Сортировка пузырьком*.*

Сортировка пузырьком – простейший алгоритм сортировки, применяемый чисто для учебных целей. Практического применения этому алгоритму нет, так как он не эффективен, особенно если необходимо отсортировать массив большого размера. К плюсам сортировки пузырьком относится простота реализации алгоритма.

Алгоритм сортировки пузырьком сводится к повторению проходов по элементам сортируемого массива. Проход по элементам массива выполняет внутренний цикл. За каждый проход сравниваются два соседних элемента, и если порядок неверный элементы меняются местами. Внешний цикл будет работать до тех пор, пока массив не будет отсортирован. Таким образом внешний цикл контролирует количество срабатываний внутреннего цикла Когда при очередном проходе по элементам массива не будет совершено ни одной перестановки, то массив будет считаться отсортированным. Чтобы хорошо понять алгоритм, отсортируем методом пузырька массив, к примеру, из 7 чисел (см. Таблица 1).  
**исходный массив: 3 3 7 1 2 5 0**

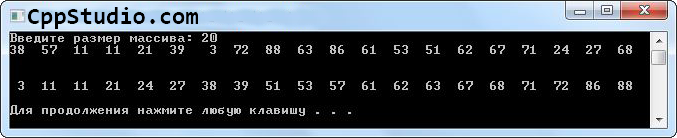
| Таблица 1 — Сортировка пузырьком | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№ итерации** | **Элементы массива** | | | | | | | **Перестановки** |
| исх. массив | 3 | 3 | 7 | 1 | 2 | 5 | 0 |  |
| 0 | 3 | 3 |  |  |  |  |  | false |
| 1 |  | 3 | 7 |  |  |  |  | false |
| 2 |  |  | **1** | **7** |  |  |  | 7>1, true |
| 3 |  |  |  | **2** | **7** |  |  | 7>2, true |
| 4 |  |  |  |  | **5** | **7** |  | 7>5, true |
| 5 |  |  |  |  |  | **0** | **7** | 7>0, true |
| тек. массив | **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 |
| тек. массив | **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 |
| тек. массив | **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 |
| тек. массив | **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 |
| конечный массив | **0** | **1** | **2** | **3** | **3** | **5** | **7** |  |
| Конец сортировки | | | | | | | | |

Для того чтобы отсортировать массив хватило пяти запусков внутреннего цикла, for. Запустившись, цикл for срабатывал 6 раз, так как элементов в массиве 7, то итераций (повторений) цикла for должно быть на одно меньше. На каждой итерации сравниваются два соседних элемента массива. Если текущий элемент массива больше следующего, то меняем их местами. Таким образом, пока массив не будет отсортирован, будет запускаться внутренний цикл и выполняться операция сравнения. Обратите внимание на то, что за каждое полное выполнение цикла for как минимум один элемент массива находит своё место. В худшем случае, понадобится n-2 запуска внутреннего цикла, где n – количество элементов массива. Это говорит о том, что сортировка пузырьком крайне не эффективна для больших массивов.

Разработаем программу, в которой сначала необходимо ввести размер одномерного массива, после чего массив заполняется случайными числами и сортируется методом пузырька.

|  |  |
| --- | --- |
| 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: определяет точку входа для консольного приложения.    #include "stdafx.h"  #include <iostream>  #include <iomanip>  #include <ctime>  using namespace std;    void bubbleSort(int \*, int); // прототип функции сортировки пузырьком    int main(int argc, char\* argv[])  {      srand(time(NULL));      setlocale(LC\_ALL, "rus");      cout << "Введите размер массива: ";      int size\_array; // длинна массива      cin >> size\_array;        int \*sorted\_array = new int [size\_array]; // одномерный динамический массив      for (int counter = 0; counter < size\_array; counter++)      {          sorted\_array[counter] = rand() % 100; // заполняем массив случайными числами          cout << setw(2) << sorted\_array[counter] << "  "; // вывод массива на экран      }      cout << "\n\n";        bubbleSort(sorted\_array, size\_array); // вызов функции сортировки пузырьком        for (int counter = 0; counter < size\_array; counter++)      {          cout << setw(2) << sorted\_array[counter] << "  "; // печать отсортированного массива      }      cout << "\n";        system("pause");      return 0;  }    void bubbleSort(int\* arrayPtr, int length\_array) // сортировка пузырьком  {   int temp = 0; // временная переменная для хранения элемента массива   bool exit = false; // болевая переменная для выхода из цикла, если массив отсортирован     while (!exit) // пока массив не отсортирован   {    exit = true;    for (int int\_counter = 0; int\_counter < (length\_array - 1); int\_counter++) // внутренний цикл      //сортировка пузырьком по возрастанию - знак >      //сортировка пузырьком по убыванию - знак <      if (arrayPtr[int\_counter] > arrayPtr[int\_counter + 1]) // сравниваем два соседних элемента      {       // выполняем перестановку элементов массива       temp = arrayPtr[int\_counter];       arrayPtr[int\_counter] = arrayPtr[int\_counter + 1];       arrayPtr[int\_counter + 1] = temp;       exit = false; // на очередной итерации была произведена перестановка элементов      }   } |

