## Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа №15

По дисциплине «Основы алгоритмизации и программирования»

На тему «**Анализ алгоритмов сортировок** »

Выполнила:

Студент 1 курса 8 группы

Рауба Арсений

Преподаватель: асс. Андронова М.В.

Минск, 2024

**Вариант 9:** ввести массив **А**, в массив **В** перенести все элементы массива **А**, имеющие нечетный индекс, справа от которых расположены элементы с нечетным значением. Массив **В** отсортировать по возрастанию, используя алгоритмы сортировок: «пузырек», сортировка Хоара.

#include <iostream>

#include <vector>

#include <algorithm> // Для std::sort

using namespace std;

// Пузырьковая сортировка

void bubbleSort(vector<int>& arr) {

int n = arr.size();

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

swap(arr[j], arr[j + 1]);

}

}

}

}

// Быстрая сортировка (сортировка Хоара)

int partition(vector<int>& arr, int low, int high) {

int pivot = arr[high]; // Опорный элемент

int i = (low - 1); // Индекс меньшего элемента

for (int j = low; j <= high - 1; j++) {

if (arr[j] <= pivot) {

i++; // Увеличиваем индекс меньшего элемента

swap(arr[i], arr[j]);

}

}

swap(arr[i + 1], arr[high]);

return (i + 1);

}

void quickSort(vector<int>& arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main() {

int n;

cout << "Введите размер массива A: ";

cin >> n;

vector<int> A(n);

cout << "Введите элементы массива A: ";

for (int i = 0; i < n; i++) {

cin >> A[i];

}

vector<int> B;

for (int i = 1; i < n; i += 2) {

if (i + 1 < n && A[i + 1] % 2 != 0) {

B.push\_back(A[i]);

}

}

// Копируем массив B для каждой из сортировок

vector<int> B\_bubble = B;

vector<int> B\_quick = B;

// Сортируем пузырьковой сортировкой

bubbleSort(B\_bubble);

cout << "Массив B после пузырьковой сортировки: ";

for (int val : B\_bubble) {

cout << val << " ";

}

cout << endl;

// Сортируем быстрой сортировкой

quickSort(B\_quick, 0, B\_quick.size() - 1);

cout << "Массив B после быстрой сортировки: ";

for (int val : B\_quick) {

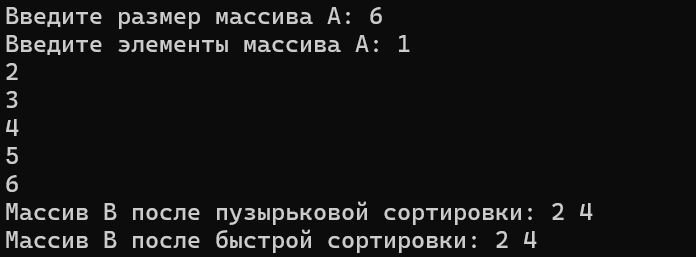
cout << val << " ";

}

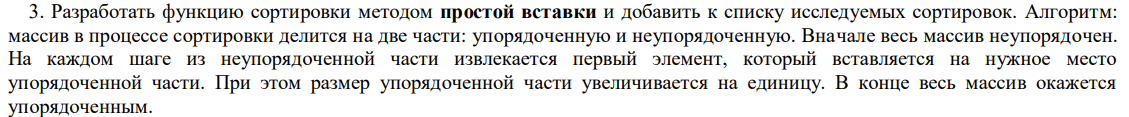
cout << endl;

return 0;

}



**Доп. Задания:**

****

#include <iostream>

#include <ctime>

#include <iomanip>

using namespace std;

// Функция сортировки вставками

void insertSort(int\* C, int N);

int main()

{

setlocale(LC\_CTYPE, "rus"); // Установка локали для вывода на русском языке

int\* A, size; // Объявление указателя и переменной для размера массива

cout << "Введите размер массива: "; // Запрос размера массива у пользователя

cin >> size;

A = new int[size]; // Выделение памяти для массива A

srand(time(0)); // Инициализация генератора случайных чисел

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

{

A[i] = rand() % 99; // Заполнение массива A случайными числами

}

cout << "Массив A: "; // Вывод массива A

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

{

cout << setw(4) << A[i] << " ";

}

insertSort(A, size); // Вызов функции сортировки вставками

delete[] A; // Освобождение памяти

}

void insertSort(int\* C, int N)

