# Лабораторна робота № 12. Алгоритми сортування розподілом та злиттям

***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**

* 1. Пірамідальне сортування
  2. Поразрядне цифрове сортування
  3. Сортування попарним злиттям

**2 Описи програм**

**2.1 Пірамідальне сортування**

**Код програми**

#include <iostream>  
  
#define P 10  
#define D 3 // Не более з-х разрядов в числах набора  
  
using namespace std;  
  
void ShowArray(int *arr*[], int *size*); // показ массива  
  
void HeapSort(int *a*[], int *size*);  
  
void downHeap(int *a*[], int *k*, int *n*);  
  
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];  
  
 cout << "Массив случайных чисел: \n";  
 for (int i = 0; i < size; i++)  
 {  
 A[i] = 1 + rand() % 444;  
 }  
 ShowArray(A, size);  
  
 clock\_t Start1 = clock();  
 HeapSort(A, size);  
 clock\_t End1= clock();  
  
 cout << "Массив отсортированый: \n";  
 ShowArray(A, size);  
  
 clock\_t Start3 = clock();  
 HeapSort(A, size);  
 clock\_t End3= clock();  
  
 ReverseSort(A, size);  
  
 clock\_t Start2 = clock();  
 HeapSort(A, size);  
 clock\_t End2= clock();  
  
 double Res1 = (double)(End1 - Start1)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки (случайный набор данных): %.10f сек\n\n", Res1);  
  
 double Res2 = (double)(End2 - Start2)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки (обратно отсортированный набор данных): %.10f сек\n\n", Res2);  
  
 double Res3 = (double)(End3 - Start3)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки (отсортированный набор данных): %.10f сек\n\n", Res3);  
  
 delete [] A;  
 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;  
}  
  
  
// Процедура просеивания следующего элемента  
void downHeap(int *a*[], int *k*, int *n*){  
 // До процедуры: a[k+1]...a[n] - пирамида  
 // После: a[k]...a[n] - пирамида  
 int new\_elem;  
 int child;  
 new\_elem = *a*[*k*];  
  
 while(*k* <= *n*/2) // пока у a[k] есть дети  
 { child = 2\**k*;  
 // выбираем большего сына  
 if( child < *n* && *a*[child] < *a*[child+1] )  
 child++;  
 if( new\_elem >= *a*[child] ) break;  
 // иначе  
 *a*[*k*] = *a*[child]; // переносим сына наверх  
 *k* = child;  
 }  
 *a*[*k*] = new\_elem;  
}  
  
// Пирамидальная сортировка  
void HeapSort(int *a*[], int *size*){  
 int i;  
 // строим пирамиду  
 for(i=*size*/2-1; i >= 0; i--) {  
 downHeap(*a*, i, *size*-1);  
 }  
 // теперь a[0]...a[size-1] пирамида  
 for(i=*size*-1; i > 0; i--)  
 { // меняем первый с последним  
 swap(*a*[i], *a*[0]);  
 // восстанавливаем пирамидальность a[0]...a[i-1]  
 downHeap(*a*, 0, i-1);  
 }  
}  
  
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 100 елементів (див. рис. 1).

