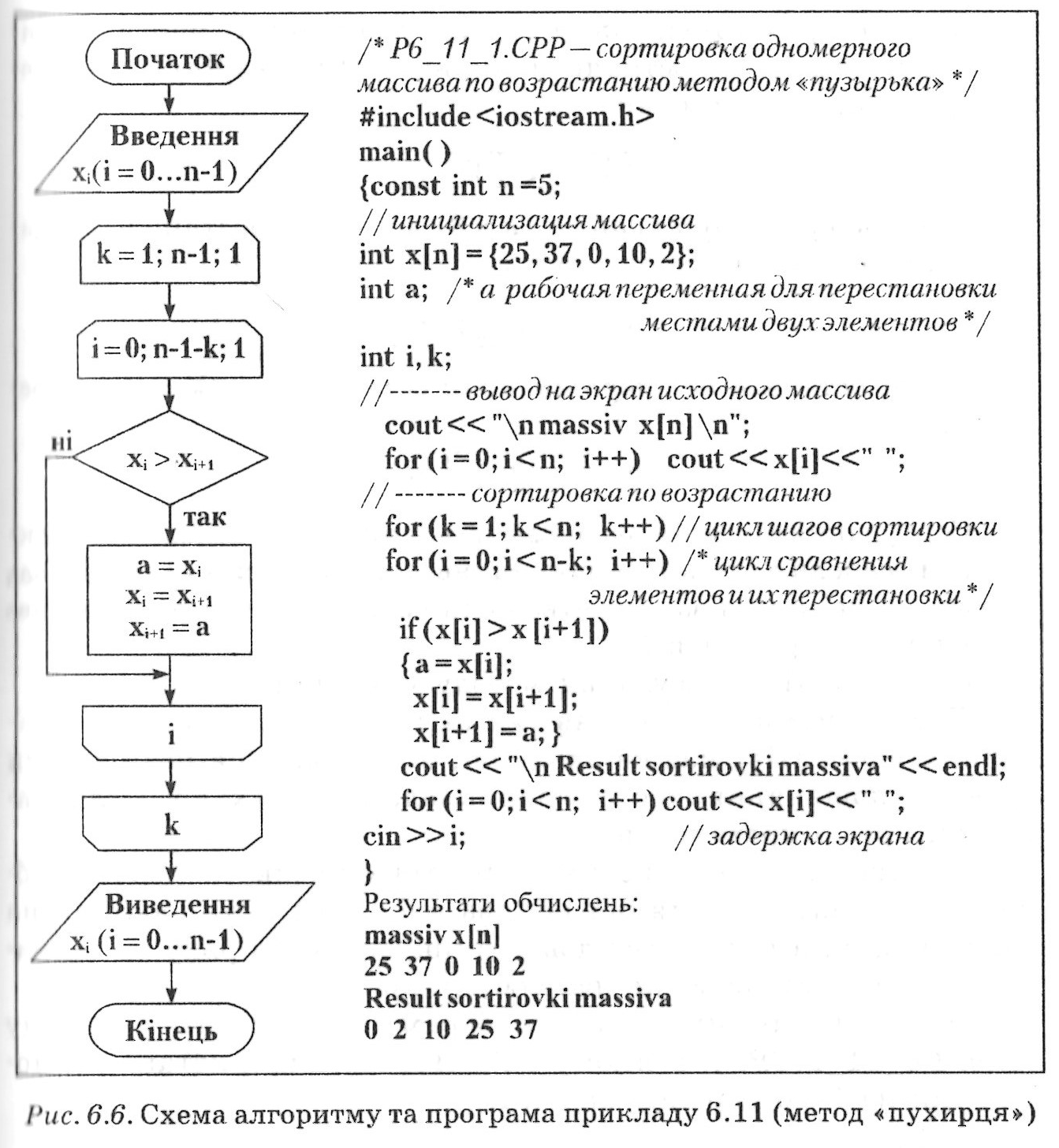
**x0 = 25; x1 = 37; х2 = 0; х3 = 10; х4 = 2.**