{

int buff; // Временная переменная для текущего элемента

int j; // Индекс для внутреннего цикла

// Внешний цикл проходит по каждому элементу начиная со второго

for (int i = 1; i < N; i++)

{

buff = C[i]; // Сохранение текущего элемента

// Внутренний цикл для перемещения элементов массива

for (j = i - 1; j >= 0 && C[j] > buff; j--)

{

C[j + 1] = C[j]; // Сдвиг элемента вправо

}

C[j + 1] = buff; // Вставка текущего элемента на правильное место

}

cout << endl << "Отсортированный массив: "; // Вывод отсортированного массива

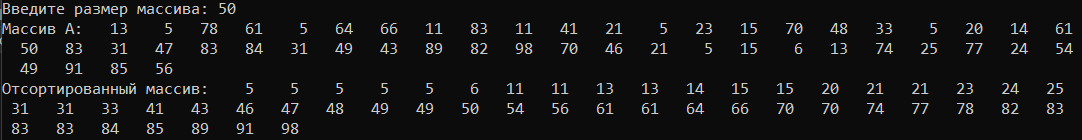
for (int i = 0; i < N; i++)

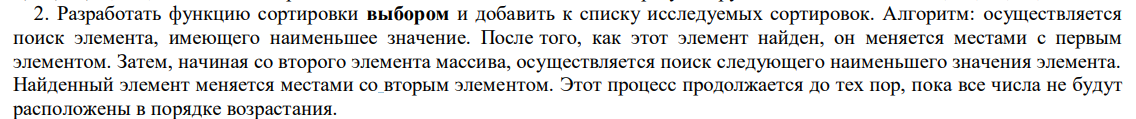
{

cout << setw(4) << C[i] << " "; // Вывод каждого элемента массива

}

}

****

****

#include <iostream>

#include <ctime>

#include <iomanip>

using namespace std;

// Функция сортировки выбором

void choiceSort(int\* D, int N);

int main()

{

setlocale(LC\_CTYPE, "rus"); // Установка локали для вывода на русском языке

int\* A, size; // Объявление указателя и переменной для размера массива

cout << "Введите размер массива: "; // Запрос размера массива у пользователя

cin >> size;

A = new int[size]; // Выделение памяти для массива A

srand(time(0)); // Инициализация генератора случайных чисел

// Заполнение массива A случайными числами

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

{

A[i] = rand() % 99;

}

// Вывод массива A

cout << "Массив A: ";

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

{

cout << setw(4) << A[i] << " ";

}

choiceSort(A, size); // Вызов функции сортировки выбором

delete[] A; // Освобождение памяти

}

// Функция сортировки выбором

void choiceSort(int\* D, int N)

{

int min, buf; // Объявление переменных для хранения индекса минимального элемента и временной переменной для обмена

// Внешний цикл проходит по каждому элементу массива

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

{

min = i; // Предполагаем, что текущий элемент является минимальным

// Внутренний цикл для поиска минимального элемента в оставшейся части массива

for (int j = i + 1; j < N; j++)

{

if (D[j] < D[min])

{

min = j; // Обновление индекса минимального элемента

}

}

// Обмен текущего элемента с найденным минимальным

if (i != min)

{

buf = D[i];

D[i] = D[min];

D[min] = buf;

}

}

// Вывод отсортированного массива

cout << endl << "Отсортированный массив: ";

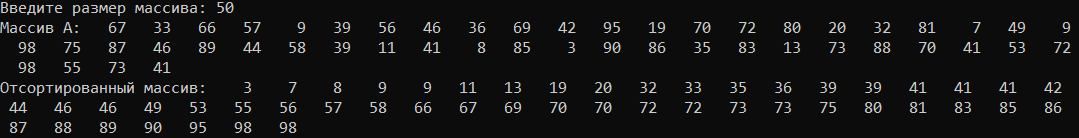
for (int i = 0; i < N; i++)

{

cout << setw(4) << D[i] << " ";

}

}

****

****

#include <iostream>

#include <ctime>

#include <iomanip>

using namespace std;

// Функция сортировки пузырьком

void bubbleSort(int\* B, int N);

// Функция преобразует поддеревье с корнем i в пирамиду

void pyr\_back(int arr[], int size, int i);