Результат работы программы показан на рисунке 1.



Сортировка вставками

Сортировка вставками — достаточно простой алгоритм. Как в и любом другом алгоритме сортировки, с увеличением размера сортируемого массива увеличивается и время сортировки. Основным преимуществом алгоритма сортировки вставками является возможность сортировать массив по мере его получения.То есть имея часть массива, можно начинать его сортировать. В параллельном программирование такая особенность играет не маловажную роль.

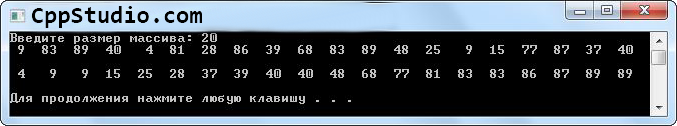
Сортируемый массив можно разделить на две части — отсортированная часть и неотсортированная. В начале сортировки первый элемент массива считается отсортированным, все остальные — не отсортированные. Начиная со второго элемента массива и заканчивая последним, алгоритм вставляет неотсортированный элемент массива в нужную позицию в отсортированной части массива. Таким образом, за один шаг сортировки отсортированная часть массива увеличивается на один элемент, а неотсортированная часть массива уменьшается на один элемент. Рассмотрим пример сортировки по возрастанию массива из 7 чисел (см. Таблица 1):  
исходный массив: 3 3 7 1 2 5 0

| Таблица 1 — Сортировка вставками | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **шаг** | **отсортированная часть массива** | | | | | | | **тек. элемент** | **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 каждом шаге сортировки сравнивается текущий элемент со всеми элементами в отсортированной части. На первом шаге сравнивается тройка с тройкой, они равны поэтому не меняем их местами. На втором шаге сравниваем 7 с двумя тройками, 7 > 3 а так как сортировка по возрастанию, то опять элементы массива остаются на своих местах. На третьем шаге единица сравнивается с тремя элементами и все они больше единицы, значит единицу вставляем на первое место, в начало массива. На четвёртом шаге текущий элемент — 2 сравниваем с элементами 1, 3, 3, 7. Получается, что  1 < 2 < 3 и 7 поэтому двойку вставляем между единицей и тройкой. Пятый и шестой шаги выполняются точно также. В итоге на шестом шагу мы получаем отсортированный по возрастанию массив. Запрограммируем алгоритм сортировки 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: определяет точку входа для консольного приложения.    #include "stdafx.h"  #include <iostream>  #include <ctime>  #include <iomanip>  using namespace std;    void insertionSort(int \*, int); // прототип функции сортировки вставками    int main(int argc, char\* argv[])  {      srand(time(NULL));      setlocale(LC\_ALL, "rus");      cout << "Введите размер массива: ";      int size\_array; // длинна массива      cin >> size\_array;        int \*sorted\_array = new int [size\_array]; // одномерный динамический массив      for (int counter = 0; counter < size\_array; counter++)      {          sorted\_array[counter] = rand() % 100; // заполняем массив случайными числами          cout << setw(2) << sorted\_array[counter] << "  "; // вывод массива на экран      }      cout << "n";        insertionSort(sorted\_array, size\_array); // вызов функции сортировки вставками        for (int counter = 0; counter < size\_array; counter++)      {          cout << setw(2) << sorted\_array[counter] << "  "; // печать отсортированного массива      }      cout << "n";      delete [] sorted\_array; // высвобождаем память      system("pause");      return 0;  }    void insertionSort(int \*arrayPtr, int length) // сортировка вставками  {      int temp, // временная переменная для хранения значения элемента сортируемого массива          item; // индекс предыдущего элемента      for (int counter = 1; counter < length; counter++)      {          temp = arrayPtr[counter]; // инициализируем временную переменную текущим значением элемента массива          item = counter-1; // запоминаем индекс предыдущего элемента массива          while(item >= 0 && arrayPtr[item] > temp) // пока индекс не равен 0 и предыдущий элемент массива больше текущего          {              arrayPtr[item + 1] = arrayPtr[item]; // перестановка элементов массива              arrayPtr[item] = temp;              item--;          }      }  } |

Программа сортирует одномерный массив по возрастанию. Изменив знак > в **строке 47** массив будет сортироваться по убыванию. Результат работы программы показан на рисунке 1.