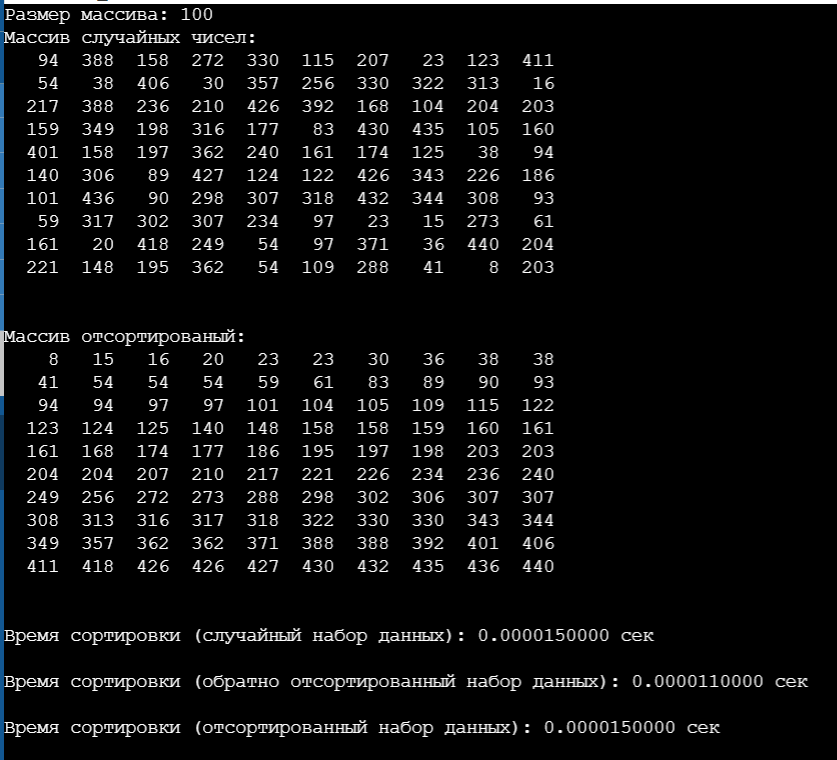


Рисунок 1 – Результати роботи першого кроку

**2.2 Порозрядне цифрове сортування**

**Код програми**

#include <iostream>  
  
#define P 10  
#define D 3 // Не более з-х разрядов в числах набора  
  
using namespace std;  
  
void ShowArray(int *arr*[], int *size*); // показ массива  
  
// Поразрядная сортировка  
void RadixSort( int *a*[], int *size*);  
  
// Возвращает n-ю цифру считая с нуля в числе v  
int digit(int *v*, int *r*);  
  
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];  
  
 cout << "Массив случайных чисел: \n";  
 for (int i = 0; i < size; i++)  
 {  
 A[i] = 1 + rand() % 444;  
 }  
 ShowArray(A, size);  
  
 clock\_t Start1 = clock();  
 RadixSort(A, size);  
 clock\_t End1= clock();  
  
 cout << "Массив отсортированый: \n";  
 ShowArray(A, size);  
  
 clock\_t Start3 = clock();  
 RadixSort(A, size);  
 clock\_t End3= clock();  
  
 ReverseSort(A, size);  
  
 clock\_t Start2 = clock();  
 RadixSort(A, size);  
 clock\_t End2= clock();  
  
 double Res1 = (double)(End1 - Start1)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки (случайный набор данных): %.10f сек\n\n", Res1);  
  
 double Res2 = (double)(End2 - Start2)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки (обратно отсортированный набор данных): %.10f сек\n\n", Res2);  
  
 double Res3 = (double)(End3 - Start3)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки (отсортированный набор данных): %.10f сек\n\n", Res3);  
  
 delete [] A;  
 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;  
}  
  
  
// Возвращает n-ю цифру считая с нуля в числе v  
int digit(int *v*, int *r*)  
{  
 for (; *r* > 0; *r*--) {  
 *v* /= P;  
 }  
 return *v* % P;  
}  
  
// Поразрядная сортировка  
void RadixSort( int *a*[], int *size*)  
{  
 int b[P]; // Индекс элемента, расположенного за последним в i-й группе  
 int i, j, k, m, x;  
  
 for (m=0; m < D; m++) // Перебираем все числа, начиная с младшего разряда  
 {  
 for (i=0; i < P; i++) b[i]=0; // Обнуляем индексы  
 for (i=0; i < *size*; i++){ // Проходим массив  
 k=digit(*a*[i],m); // Определяем m-ю цифру  
 x=*a*[i]; // Сохраняем элемент  
 for (j=i; j > b[k]; j--) { // И затираем его сдвигая массив вправо  
 *a*[j] = *a*[j - 1];  
 }  
 *a*[b[k]]=x; // Записываем его в конец k-й группы  
 for (j=k; j < P; j++) { // Модифицируем все индексы не меньшие k  
 b[j] = b[j] + 1;  
 }  
 }  
 }  
}  
  
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 100 елементів (див. рис. 2).

