Сортування бульбашкою*.*

Сортування бульбашкою - найпростіший алгоритм сортування, застосовуваний чисто для навчальних цілей. Практичного застосування цього алгоритму немає, так як він не ефективний, особливо якщо необхідно впорядкувати масив великого розміру. До плюсів сортування бульбашкою відноситься простота реалізації алгоритму.

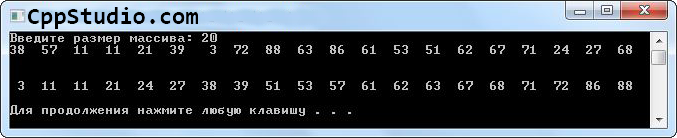
Алгоритм сортування бульбашкою зводиться до повторення проходів за елементами сортованого масиву. Прохід по елементам масиву виконує внутрішній цикл. За кожен прохід порівнюються два сусідні елементи, і якщо порядок невірний елементи міняються місцями. Зовнішній цикл буде працювати до тих пір, поки маса не буде відсортований. Таким чином зовнішній цикл контролює кількість спрацьовувань внутрішнього циклу Коли при черговому проході за елементами масиву не буде здійснено жодної перестановки, то масив буде вважатися відсортованим. Щоб добре зрозуміти алгоритм, відсортуємо методом бульбашки масив, наприклад, з 7 чисел (див. Таблиця 1).  
**початковий масив: 3 3 7 1 2 5 0**

| Таблиця 1 - Сортування бульбашкою | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№ ітерації** | **Елементи масиву** | | | | | | | **Перестановки** |
| поч. масив | 3 | 3 | 7 | 1 | 2 | 5 | 0 |  |
| 0 | 3 | 3 |  |  |  |  |  | false |
| 1 |  | 3 | 7 |  |  |  |  | false |
| 2 |  |  | **1** | **7** |  |  |  | 7>1, true |
| 3 |  |  |  | **2** | **7** |  |  | 7>2, true |
| 4 |  |  |  |  | **5** | **7** |  | 7>5, true |
| 5 |  |  |  |  |  | **0** | **7** | 7>0, true |
| пот. масив | **3** | **3** | **1** | **2** | **5** | **0** | **7** |  |
| 0 | 3 | 3 |  |  |  |  |  | false |
| 1 |  | **1** | **3** |  |  |  |  | 3>1, true |
| 2 |  |  | **2** | **3** |  |  |  | 3>2, true |
| 3 |  |  |  | **0** | **3** |  |  | 3>0, true |
| 4 |  |  |  |  | 3 | 5 |  | false |
| 5 |  |  |  |  |  | 5 | 7 | false |
| пот. масив | **3** | **1** | **2** | **0** | **3** | **5** | **7** |  |
| 0 | **1** | **3** |  |  |  |  |  | 3>1, true |
| 1 |  | **2** | **3** |  |  |  |  | 3>2, true |
| 2 |  |  | **0** | **3** |  |  |  | 3>0, true |
| 3 |  |  |  | 3 | 3 |  |  | false |
| 4 |  |  |  |  | 3 | 5 |  | false |
| 5 |  |  |  |  |  | 5 | 7 | false |
| пот. масив | **1** | **2** | **0** | **3** | **3** | **5** | **7** |  |
|  | 1 | 2 |  |  |  |  |  | false |
|  |  | **0** | **2** |  |  |  |  | 2>0, true |
|  |  |  | 2 | 3 |  |  |  | false |
|  |  |  |  | 3 | 3 |  |  | false |
|  |  |  |  |  | 3 | 5 |  | false |
|  |  |  |  |  |  | 5 | 7 | false |
| пот. масив | **1** | **0** | **2** | **3** | **3** | **5** | **7** |  |
|  | **0** | **1** |  |  |  |  |  | 1>0, true |
|  |  | 1 | 2 |  |  |  |  | false |
|  |  |  | 2 | 3 |  |  |  | false |
|  |  |  |  | 3 | 3 |  |  | false |
|  |  |  |  |  | 3 | 5 |  | false |
|  |  |  |  |  |  | 5 | 7 | false |
| кінцевий масив | **0** | **1** | **2** | **3** | **3** | **5** | **7** |  |
| Конец сортировки | | | | | | | | |