Сортировка выбором

Пожалуй, самый простой алгоритм сортировок – это сортировка выбором. Судя по названию сортировки, необходимо что-то выбирать (максимальный или минимальный элементы массива). Алгоритм сортировки выбором находит в исходном массиве максимальный или минимальный элементы, в зависимости от того как необходимо сортировать массив, по возрастанию или по убыванию. Если массив должен быть отсортирован по возрастанию, то из исходного массива необходимо выбирать минимальные элементы. Если же массив необходимо отсортировать по убыванию, то выбирать следует максимальные элементы.

Допустим необходимо отсортировать массив по возрастанию. В исходном массиве находим минимальный элемент, меняем его местами с первым элементом массива. Уже, из всех элементов массива один элемент стоит на своём месте. Теперь будем рассматривать не отсортированную часть массива, то есть все элементы массива, кроме первого. В неотсортированной части массива опять ищем минимальный элемент. Найденный минимальный элемент меняем местами со вторым элементом массива и т. д. Таким образом, суть алгоритма сортировки выбором сводится к многократному поиску минимального (максимального) элементов в неотсортированной части массива. Отсортируем массив из семи чисел согласно алгоритму «Сортировка выбором».

**исходный массив: 3 3 7 1 2 5 0**  
1)Итак, находим минимальный элемент в массиве. 0 – минимальный элемент  
2)Меняем местами минимальный и первый элементы массива.  
Текущий массив: 0 3 7 1 2 5 3  
3) Находим минимальный элемент в неотсортированной части массива. 1 – минимальный элемент  
4) Меняем местами минимальный и первый элементы массива.  
Текущий массив: 0 1 7 3 2 5 3  
5) min = 2  
6) Текущий массив: 0 1 2 3 7 5 3  
7)min = 3  
8) Текущий массив:  0 1 2 3 7 5 3 в массиве ничего не поменялось, так как 3 стоит на своём месте  
9) min = 3  
10) Конечный вид массива:  0 1 2 3 3 5 7 – массив отсортирован

Запрограммируем алгоритм сортировки выбором 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: определяет точку входа для консольного приложения.    #include "stdafx.h"  #include <iostream>  #include <ctime>  #include <iomanip>  using namespace std;    void choicesSort(int\*, int); // прототип функции сортировки выбором    int main(int argc, char\* argv[])  {      srand(time(NULL));      setlocale(LC\_ALL, "rus");      cout << "Введите размер массива: ";      int size\_array; // длинна массива      cin >> size\_array;        int \*sorted\_array = new int [size\_array]; // одномерный динамический массив      for (int counter = 0; counter < size\_array; counter++)      {          sorted\_array[counter] = rand() % 100; // заполняем массив случайными числами          cout << setw(2) << sorted\_array[counter] << "  "; // вывод массива на экран      }      cout << "\n\n";        choicesSort(sorted\_array, size\_array); // вызов функции сортировки выбором        for (int counter = 0; counter < size\_array; counter++)      {          cout << setw(2) << sorted\_array[counter] << "  "; // печать отсортированного массива      }      cout << "\n";      delete [] sorted\_array; // высвобождаем память      system("pause");      return 0;  }    void choicesSort(int\* arrayPtr, int length\_array) // сортировка выбором  {      for (int repeat\_counter = 0; repeat\_counter < length\_array; repeat\_counter++)      {          int temp = arrayPtr[0]; // временная переменная для хранения значения 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 сортировки выбором основан на алгоритме поиска максимального (минимального) элемента. Фактически алгоритм поиска является важнейшей частью сортировки выбором. Так как основная задача сортировки — упорядочивание элементов массива, необходимо выполнять перестановки. Обмен значений элементов сортируемого массива происходит в **строках 48**—**50**. Если поменять знак >в **строке 46**на знак меньше, то сортироваться массив будет по убыванию. Результат работы программы показан на рисунке 1.