Слід зауважити, що **жоден метод сортування не** **досягає результату за один перегляд масиву, для цього застосовується, як правило, цикл у циклі.**

В алгоритмі сортування за методом «пухирця» (***див. рис. 6.6.***) з цією метою використано зовнішній цикл (цикл кроків сорту­вання) за параметром **k** та внутрішній цикл (цикл порівнянь перестановок) за параметром **і**. Оскільки кількість кроків сорту­вання має бути **n-1**, то параметр **k**зовнішнього циклу змінюеть­ся від 1 до **n-1** включно. На кожному кроці сортування у цик­лі за параметром **і** відбувається порівняння пар сусідніх елемен­тів та їх перестановка у випадку, коли пара розташована не за зростанням. Параметр **і** відповідає номеру елемента масиву.

**Перший крок сортування** **(k = 1)** здійснюється так:

* послідовно порівнюються пари сусідніх елементів («25» «37») і, якщо перший елемент більше за другий, вони міняються місцями, тобто на друге місце, як пухирець, «спливає» більший з двох елементів (у даному випадку елементи залишаються на своїх місцях, елемент, що «спливає», виділенні шрифтом):
* потім другий елемент («37»), більший з двох, порівнюється з третім елементом («0»), і на третє місце «спливає» більший з трьох («37»):
* далі третій елемент («37») порівнюється з четвертим елементом («10»):
* перегляд продовжується до кінця масиву, і найбільший елемент «спливе» та займе останнє місце у масиві:

[](http://cpp.dp.ua/uploads/posts/2015-12/1450820334_6_4.png)  
**Другий крок сортування (k = 2)**, у результаті якого «спливає» вправо на передостаннє місце наступний найбільший елемент, дає таке розміщення елементів:

**Третій крок сортування (k = 3)** дозволяє розташувати наступний за величиною елемент третім справа:

І  далі останній **четвертий крок сортування (k = 4)** дає результат:

Сортування масивів за методом «пухирця» є найменш ефективним, середня кількість порівнянь дорівнює **(n2 – n)/2**. Незважаючи на це, метод залишається одним з найпопулярніших завдяки простоті реалізації.

Особливості застосування методу «пухирця»:

* алгоритми сортування, як за зростанням, так і за спаданням елементів, майже однакові, відрізняються вони тільки знаками порівняння, тобто, наприклад, замість **xі > хі+1** слід записати прямо протилежне — **xi < хі+1**
* присутність однакових елементів у масиві не додає проблем: у момент порівняння обидва елементи залишаються ні своїх місцях, а потім послідовно зміщуються по ряду та займають своє остаточне місце на сусідніх позиціях. Наведемо другий варіант програмної реалізації прикладу (**див. Р6\_11\_2.СРР**), в якому ім’я масиву використано як покажчик на його перший елемент:

/\* Рr6\_11 \_2.СРР — сортировка одномерного массива по возрастанию методом ***«***пузырька***»*** \*/

//--- использование имени массива как указателя

**#inchide <iostream.h>**

**#include <conio.h>**

**main( )**

**{ const int n = 5;**

**int x[n], i, k;  int а;** /\* а — рабочая переменная для перестановки местами двух элементов \*/

//----------- ввод исходного массива

**for (і = 0; і < n; i++)**

**сіn >> \*(x+i);**

//----------- вывод на экран исходного массива

**cout << "\n massiv х[n] \n";**

**for (i = 0; i < n; i++)**

**cout << \*(x+i) << " ";**

//------------- сортировка no возрастанию

**for (k = 1; k < n; k++)**     // цикл шагов сортировки

**for (і = 0; і < n-k; і++)**    /\* цикл сравненья элементов и их перестановки \*/

**if (\*(x+i) > \*(x+i+1))**

**{ а = \*(x+i);**

**\*(x+i) = \*(x+i+1);**

**\*(x+i+1) = a; }**

**cout << "\n Result sortirovki massiva " << endl;**

**for (i=0; і < n; i++)**

**cout << \*(x+i) << " ";**

**getch();**

**}**

***Приклад 6.12.*** Для матриці **matr(5,6)** знайти суми елементів кожного рядка та записати їх в одновимірний масив. Отриманий масив відсортувати за зростанням методом «пухирця».