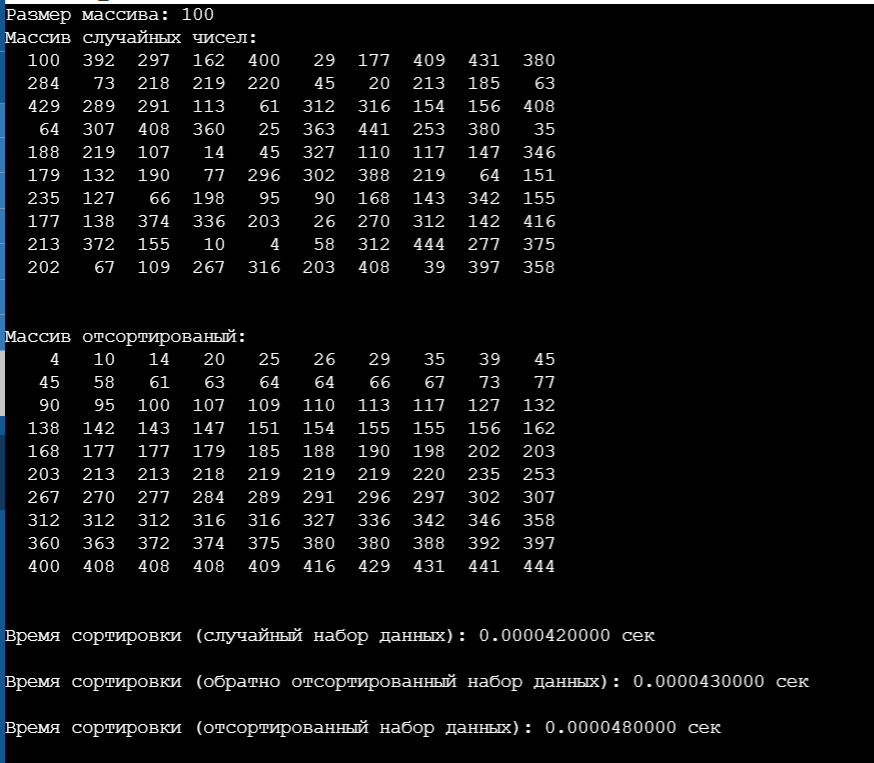


Рисунок 2 – Результати роботи першого кроку

**2.3 Сортування попарним злиттям**

**Код програми**

#include <iostream>  
  
#define P 10  
#define D 3 // Не более з-х разрядов в числах набора  
  
using namespace std;  
  
void ShowArray(int *arr*[], int *size*); // показ массива  
  
// Поразрядная сортировка  
void PairMergeSort(int *a*[], int *size*);  
  
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];  
  
 cout << "Массив случайных чисел: \n";  
 for (int i = 0; i < size; i++)  
 {  
 A[i] = 1 + rand() % 444;  
 }  
 ShowArray(A, size);  
  
 clock\_t Start1 = clock();  
 PairMergeSort(A, size);  
 clock\_t End1= clock();  
  
 cout << "Массив отсортированый: \n";  
 ShowArray(A, size);  
  
 clock\_t Start3 = clock();  
 PairMergeSort(A, size);  
 clock\_t End3= clock();  
  
 ReverseSort(A, size);  
  
 clock\_t Start2 = clock();  
 PairMergeSort(A, size);  
 clock\_t End2= clock();  
  
 double Res1 = (double)(End1 - Start1)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки (случайный набор данных): %.10f сек\n\n", Res1);  
  
 double Res2 = (double)(End2 - Start2)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки (обратно отсортированный набор данных): %.10f сек\n\n", Res2);  
  
 double Res3 = (double)(End3 - Start3)/ CLOCKS\_PER\_SEC;  
 printf("Время сортировки (отсортированный набор данных): %.10f сек\n\n", Res3);  
  