// Функция сортировки пирамидой

void pyr\_Sort(int arr[], int size);

int main()

{

setlocale(LC\_CTYPE, "rus"); // Установка локали для вывода на русском языке

int\* A, \* B, \* C, size, k = 0; // Объявление указателей и переменных

cout << "Введите размер массива: "; // Запрос размера массива у пользователя

cin >> size;

A = new int[size]; // Выделение памяти для массива A

srand(time(0)); // Инициализация генератора случайных чисел

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

{

A[i] = rand() % 99; // Заполнение массива A случайными числами

}

B = new int[size / 2]; // Выделение памяти для массива B

C = new int[size / 2]; // Выделение памяти для массива C

k = 0; // Инициализация счетчика

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

{

if (i % 2 == 0)

{

B[k] = A[i]; // Копирование четных элементов в массив B

C[k] = A[i]; // Копирование четных элементов в массив C

k++;

}

}

cout << "Массив A: "; // Вывод массива A

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

{

cout << setw(4) << A[i] << " ";

}

cout << endl << "Массив B: "; // Вывод массива B

for (int i = 0; i < size / 2; i++)

{

cout << setw(4) << B[i] << " ";

}

cout << endl << "Массив C: "; // Вывод массива B

for (int i = 0; i < size / 2; i++)

{

cout << setw(4) << C[i] << " ";

}

cout << endl << "Пузырьком - 1" << endl << "Пирамидой - 2"; // Запрос способа сортировки

char choice;

do

{

cout << endl << "?: ";

cin >> choice;

switch (choice)

{

case '1':

bubbleSort(B, k); // Сортировка пузырьком

break;

case '2':

pyr\_Sort(C, size); // Сортировка слиянием

cout << endl << "Отсортированный массив: ";

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

{

cout << setw(4) << C[i] << " ";

}

break;

}

} while (choice != '3'); // Цикл продолжается пока пользователь не введет '3'

delete[] A; // Освобождение памяти

delete[] B; // Освобождение памяти

delete[] C; // Освобождение памяти

}

void bubbleSort(int\* B, int N)

{

int temp;

// Внешний цикл

for (int i = 0; i < N - 1; i++)

{

// Внутренний цикл

for (int j = 0; j < N - i - 1; j++)

{

// Обмен элементов если текущий больше следующего

if (B[j] > B[j + 1])

{

temp = B[j];

B[j] = B[j + 1];

B[j + 1] = temp;

}

}

}

cout << endl << "Отсортированный массив: "; // Вывод отсортированного массива

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

{

cout << setw(4) << B[i] << " ";

}

}

void pyr\_back(int arr[], int size, int i) {

// Устанавливаем максимальный элемент в корень

int largest = i;

int left = 2 \* i + 1; // Индекс левого дочернего элемента

int right = 2 \* i + 2; // Индекс правого дочернего элемента

// Проверяем, существует ли левый дочерний элемент и больше ли он корня, и не выходим за границы массива

if (left < size && arr[left] > arr[largest]) {

largest = left;

}

// Проверяем, существует ли правый дочерний элемент и больше ли он корня, и не выходим за границы массива

if (right < size && arr[right] > arr[largest]) {

largest = right;

}

// Если максимальный элемент не является корнем, меняем местами

if (largest != i) {

swap(arr[i], arr[largest]);

pyr\_back(arr, size, largest); // Рекурсивно вызываем heapify для нового корня

}

}

void pyr\_Sort(int arr[], int size) {

// Преобразуем массив в пирамиду

for (int i = size / 2 - 1; i >= 0; --i) {

pyr\_back(arr, size, i);

}

// Поочередно извлекаем максимальные элементы из пирамиды

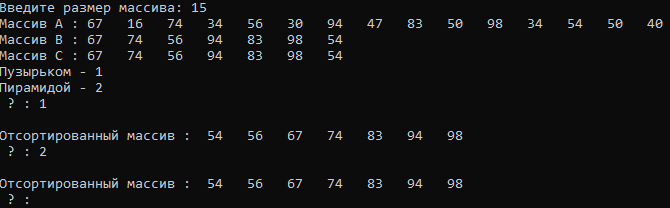
for (int i = size - 1; i > 0; --i) {

swap(arr[0], arr[i]);

pyr\_back(arr, i, 0); // Восстанавливаем пирамиду после извлечения

}

}

****