// Р6 12.СРР - использование двумерных массивов

/\* занесение сумм элементов строк матрицы в массив и сортировка его по возрастанию методом "пузырька" \*/

**#include <iostream.h>**

**#include <conio.h>**

**void main()**

**{ int matr[5][6], mas[5];**    // mas[ ] — массив сумм  строк

**int і, j, sum, stk;**

//------ ввод матрицы matr(5,6)

**cout << "Vvod matr[5][6] \n";**

**for (і = 0; і < 5; і++)**

**for (j = 0; j < 6; j++)**

**cin >> matr[i][j];**

//----- формирование массива сумм элементов строк

**for (і = 0; sum = 0; і < 5; і++)**

**{** // нахождение суммы элементов строки

**for (j = 0; j < 6; j++)**

**sum += matr[i][j];**

**mas[i] = sum }**      //занесение суммы строки в массив

**cout << "\nMassiv summ el. strok" << endl;**

**for (і = 0; і < 5; i++)**

**cout << mas[i] << " ";**

//----- сортировка массива сумм по возрастанию

**for (і = 1;   і < 5; і++)**   // цикл шагов сортировки

**for (j = 0; j < 5-і; j++)**

// цикл сравнен. и перестан. элем.

**if (mas[j] > mas[j+1])**

**{ stk = mas[j];**  // stk — для перестановки элементов

**mas[j] = mas[j+1];**

**mas[j+1] = stk; }**

**cout << "\nOtsortirovanniy massiv \n";**

**for (i = 0; і < 5; i++)**

**cout << mas[i] << " ";**

**getch();**            //задержка экрана вывода результата

**}**

Результати обчислень:  
**Vvod matr[5][6]**

**1 6 4 9 3 2**

**2 8 5 7 3 1**

**0 7 5 0 3 2**

**6 3 9 2 9 4**

**0 4 8 3 9 2**

**Massiv summ elementov strok**

**25 26 21 33 31**

**Otsortirovanniy massiv**

**21 25 26 31 33**

У програмі використані типові прийоми алгоритмізації – накопичення суми **sum** та формування робочого масиву **masj**, що запам’ятовує суми елементів кожного рядка матриці. Застосування цих прийомів докладно пояснене відповідно у **прикладі 1.2** та **прикладі 1.5.** Кількість елементів створеного масиву дорівнює кількості рядків матриці. Для сортування отриманого масиву за зростанням використано сортування методом «пухирця».

#### [Фрагмент лекції професора Девіда Малана із курса CS50 сортування “бульбашкою”](http://cpp.dp.ua/sortuvannya-bulbashkoyu/)

**(повний курс https://habr.com/ru/company/vertdider/blog/403823/)**

**Сортування за методом вибору** розглянемо на прикладі.

***Приклад 6.13.*** Упорядкувати за зростанням методом вибору вихiдний масив, що має такі значення:

**x0 = 20; x1 = 1; х2 = 30; х3 = 2; х4 = 7, х5 = 5.**

Схему алгоритму та програму реалізації даної задачі наведено на **рис. 6.7.**

Процес сортування за зростанням здійснюється за кроками. Позначимо номер кроку сортування параметром **і**. На кожному кроці шукається найменший елемент, що міняється місцями з елементом, номер якого збігається з номером кроку **і**.

