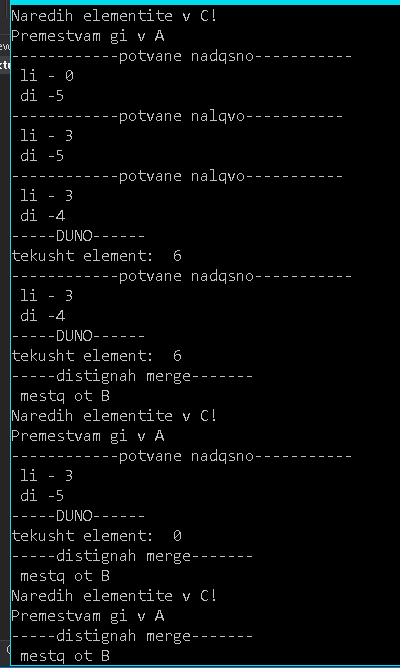
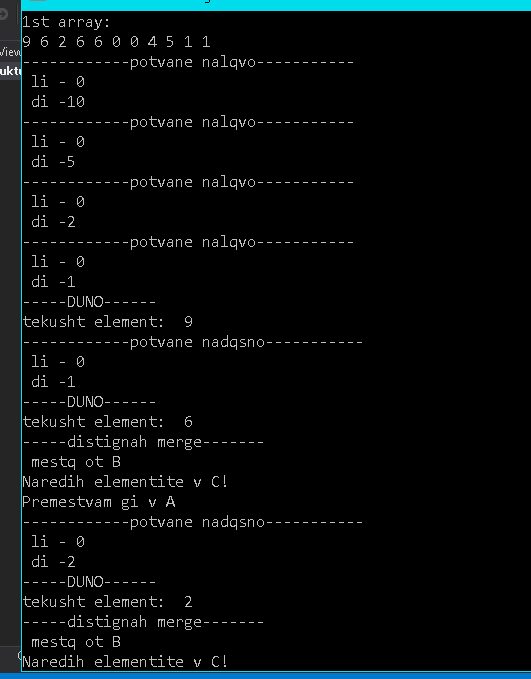
Домашно към тема 5. Сортиране чрез Сливане.

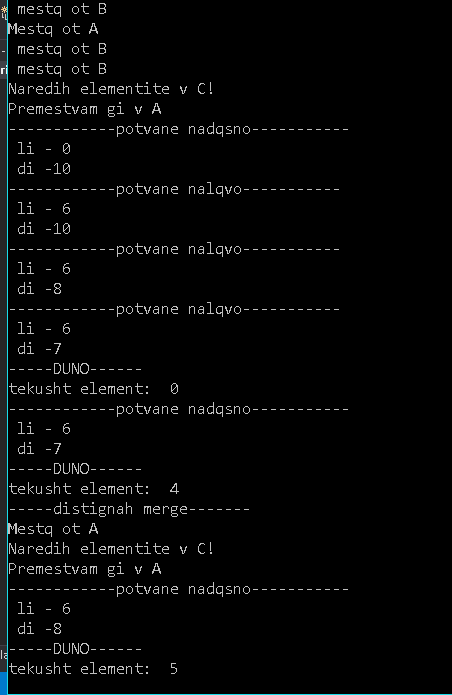
1. Съставете рекурсивната програма за Сортиране чрез сливане.

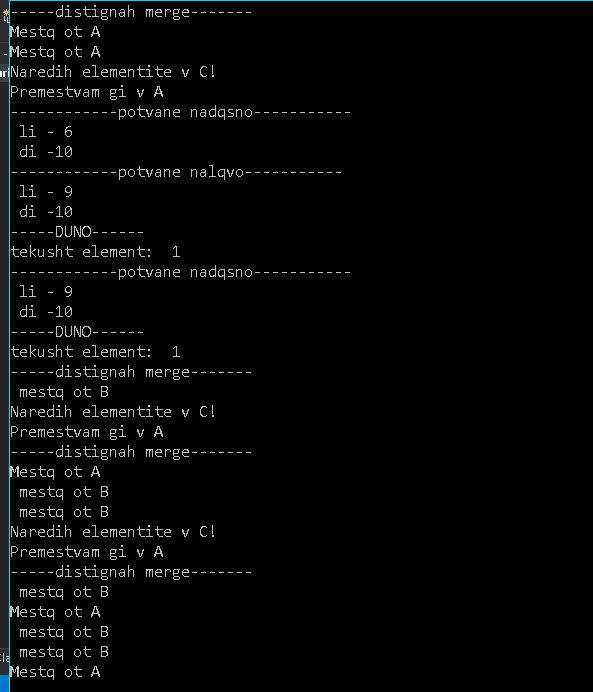
На изхода от програмата да се разпачатват - на потъване коментарите “потъвам наляво – граници ляв ХХ и десен УУ” и “потъвам надясно - граници ляв ХХ и десен УУ”. Сливането да разпечатва “Сливам в С”и “Прехварлям в А”. На изплуване да се разпечатва „слях ги и са наредени от индекс // до индекс VV“.

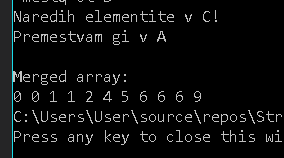
1. Така съставената програма, изчистена от коментарите, пуснете за сравнение със Сортиране по дялове, при еднакви за двата алгоритъма поредици от входни масиви от случайни стойности (нарастващи постъпково по размер) и, по известната методика, съставете графиката и ги съпоставете.