Для того щоб впорядкувати масив вистачило п'яти запусків внутрішнього циклу, for. Запустити, цикл for спрацьовував 6 разів, так як елементів в масиві 7, то ітерацій (повторень) циклу for має бути на одне менше. На кожній ітерації порівнюються два сусідні елементи масиву. Якщо поточний елемент масиву повинна перевищувати, то міняємо їх місцями. Таким чином, поки маса не буде відсортований, буде запускатися внутрішній цикл і виконуватися операція порівняння. Зверніть увагу на те, що за кожне повне виконання циклу for як мінімум один елемент масиву знаходить своє місце. У гіршому випадку, знадобиться n-2 запуску внутрішнього циклу, де n - кількість елементів масиву. Це говорить про те, що сортування бульбашкою вкрай неефективна для великих масивів.

Розробимо програму, в якій спочатку необхідно ввести розмір одновимірного масиву, після чого масив заповнюється випадковими числами і сортується методом бульбашки.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | // bu\_sort.cpp: визначає точку входу для консольного застосування.    #include "stdafx.h"  #include <iostream>  #include <iomanip>  #include <ctime>  using namespace std;    void bubbleSort(int \*, int); // прототип функції сортування бульбашкою    int main(int argc, char\* argv[])  {      srand(time(NULL));      setlocale(LC\_ALL, "rus");      cout << " Введіть розмір масиву: ";      int size\_array; // довжина масиву      cin >> size\_array;        int \*sorted\_array = new int [size\_array]; // одновимірний динамічний масив      for (int counter = 0; counter < size\_array; counter++)      {          sorted\_array[counter] = rand() % 100; // заповнюємо масив випадковими числами          cout << setw(2) << sorted\_array[counter] << "  "; // вивід масиву на екран      }      cout << "\n\n";        bubbleSort(sorted\_array, size\_array); // виклик функції сортування бульбашкою        for (int counter = 0; counter < size\_array; counter++)      {          cout << setw(2) << sorted\_array[counter] << "  "; // друк відсортованого масиву      }      cout << "\n";        system("pause");      return 0;  }    void bubbleSort(int\* arrayPtr, int length\_array) // сортування бульбашкою  {   int temp = 0; // тимчасова змінна для зберігання елемента масиву   bool exit = false; // больова змінна для виходу з циклу, якщо масив відсортований   while (!exit) // поки маса не впорядкований   {    exit = true;    for (int int\_counter = 0; int\_counter < (length\_array - 1); int\_counter++) // внутрішній цикл      // сортування бульбашкою по зростанню - знак >      // сортування бульбашкою по спадаючій - знак <      if (arrayPtr[int\_counter] > arrayPtr[int\_counter + 1]) // порівнюємо два сусідні елементи      {       // виконуємо перестановку елементів масиву       temp = arrayPtr[int\_counter];       arrayPtr[int\_counter] = arrayPtr[int\_counter + 1];       arrayPtr[int\_counter + 1] = temp;       exit = false; // на черговій ітерації була проведена перестановка елементів    }   } |

Результат роботи програми показаний на малюнку 1.

Сортування вставками

Сортування вставками - досить простий алгоритм. Як в і будь-якому іншому алгоритмі сортування, зі збільшенням розміру сортованого масиву збільшується і час сортування. Основною перевагою алгоритму сортування вставками є можливість сортувати масив у міру його полученія.То є маючи частина масиву, можна починати його сортувати. У паралельному програмування така особливість відіграє не останню роль.

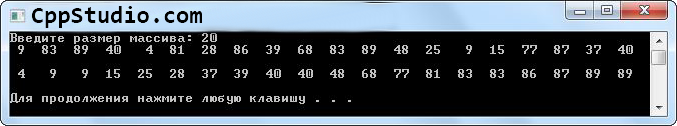
Сортований масив можна розділити на дві частини - відсортована частина і несортованими. На початку сортування перший елемент масиву вважається відсортованим, все інші - не відсортовані. Починаючи з другого елементу масиву і закінчуючи останнім, алгоритм вставляє невідсортоване елемент масиву в потрібну позицію в відсортованої частини масиву. Таким чином, за один крок сортування відсортована частина масиву збільшується на один елемент, а не відсортованого частина масиву зменшується на один елемент. Розглянемо приклад сортування за зростанням масиву з 7 чисел (див. Таблиця 1):  
початковий масив: 3 3 7 1 2 5 0