**Нульовий крок сортування** **(і = 0):**

у процесі розгляду елементів масиву, починаючи з першого, знаходять найменший елемент («1») і розташовують його на місце першого елемента, а перший («20») — на місце мінімального. У результаті найменший елемент масиву потрапляє на нульову позицію, тобто на перше місце зліва (підкресленi елементи, що переставляються):

http://cpp.dp.ua/uploads/posts/2015-12/1450821131_6_4_2.png

**Перший крок сортування (і = 1):**

наступний найменший елемент («2»)знаходять у частинi масиву, що починається з першої позиції. Він міняється місцями з другим елементом («20»), тобто другий за значенням елмент («2») розташується на першій позиції, а саме на другому місці зліва:

http://cpp.dp.ua/uploads/posts/2015-12/1450821231_6_4_3.png

**Другий крок сортування** **(і = 2):**

третій за значенням елемент («5») знайдемо у масиві, починаючи з другої позиції («З0»), та поміняємо його місцем з елементом  масиву, що розташувався на другій позиції:

http://cpp.dp.ua/uploads/posts/2015-12/1450821238_6_4_4.png

Подальші кроки сортування дають такі перетворення: **третій крок сортування** **(і = 3):**

http://cpp.dp.ua/uploads/posts/2015-12/1450821258_6_4_5.png

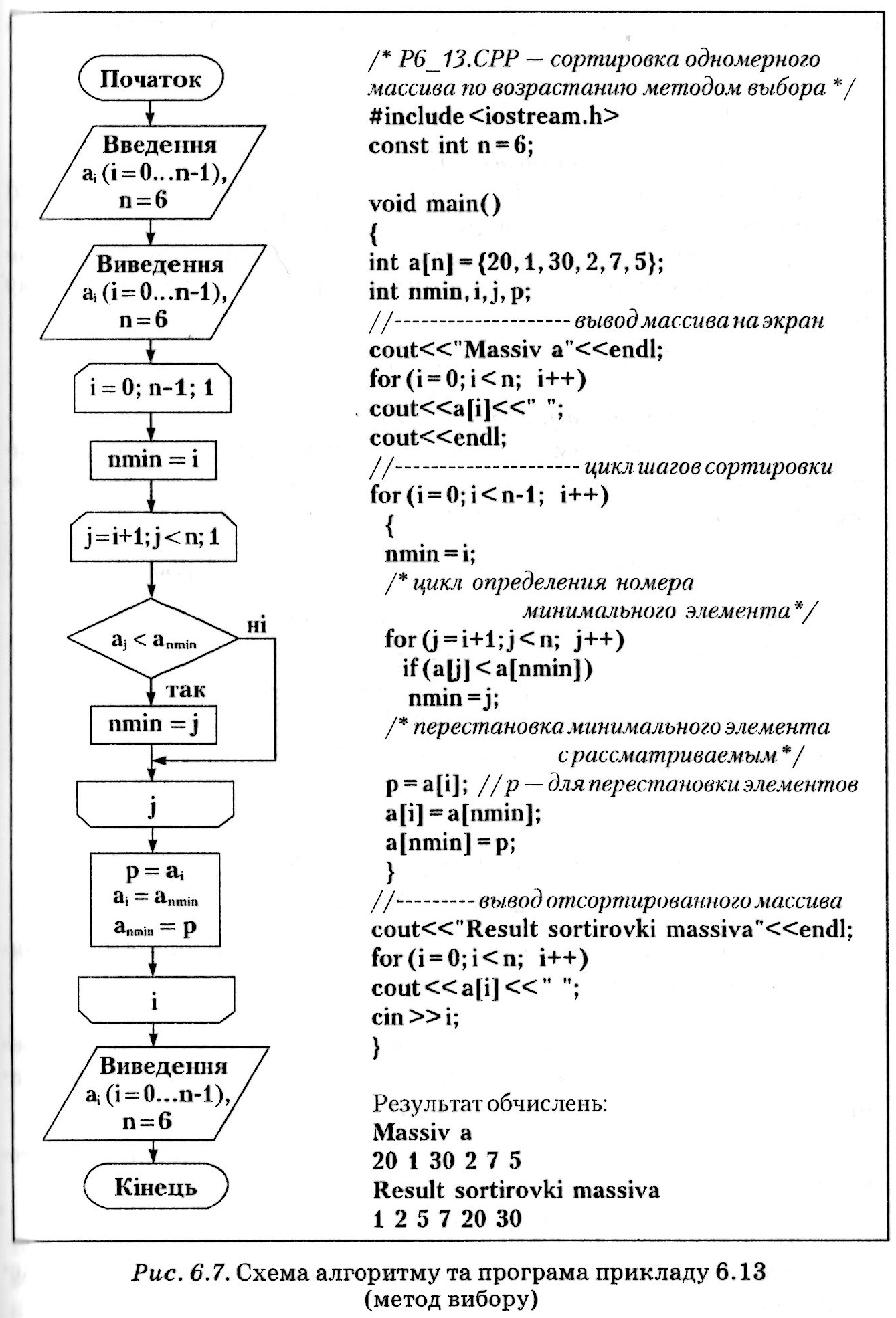
**Четвертий крок сортування (і=4):**

http://cpp.dp.ua/uploads/posts/2015-12/1450821360_6_4_6.png

У результаті маємо відсортований за зростанням масив:

http://cpp.dp.ua/uploads/posts/2015-12/1450821340_6_4_7.png

Загальна кількість дій алгоритму сортування за методом вибору дорівнює **(n2/4 + 3\*n)** операціям.

