# Лабораторна робота № 11. Алгоритми пошуку

# з використанням таблиць

***Мета:***  закріпити теоретичні знання та набути практичний досвід впорядкування набору статичних та динамічних структур даних.

**1 Вимоги**

**1.1 Розробник**

* Макаренко Владислав Олександрович
* Студент 1-го курсу
* Групи КІТ-120а

**1.2 Загальне завдання**

Написати програму, що реалізує сортування статичного та/або динамічного набору даних заданим способом згідно даних табл.

**1.2 Загальне завдання**

Модифіковане бульбашкове сортування

**2 Описи програм**

**Код програми**

#include <iostream>  
  
using namespace std;  
  
void ShowArray(int *arr*[], int *size*); // показ массива  
  
void BubbleSort(int *a*[], int *size*); //сортировка масива  
  
void ReverseSort(int *a*[], int *size*);  
  
int main()  
{  
 setlocale(LC\_ALL, "rus");  
 srand(time(NULL));  
 int size = 0;  
 cout << "Размер массива: ";  
 cin >> size;  
 int \*A = new int[size];  
 int B[size];  
  
 cout << "Массив случайных чисел: \n";  
 for (int i = 0; i < size; i++)  
 {  
 A[i] = 1 + rand() % 444;  
 B[i] = A[i];  
 }  
 ShowArray(A, size);  
  
 clock\_t Start1 = clock();  
 BubbleSort(A, size);  
 clock\_t End1= clock();  
  
 clock\_t Start4 = clock();  
 BubbleSort(B, size);  
 clock\_t End4= clock();  
  
 cout << "Массив отсортированый пузырьком: \n";  
 ShowArray(A, size);  
  
 clock\_t Start3 = clock();  
 BubbleSort(A, size);  
 clock\_t End3= clock();  
  
 clock\_t Start6 = clock();  
 BubbleSort(B, size);  
 clock\_t End6= clock();  
  
 ReverseSort(A, size);  
 ReverseSort(B, size);  
  
 clock\_t Start2 = clock();  
 BubbleSort(A, size);  
 clock\_t End2= clock();  
  
 clock\_t Start5 = clock();  
 BubbleSort(B, size);  
 clock\_t End5= clock();  
  
 double Res1 = (double)(End1 - Start1)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки динамического массива пузырьком(случайный набор данных): %.10f сек\n\n", Res1);  
  
 double Res2 = (double)(End2 - Start2)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки динамического массива пузырьком(обратно отсортированный набор данных): %.10f сек\n\n", Res2);  
  
 double Res3 = (double)(End3 - Start3)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки динамического массива пузырьком(отсортированный набор данных): %.10f сек\n\n\n", Res3);  
  
 double Res4 = (double)(End4 - Start4)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки статического массива пузырьком(случайный набор данных): %.10f сек\n\n", Res4);  
  
 double Res5 = (double)(End5 - Start5)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки статического массива пузырьком(обратно отсортированный набор данных): %.10f сек\n\n", Res5);  
  
 double Res6 = (double)(End6 - Start6)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки статического массива пузырьком(отсортированный набор данных): %.10f сек\n\n", Res6);  
  
 return 0;  
}  
  
//вывод массива на экран  
void ShowArray(int *arr*[], int *size*)  
{  
 for (int i = 0; i < *size*; i++)  
 {  
 cout.width(5);  
 cout << arr[i];  
 if ((i + 1) % 10 == 0)  
 {  
 cout << endl;  
 }  
 }  
 cout << endl << endl;  
}  
  
void BubbleSort(int *a*[], int *size*)  
{  
 int counter = 0;  
 for (int i=0; i<*size*; i++){  
 for (int j=0; j<*size*-1-i; j++){  
 counter++;  
 if (*a*[j]>*a*[j+1]){  
 swap(*a*[j],*a*[j+1]);  
 }  
 }  
 }  
 cout << "Количество сравнений: " << counter << endl;  
}  
  
void ReverseSort(int *a*[], int *size*)  
{  
 int counter = 0;  
 for (int i=0; i<*size*; i++){  
 for (int j=0; j<*size*-1-i; j++){  
 counter++;  
 if (*a*[j] < *a*[j + 1]){  
 swap(*a*[j],*a*[j + 1]);  
 }  
 }  
 }  
}

**Результати виконання програми**

За алгоритмом коду демонструємо роботу програми для 20 елементів (див. рис. 1).