| Таблиця 1 - Сортування вставками | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **крок** | **відсортована частина масиву** | | | | | | | **пот. елемент** | **вставка** |
| **1** | 3 |  |  |  |  |  |  | 3 | false |
| **2** | 3 | 3 |  |  |  |  |  | 7 | false |
| **3** | 3 | 3 | 7 |  |  |  |  | 1 | true |
| **4** | 1 | 3 | 3 | 7 |  |  |  | 2 | true |
| **5** | 1 | 2 | 3 | 3 | 7 |  |  | 5 | true |
| **6** | 1 | 2 | 3 | 3 | 5 | 7 |  | 0 | true |
| **—** | 0 | 1 | 2 | 3 | 3 | 5 | 7 | — | — |

На кожному кроці сортування порівнюється поточний елемент з усіма елементами в відсортованої частини. На першому кроці порівнюється трійка з трійкою, вони рівні тому не змінюємо їх місцями. На другому кроці порівнюємо 7 з двома трійками, 7> 3 а так як сортування за зростанням, то знову елементи масиву залишаються на своїх місцях. На третьому кроці одиниця порівнюється з трьома елементами і всі вони більше одиниці, значить одиницю вставляємо на перше місце, в початок масиву. На четвертому кроці поточний елемент - 2 порівнюємо з елементами 1, 3, 3, 7. Виходить, що 1 <2 <3 і 7 тому двійку вставляємо між одиницею і трійкою. П'ятий і шостий кроки виконуються точно також. У підсумку на шостому кроці ми отримуємо відсортований за зростанням масив. Запрограмуємо алгоритм сортування вставками на С ++.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | // insertion\_sort.cpp: визначає точку входу для консольного застосування.    #include "stdafx.h"  #include <iostream>  #include <ctime>  #include <iomanip>  using namespace std;    void insertionSort(int \*, int); // прототип функції сортування вставками    int main(int argc, char\* argv[])  {      srand(time(NULL));      setlocale(LC\_ALL, "rus");      cout << " Введіть розмір масиву: ";      int size\_array; // довжина масиву      cin >> size\_array;        int \*sorted\_array = new int [size\_array]; // одновимірний динамічний масив      for (int counter = 0; counter < size\_array; counter++)      {          sorted\_array[counter] = rand() % 100; // заповнюємо масив випадковими числами          cout << setw(2) << sorted\_array[counter] << "  "; // висновок масиву на екран      }      cout << "n";        insertionSort(sorted\_array, size\_array); // виклик функції сортування вставками        for (int counter = 0; counter < size\_array; counter++)      {          cout << setw(2) << sorted\_array[counter] << "  "; // друк відсортованого масиву      }      cout << "n";      delete [] sorted\_array; // вивільняє пам'ять      system("pause");      return 0;  }    void insertionSort(int \*arrayPtr, int length) // сортування вставками  {      int temp, // тимчасова змінна для зберігання значення елемента сортованого масиву          item; // індекс попереднього елемента      for (int counter = 1; counter < length; counter++)      {          temp = arrayPtr[counter]; // инициализируем тимчасову змінну поточним значенням елемента масиву          item = counter-1; // запам'ятовуємо індекс попереднього елемента масиву          while(item >= 0 && arrayPtr[item] > temp) // поки індекси не дорівнює 0 і попередній елемент масиву більше поточного          {              arrayPtr[item + 1] = arrayPtr[item]; // перестановка елементів масиву              arrayPtr[item] = temp;              item--;          }      }  } |

Програма сортує одновимірний масив по зростанню. Змінивши знак> в **рядку 47** масив буде сортуватися по спадаючій. Результат роботи програми показаний на малюнку 1.



Сортування вибором

Мабуть, найпростіший алгоритм угруповань - це сортування вибором. Судячи з назви сортування, необхідно щось вибирати (максимальний або мінімальний елементи масиву). Алгоритм сортування вибором знаходить у вихідному масиві максимальний або мінімальний елементи, в залежності від того як необхідно сортувати масив, по зростанням або за спаданням. Якщо масив повинен бути відсортований по зростанню, то з вихідного масиву необхідно вибирати мінімальні елементи. Якщо ж масив необхідно впорядкувати за спаданням, то вибирати слід максимальні елементи.