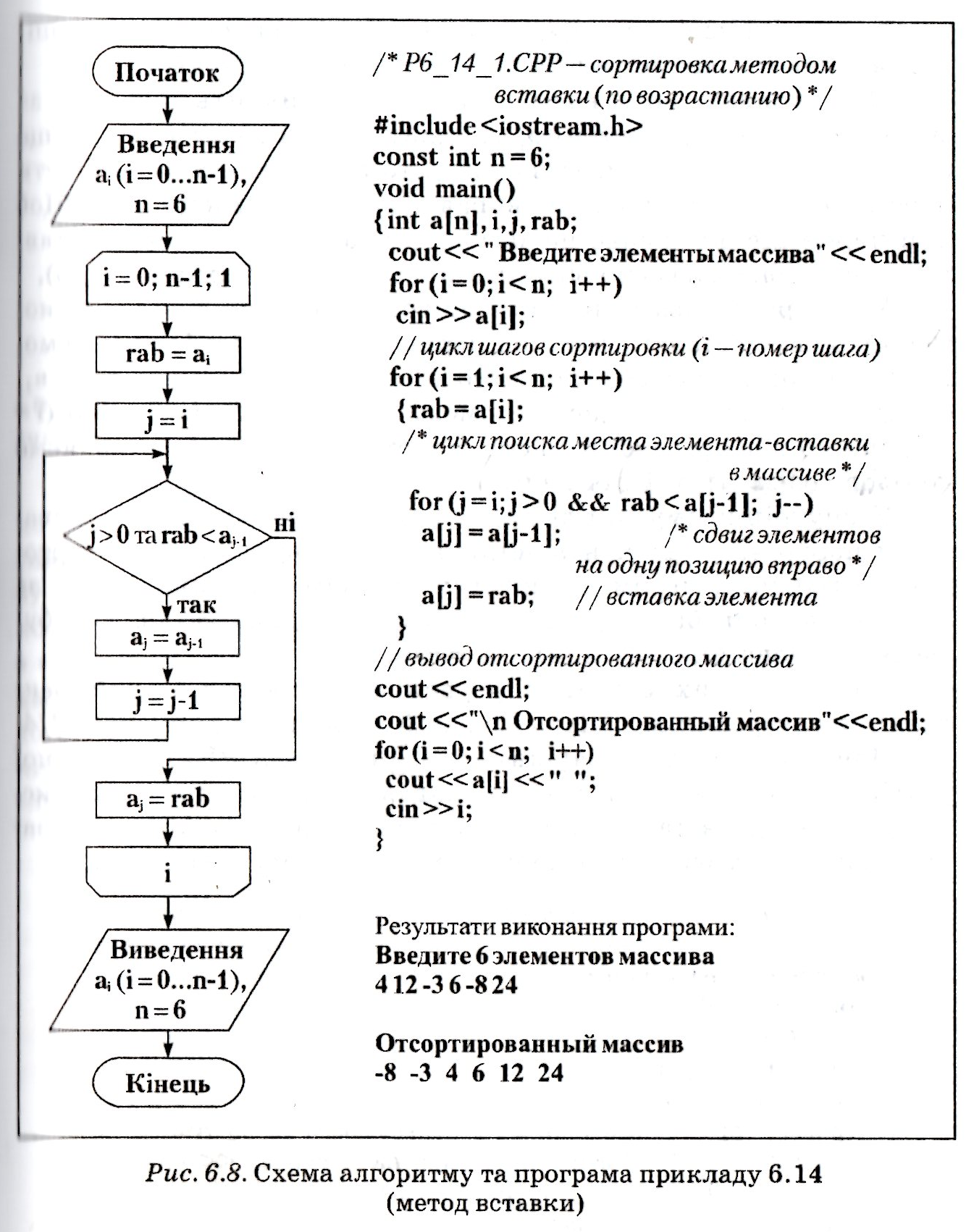


#### [Фрагмент лекції професора Девіда Малана із курса CS50 сортування вибором](http://cpp.dp.ua/sortuvannya-vyborom/)

Сортування за методом вставки полягає в тому, що на кожному кроці відбувається вставка елемента у відсортований масив. Розглянемо цей метод на конкретному прикладі.

***Приклад 6.14.*** Упорядкувати за зростанням (за спаданням) методом  вставки масив **аі (і = 0…n-1), n = 6**, що має такі значення:

**a0 = 4; а1 = 13; a2 = -3; а3 = 6; a4 = \_8, a5 = 24.**

Схему алгоритму та програму реалізації цієї задачі наведено на **рис. 6.8.**  
[](http://cpp.dp.ua/uploads/posts/2015-12/1450906389_6_4_9.png)

**Сутність алгоритму сортування за методом вставки:**

* якщо **j (а0…аj-1)** елементів масиву **а** відсортовані за зростанням, а елемент **aj** має довільне значення, потрібно порівнювати цей елемент по черзі з елементами **аj-1, aj-2,…**, доки для **aj** не знайдеться місце у відсортованому масиві. При цьому всі розглянуті елементи **aj-1, aj-2** **,…**,що будуть за значенням більш ніж **aj** , мають переміститися на одну позицію вправо;