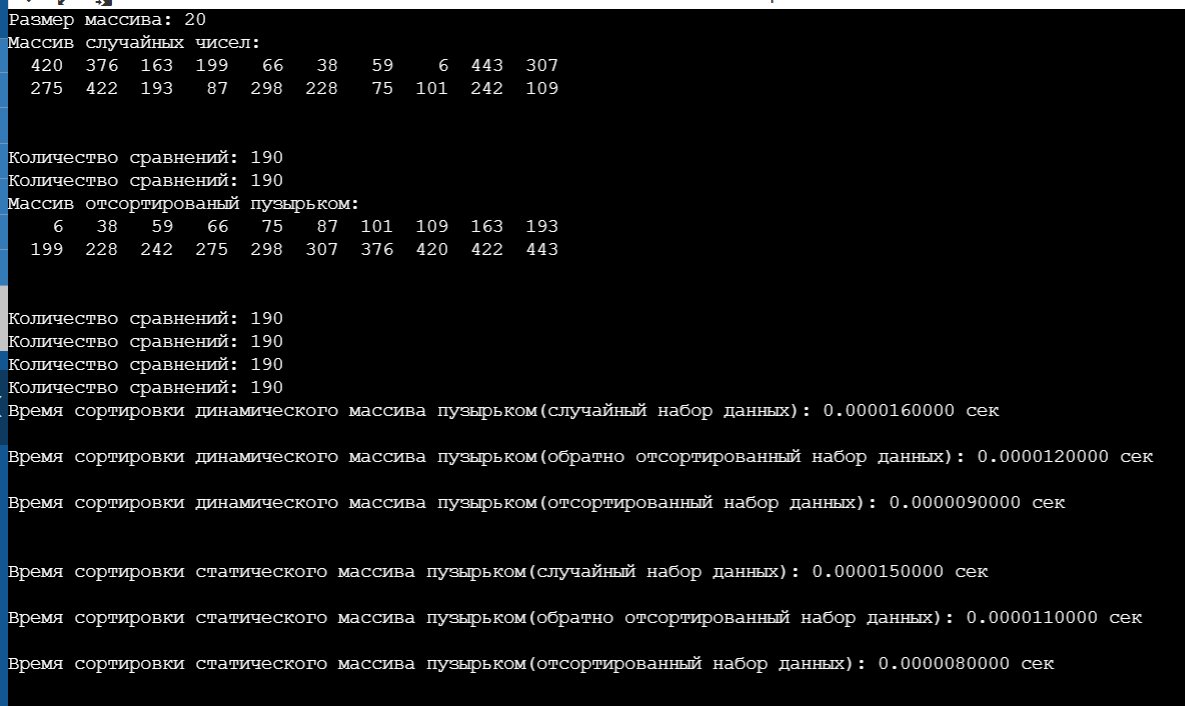


Рисунок 1 – Результати роботи першого кроку

Таблиця 1 – Результати тестування алгоритму сортування на основі динамічного масиву

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Відсортований набір даних* | 20 | 1000 | 5000 | 10000 | 50000 |
| Кількість порівнянь | 190 | 499500 | 12497500 | 49995000 | 1249975000 |
| Кількість пересилань | 0 | 0 | 0 | 0 | 0 |
| Час сортування | 0.000009 | 0.002177 | 0.039347 | 0.156744 | 3.92889 |
| *Відсортований в зворотньому порядку* | 20 | 1000 | 5000 | 10000 | 50000 |
| Кількість порівнянь | 190 | 499500 | 12497500 | 49995000 | 1249975000 |
| Кількість пересилань | 190 | 498376 | 12469331 | 49881809 | 1247162055 |
| Час сортування | 0.000012 | 0.009797 | 0.173589 | 0.682226 | 17.500296 |
| *Випадковий набір даних* | 20 | 1000 | 5000 | 10000 | 50000 |
| Кількість порівнянь | 190 | 499500 | 12497500 | 49995000 | 1249975000 |
| Кількість пересилань | 128 | 252110 | 6249327 | 24941742 | 624619551 |
| Час сортування | 0.000016 | 0.006554 | 0.142966 | 0.599905 | 15.374894 |

Таблиця 2 – Результати тестування алгоритму сортування на основі статичного масиву

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Відсортований набір даних* | 20 | 1000 | 5000 | 10000 | 50000 |
| Кількість порівнянь | 190 | 499500 | 12497500 | 49995000 | 1249975000 |
| Кількість пересилань | 0 | 0 | 0 | 0 | 0 |
| Час сортування | 0.000008 | 0.001639 | 0.050826 | 0.169228 | 3.933395 |
| *Відсортований в зворотньому порядку* | 20 | 1000 | 5000 | 10000 | 50000 |
| Кількість порівнянь | 190 | 499500 | 12497500 | 49995000 | 1249975000 |
| Кількість пересилань | 190 | 498376 | 12469331 | 49881809 | 1247162055 |
| Час сортування | 0.000011 | 0.007033 | 0.183533 | 0.897847 | 17.088548 |
| *Випадковий набір даних* | 20 | 1000 | 5000 | 10000 | 50000 |
| Кількість порівнянь | 190 | 499500 | 12497500 | 49995000 | 1249975000 |
| Кількість пересилань | 128 | 252110 | 6249327 | 24941742 | 624619551 |
| Час сортування | 0.000015 | 0.005731 | 0.173863 | 0.701638 | 15.064323 |

**Висновок:** на цій лабораторній роботі ми закріпили теоретичні знання та набули практичний досвід впорядкування набору статичних та динамічних структур даних.