Припустимо необхідно впорядкувати масив по зростанню. У вихідному масиві знаходимо мінімальний елемент, міняємо його місцями з першим елементом масиву. Уже, з усіх елементів масиву один елемент стоїть на своєму місці. Тепер будемо розглядати не відсортовану частину масиву, тобто всі елементи масиву, крім першого. У невідсортоване частини масиву знову шукаємо мінімальний елемент. Знайдений мінімальний елемент міняємо місцями з другим елементом масиву і т. Д. Таким чином, суть алгоритму сортування вибором зводиться до багаторазового пошуку мінімального (максимального) елементів в невідсортоване частини масиву. Відсортуємо масив з семи чисел згідно з алгоритмом «Сортування вибором».

**вихідний масив: 3 3 7 1 2 5 0**

1) Отже, знаходимо мінімальний елемент в масиві. 0 - мінімальний елемент

2) Міняємо місцями мінімальний і перший елементи масиву.

Поточний масив: 0 3 7 1 2 5 3

3) Знаходимо мінімальний елемент в невідсортоване частини масиву. 1 - мінімальний елемент

4) Міняємо місцями мінімальний і перший елементи масиву.

Поточний масив: 0 1 7 3 2 5 3

5) min = 2

6) Поточний масив: 0 1 2 3 7 5 3

7) min = 3

8) Поточний масив: 0 1 2 3 7 5 3 в масиві нічого не змінилося, так як 3 стоїть на своєму місці

9) min = 3

10) Кінцевий вигляд масиву: 0 1 2 3 3 5 7 - масив відсортований

Запрограмуємо алгоритм сортування вибором в С ++.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | // sorting\_choices.cpp: визначає точку входу для консольного застосування.    #include "stdafx.h"  #include <iostream>  #include <ctime>  #include <iomanip>  using namespace std;    void choicesSort(int\*, int); // прототип функції сортування вибором    int main(int argc, char\* argv[])  {      srand(time(NULL));      setlocale(LC\_ALL, "rus");      cout << " Введіть розмір масиву: ";      int size\_array; // довжина масиву      cin >> size\_array;        int \*sorted\_array = new int [size\_array]; // одновимірний динамічний масив      for (int counter = 0; counter < size\_array; counter++)      {          sorted\_array[counter] = rand() % 100; // заповнюємо масив випадковими числами          cout << setw(2) << sorted\_array[counter] << "  "; // висновок масиву на екран      }      cout << "\n\n";        choicesSort(sorted\_array, size\_array); // виклик функції сортування вибором        for (int counter = 0; counter < size\_array; counter++)      {          cout << setw(2) << sorted\_array[counter] << "  "; // друк відсортованого масиву      }      cout << "\n";      delete [] sorted\_array; // вивільняє пам'ять      system("pause");      return 0;  }    void choicesSort(int\* arrayPtr, int length\_array) // сортування вибором  {      for (int repeat\_counter = 0; repeat\_counter < length\_array; repeat\_counter++)      {          int temp = arrayPtr[0]; // тимчасова змінна для зберігання значення перестановки          for (int element\_counter = repeat\_counter + 1; element\_counter < length\_array; element\_counter++)          {              if (arrayPtr[repeat\_counter] > arrayPtr[element\_counter])              {                  temp = arrayPtr[repeat\_counter];                  arrayPtr[repeat\_counter] = arrayPtr[element\_counter];                  arrayPtr[element\_counter] = temp;              }          }      }  } |

Алгоритм сортування вибором заснований на алгоритмі пошуку максимального (мінімального) елемента. Фактично алгоритм пошуку є найважливішою частиною сортування вибором. Так як основне завдання сортування - упорядкування елементів масиву, необхідно виконувати перестановки. Обмін значень елементів сортованого масиву відбувається в **рядках 48 -50**. Якщо поміняти знак >**в рядку 46** на знак менше, то сортуватися масив буде по спадаючій. Результат роботи програми показаний на малюнку 1.