* у випадку, коли елемент-вставка **aj** виявиться більше за значенням , ніж **аj-1**, елемент **aj** залишиться на своєму місці. Якщо **a**j буде менше всіх елементів **а0…аj-1**, то всі елементи мають бути переміщені на одну позицію вправо, a **aj** займе місце **j = 0**. Що при переміщенні вправо не загубити значення елемента-вставки aj, потрібно завчасно зберегти його в робочій змінній (**rab**)

У програмі параметр **і** зовнішнього циклу відповідає за номер елемента-вставки, тому на першому кроці **(і = 1)** вважаємо що відсортований масив, до якого вставляється елемент **rab = aj** складається тільки з одного елемента, на наступному кроці **(і  = 2)** маємо відсортований масив з двох елементів, а вставляємо до нього елемент **а2** і так далі.

У внутрішньому циклі програми за параметром **j** відбувається порівняння елемента-вставки **rab** з елементами відсортованого масиву і переміщення цих елементів вправо, якщо за значенням вони більші, ніж **rab**. Безпосередньо вставка відбувається у зовнішньому циклі: **aj = rab.**

З розглянутих методів сортування цей метод — найефективніший, середня кількість операцій приблизно дорівнює **n2 / 4.** Наведемо ще один варіант (**див. Р6\_14\_2.СРР**) програмної реалізації сортування масиву за спаданням елементів з використанням методу вставки, в якій використано покажчики та застосовано цикл while для пошуку мiсця елемента вставки.