 delete [] A;  
 return 0;  
}  
  
void PairMergeSort(int *a*[], int *size*)  
{ int i0,j0,i,j,si,sj,k,ke,t,m ;  
 si=1; // начальный размер одного множества  
 while (si<*size*) //цикл пока одно множество не составит весь массив}  
 { i0 =0; // нач. индекс 1-го множества пары  
 while (i0<*size*) // цикл пока не пересмотрим весь массив  
 { j0 = i0+si; // нач. индекс 2-го множества пары  
 i=i0; j=j0;  
//размер 2-го множества пары может ограничиваться концом массива  
 if (si>=*size*-j0) sj=*size*-j0;  
 else sj=si;  
 if (sj>0)  
 { k = i0; // нач. индекс слитого множества  
 while ( (i < i0+si+sj) && (j < j0+sj)) // цикл пока не исчерпаются оба входные множества  
 { if (*a*[i]>*a*[j])  
// если эл-т 1-го <= элемента 2-го, он остается на своем месте, но вых.множество расширяется  
// иначе - освобождается место в вых.множестве и туда заносится эл-т из 2-го множества  
 { t=*a*[j];  
 for (m =j-1; m>=k; m--) *a*[m+1] =*a*[m];  
 *a*[k] =t; j++; //к след. эл-ту во 2-м множестве  
 }  
 k++; // вых. множество увеличилось  
 i++; // если был перенос - за счет сдвига, если не было - за счет перехода эл-та в вых.  
 }  
 }  
 i0 += si\*2; // начало следующей пары  
 }  
 si\*=2; // размер эл-тов пары увеличивается вдвое  
 }  
}  
  
//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 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 100 елементів (див. рис. 3).

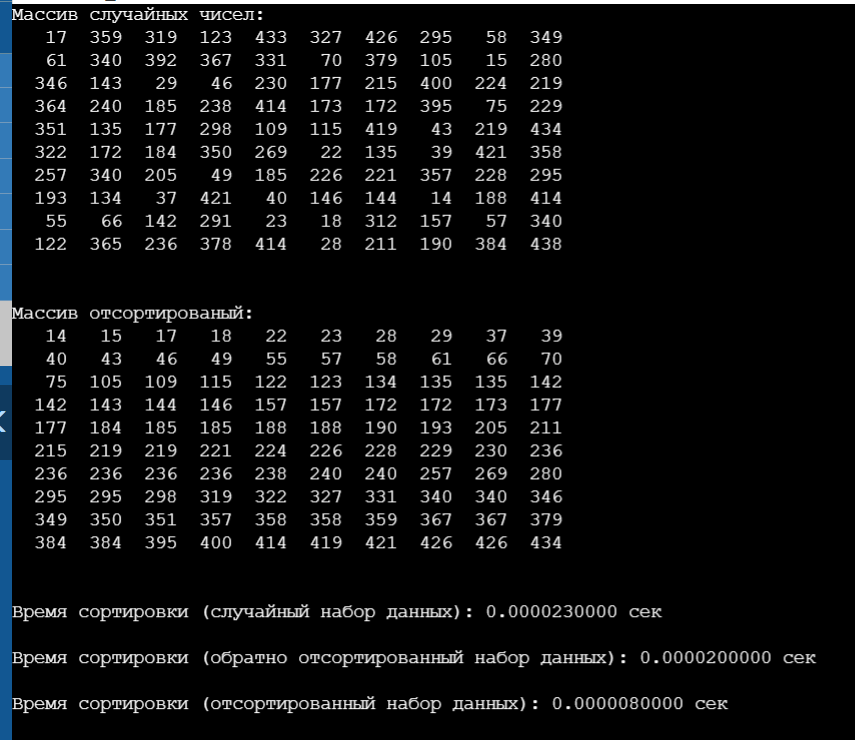


Рисунок 3 – Результати роботи першого кроку

Таблиця 1 – Результати тестування алгоритму сортування на основі динамічного масиву

|  |  |  |  |
| --- | --- | --- | --- |
|  | Пірамідальне сортування | Поразрядне цифрове сортування | Сортування злиттям |
| n = 100 (випадковий набір даних) | 0.000015 | 0.000044 | 0.000023 |
| n = 100 (впорядкований за зростанням) | 0.000014 | 0.000061 | 0.000008 |
| n = 100 (впорядкований за спаданням) | 0.000012 | 0.000044 | 0.0000021 |
| n = 1,000 (випадковий набір даних) | 0.000189 | 0.00283 | 0.001136 |
| n = 1,000 (впорядкований за зростанням) | 0.000163 | 0.002287 | 0.000079 |
| n = 1,000 (впорядкований за спаданням) | 0.00015 | 0.002092 | 0.001298 |
| n = 10,000 (випадковий набір даних) | 0.001888 | 0.207634 | 0.063159 |
| n = 10,000 (впорядкований за зростанням) | 0.001567 | 0.199353 | 0.000932 |
| n = 10,000 (впорядкований за спаданням) | 0.00146 | 0.200847 | 0.092918 |

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