Заданието е неразделна част от домашното.









Код:

#include <iostream>

using namespace std;

int arrA[11] = { 9,6,2,6,6,0,0,4,5,1,1 };

int arrC[11];

void merge(int li,int di) {

int aBeg = li, bBeg = ((li + di) / 2) + 1, cPlace = li;

while (aBeg < ((li + di) / 2) + 1 && bBeg < di + 1)

{

if (arrA[aBeg] < arrA[bBeg])

{

cout << "Mestq ot A"<<endl;

arrC[cPlace] = arrA[aBeg];

aBeg++;

}

else

{

cout << " mestq ot B" << endl;

arrC[cPlace] = arrA[bBeg];

bBeg++;

}

cPlace++;

}

for (int i = aBeg; i < ((li + di) / 2) + 1; i++)

{

arrC[cPlace] = arrA[i];

cPlace++;

}

for (int i = bBeg; i < di + 1; i++)

{

arrC[cPlace] = arrA[i];

cPlace++;

}

cout << "Naredih elementite v C!" << endl;

cout << "Premestvam gi v A" << endl;

for (int i = li; i < di+1; i++) {

arrA[i] = arrC[i];

}

}

void mergeSort(int li, int di) {

if (li < di) {

cout << "------------potvane nalqvo-----------" << endl;

cout << " li - " << li << endl << " di -" << di << endl;

mergeSort(li, (li + di) / 2);

cout << "------------potvane nadqsno-----------" << endl;

cout << " li - " << li << endl << " di -" << di << endl;

mergeSort(((li + di) / 2) + 1, di);

cout << "-----distignah merge-------" << endl;

merge(li, di);

}

else

{

cout << "-----DUNO------" << endl;

cout << "tekusht element: " << arrA[li]<<endl;

}

}

int main() {

cout << "1st array: " << endl;

for (int i = 0; i <11; i++)

{

cout << arrA[i] << " ";

}

cout << endl;

mergeSort(0, 10);

cout << endl << "Merged array: " << endl;

for (int i = 0; i < 11; i++)

{

cout << arrC[i] << " ";

}

return 0;

}

Зад 2,

#include<iostream>

#include <ctime>

using namespace std;

int n = 10000;

int\* arrA= new int[n];

int\* arrC=new int[n];

int\* arr = new int[n];

void merge(int li, int di) {

int aBeg = li, bBeg = ((li + di) / 2) + 1, cPlace = li;

while (aBeg < ((li + di) / 2) + 1 && bBeg < di + 1)

{

if (arrA[aBeg] < arrA[bBeg])

{

arrC[cPlace] = arrA[aBeg];

aBeg++;

}

else

{

arrC[cPlace] = arrA[bBeg];

bBeg++;

}

cPlace++;

}

for (int i = aBeg; i < ((li + di) / 2) + 1; i++)

{

arrC[cPlace] = arrA[i];

cPlace++;

}

for (int i = bBeg; i < di + 1; i++)

{

arrC[cPlace] = arrA[i];

cPlace++;

}

for (int i = li; i < di + 1; i++) {

arrA[i] = arrC[i];

}

}

void mergeSort(int li, int di) {

if (li < di) {

mergeSort(li, (li + di) / 2);

mergeSort(((li + di) / 2) + 1, di);

merge(li, di);

}

}

void QuickSort(int li, int di) {

int j = li, k = di;

if (li < di) {

int Etalon = arr[(di + li) / 2];

int temp;

do

{

while (arr[j] < Etalon) j++;

while (arr[k] > Etalon) k--;

if (j <= k)

{

temp = arr[k];

arr[k] = arr[j];

arr[j] = temp;

j++;

k--;

}

} while (j <= k);

QuickSort( li, k);

QuickSort( j, di);

}

}

void test(int num) {

for (int i = 0; i < num; i++)

{

arrA[i] = rand()%100;

}

for (int j = 0; j < num; j++)

{

arr[j] = arrA[j];

}

clock\_t c0, c1, c2, c3;

c0 = clock();

QuickSort(0, num-1);

c1 = clock();

float time1 = (float)(c1 - c0) / CLOCKS\_PER\_SEC;

cout << " elements: " << num << " - time for Q SORT " << (float)time1 << endl;

c2 = clock();

mergeSort(0, num-1);

c3 = clock();

float time2 = (float)(c3 - c2) / CLOCKS\_PER\_SEC;

cout << " elements: " << num << " - time for MERGE SORT " << (float)time2 << endl;

}

int main() {

test(n);

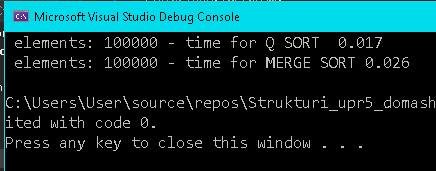
delete[]arrA;

delete[]arr;

delete[]arrC;

return 0;

}



|  |  |  |
| --- | --- | --- |
| ELEMENTI | QUICK SORT | MERGE SORT |
| 10000 | 1 | 2 |
| 20000 | 3 | 6 |
| 30000 | 4 | 9 |
| 40000 | 7 | 13 |
| 50000 | 6 | 16 |
| 60000 | 8 | 18 |
| 70000 | 15 | 20 |
| 80000 | 10 | 24 |
| 90000 | 18 | 28 |
| 100000 | 17 | 26 |