//Р6\_14\_2.СРР — сортировка методом вставки (по убыванию)

//---------------------- использование указателей

**#include <iostream.h>**

**#include <conio.h>**

**const int n=6;**

**main ( )**

**{ int a [n], i, j, rab;**

**int \*pmas;**

**pmas = a;** // pmas - &a[0] — указатель на начало массива

**cout << "\*\*\*\*\* Введите 6 элементов массива" << endl;**

**for (і = 0; і < n; i++)**

**cin >> \*pmas++;**

**pmas = a;**    //pmas -= n; — указатель на начало массивa

**for (і = 1; і < n; і++)**

**{ rab= \*(pmas + і);**

**j = і;**

**while (j > 0 && rab > \*(pmas + j-1))**

**{ \*(pmas + j) = \*(pmas + j-1);**

**j-- }**

**\*(pmas + j) = rab; }**

**cout << "Отсортированный массив по убыванию" << endl;**

**for (і = 0; і < n; i++)**

**cout << \*(pmas + i) << " ";**

**getch();**

**}**

Результати виконання програми:

**Введите 6 элементов массива**

**4 12 -3 6 -8 24**

**Отсортированный массив по убыванию**

**24 12 6 4 -3 -8**

#### [Фрагмент лекції професора Девіда Малана із курса CS50 сортування вибором](http://cpp.dp.ua/sortuvannya-vstavkoyu/)

***Приклад 6.15.*** З двох впорядкованих за зростанням масивів **ai (i = 0...n-1) і bj (j = 0…m-1)** сформувати третій, також впорядкований, масив без додаткового сортування **(n = 8, m = 12).**

Алгоритм **злиття двох відсортованих масивів** (**див. рис. 6.9)** починається з порівняння перших елементів відповідних масивiв **а і b**. За лічильником і будемо вибирати елементи з масиву **а**, за лічильником  **j** — з масиву **b**, а за параметром  **k** — заносити елементи до масиву с. У масив с заноситься менший з елементів, що порівнюються, а далі в порівнянні бере участь наступний елемент того масиву, елемент якого вже записаний до с. Ця процедура повторюється, доки один з масивів не закiнчиться. Наприкінці слід тільки переписати до масиву с всі елементи іншого масиву, що залишилися.

/\* Р6\_15.СРР ***—*** из двух массивов***,*** отсортированных по возрастанию***,*** сформировать третий \*/

**#include <iostream.h>**

**#include <conio.h>**

**const int n=8, m=12;**

**void main()**

**{ int a[n], b[m], c[n+m];**

**int i, j, k, 1;**

//--------------- ввод двух отсортированных массивов

**cout << "Vvedite 2 otsortirovanih massiva" << endl;**

**for (i = 0; і < n; i++)**

**cin >> a[i];**

**for (j = 0; j < m; j++)**

**cin >> b[j];**

//------------------- слияние массивов a и b в массив с

**for (і = 0, j = 0, к = 0; і < n && j < m; k++)**

**if (a[i] < b[j]) c[k] = a[i++];**

**else c[k] = b[j++];**

**if (i = = n)**

**for (l = k; l < n+m; l++)**

**c[l] = b[j++];**

**else**

**for (l = k; l < n+m; l++)**

**c[l] = a[i++];**

//-------------- вывод полученного массива с

**cout << "Rezult — massiv c" << endl;**

**for (і = 0; і < n+m; i++)**

**cout << c[i] << " ";**

**getch();**

**}**

Результати обчислень:

