# Додаток А

Код програми

## А.1 – файл ресурсів main.cpp

#include <iostream>

#include <locale>

#include "sorts.h"

using namespace std;

int\* mainArray;

int length;

bool readNumber(int &a)

{

cin >> a;

if (!cin.good())

{

a = 0;

cin.clear();

cin.sync();

return false;

}

return true;

}

bool isArrayInitiallized()

{

return length!=0;

}

void fillArray()

{

if (isArrayInitiallized())

{

delete[] mainArray;

}

cout << "Введите размер массива" << endl;

if (!readNumber(length))

{

cout << "Длина массива должна быть числом" << endl;

return;

}

if (length <= 0)

{

cout << "Длина массива должна быть положительная" << endl;

length = 0;

return;

}

cout << "Введите все элементы массива:" << endl;

mainArray = new int[length];

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

{

if (!readNumber(mainArray[i]))

{

cout << "Элементы масива должны быть числами" << endl;

delete[] mainArray;

length = 0;

return;

}

}

}

int\* copyMainArray()

{

int\* res = new int[length];

memcpy(res, mainArray, length\*sizeof(int));

return res;

}

void printArray(int\* array, int length)

{

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

{

cout << array[i] << " ";

}

cout << endl;

}

void generateRandomMas()

{

if (isArrayInitiallized())

{

delete[] mainArray;

}

cout << "Введите размер массива" << endl;

if (!readNumber(length))

{

cout << "Длина массива должна быть числом" << endl;

return;

}

if (length <= 0)

{

cout << "Длина массива должна быть положительная" << endl;

length = 0;

return;

}

mainArray = new int[length];

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

{

mainArray[i] = rand();

}

cout << "Массив сгенерирован" << endl;

}

void timer(void(\*function) (int\* m, int l))

{

if (!isArrayInitiallized())

{

cout << "Массив пустой" << endl;

return;

}

unsigned int start\_time = clock();

int\* tempArray = copyMainArray();

function(tempArray, length);

//printArray(tempArray, length);

delete[] tempArray;

unsigned int end\_time = clock();

unsigned int search\_time = end\_time - start\_time;

cout << "Время выполнения сортировки: " << search\_time << endl;

}

int main()

{

setlocale(LC\_ALL, "rus");

int command = -1;

length = 0;

while (command != 0)

{

cout << "Доступные команды:" << endl;

cout << "1. Ввести массив для тестирования" << endl;

cout << "2. Сортировка слиянием" << endl;

cout << "3. Быстрая сортировка" << endl;

cout << "4. С помощью кучи" << endl;

cout << "5. Распечатать текущий массив" << endl;

cout << "6. Сгенерировать массив заданной длины" << endl;

cout << "7. Очистить экран" << endl;

cout << "0. Выйти" << endl;

if (!readNumber(command))

{

cout << "Неправильная команда" << endl << endl;

command = -1;

continue;

}

switch (command)

{

case 1:

fillArray();

break;

case 2:

timer(mergeSort);

break;

case 3:

timer(quickSort);

break;

case 4:

timer(heapSort);

break;

case 5:

if (!isArrayInitiallized())

{

cout << "Массив пустой" << endl;

}

else

{

printArray(mainArray, length);

}

break;

case 6:

generateRandomMas();

break;

case 7:

system("cls");

break;

}

cout << "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

}

return 0;

}

## А.2 – файл ресурсів mergeSort.cpp

#include "sorts.h"

void merge(int\* const A, int const lengthA, int\* const B, int const lengthB, int\* C)

{

//Выполнить слияние массива A, содержащего nA элементов,

// и массива B, содержащего nB элементов.

// Результат записать в массив C.

int a = 0, b = 0; //Номера текущих элементов в массивах A и B

while (a + b < lengthA + lengthB) //Пока остались элементы в массивах

{

if ((b >= lengthB) || ((a < lengthA) && (A[a] <= B[b])))

{ //Копирую элемент из массива A

C[a + b] = A[a];

++a;

}

else { //Копирую элемент из массива B

C[a + b] = B[b];

++b;

}

}

}

void mergeSort(int\* array, int length)

{

if (length < 2) return; //Сортировка не нужна

if (length == 2) //Два элемента проще поменять местами,

{ // если нужно, чем делать слияние

if (array[0] > array[1])

{

int temp = array[0];

array[0] = array[1];

array[1] = temp;

}

return;

}

mergeSort(array, length / 2); //Сортируем первую половину

mergeSort(array + length / 2, length - length / 2); //Сортируем вторую половину

int\* tempArr = new int[length]; //Сюда запишем результат слияния

merge(array, length / 2, array + length / 2, length - length / 2, tempArr); //Слияние половин

//Копирование результата слияния в исходный массив:

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

{

array[i] = tempArr[i];

}

delete[] tempArr; //Удаляем временный буфер

}

## А.3 – файл ресурсів quicksort.cpp

// нахождение опорного элемента

int partition(int\* a, int p, int r)

{

int x = \*(a + r);

int i = p - 1;

int j;

int tmp;

for (j = p; j < r; j++)

{

if (\*(a + j) <= x)

{

i++;

tmp = \*(a + i);

\*(a + i) = \*(a + j);

\*(a + j) = tmp;

}

}

tmp = \*(a + r);

\*(a + r) = \*(a + i + 1);

\*(a + i + 1) = tmp;

return i + 1;

}

void quickSort(int\* a, int p, int r)

{

int q;

if (p < r)

{

q = partition(a, p, r);

quickSort(a, p, q - 1);

quickSort(a, q + 1, r);

}

}

void quickSort(int\* a, int length)

{

quickSort(a, 0, length - 1);

}

## А.4 – файл ресурсів heapSort.cpp

#include <iostream>

#include "sorts.h"

using namespace std;

// процедура просеивания следующего элемента

// До процедуры: a[k+1]...a[n] - пирамида

// После: a[k]...a[n] - пирамида

void downHeap(int a[], int k, int n) {

int new\_elem;

int child;

new\_elem = a[k];

while (2\*k + 1 < n) { // пока у a[k] есть дети

child = 2 \* k + 1;

// 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;

}

void heapSort(int\* a, int length)

{

int temp;

// строим пирамиду

for (int i = length / 2 - 1; i >= 0; i--) downHeap(a, i, length - 1);

// теперь a[0]...a[length-1] пирамида

for (int i = length - 1; i > 0; i--) {

// меняем первый с последним

temp = a[i]; a[i] = a[0]; a[0] = temp;

// восстанавливаем пирамидальность a[0]...a[i-1]

downHeap(a, 0, i - 1);

}

}

## A.5 – заголовочний файл sorts.h

void mergeSort(int\* array, int length);

void quickSort(int\* a, int length);

void heapSort(int\* a, int length);