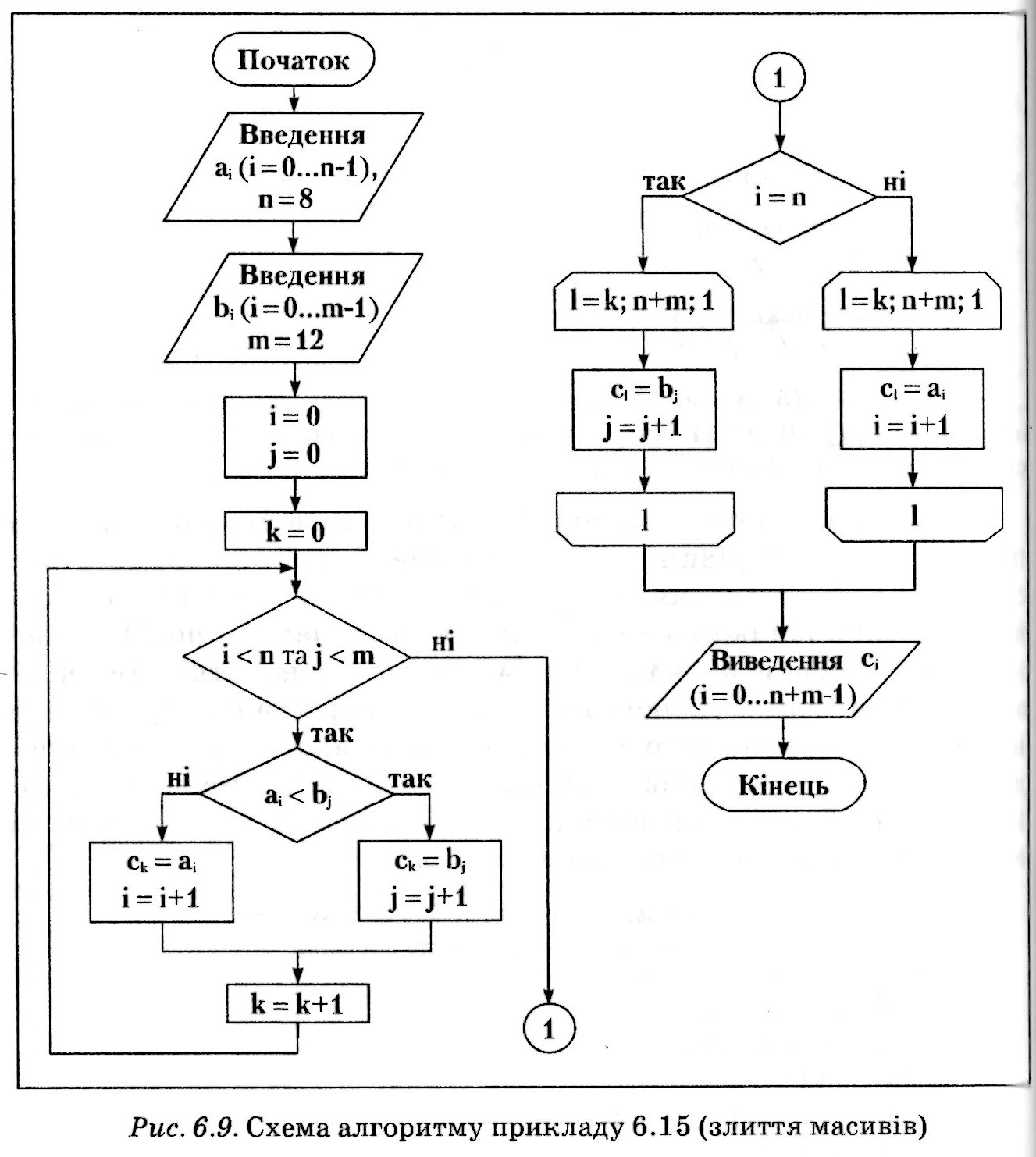
**Vvedite 2 otsortirovanih massiva**

**1 4 11 17 33 54 60 62**

**1 5 9 14 15 19 20 23 44 48 72 80**

**Rezult — massiv c**

**1 2 4 5 9 11 14 15 17 19 20 23 33 44 48 54 60 62 72 80**

[](http://cpp.dp.ua/uploads/posts/2015-12/1450821386_6_4_8.png)