**Пример на C++**

1. **сортировка выбором**

#include <iostream>

using namespace std;

void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

// Функция сортировки выбором

void selectionSort(int arr[], int n) {

// Проходим по каждому элементу массива

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

// Предполагаем, что текущий индекс имеет минимальное значение

int min\_idx = i;

// Проверяем оставшиеся элементы справа

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

if (arr[j] < arr[min\_idx]) {

min\_idx = j;

}

}

// Если найдена позиция меньшего элемента, меняем её с текущим

if (min\_idx != i) {

swap(arr[i], arr[min\_idx]);

}

}

}

// Вспомогательная функция для печати массива

void printArray(int arr[], int size) {

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

cout << arr[i] << " ";

}

cout << endl;

}

int main() {

int arr[] = {64, 25, 12, 22, 11};

int n = sizeof(arr)/sizeof(arr[0]);

selectionSort(arr, n);

cout << "Отсортированный массив: ";

printArray(arr, n);

return 0;

}

**Пример вывода:**Отсортированный массив: 11 12 22 25 64

1. **сортировка обменом (пузырек)**

#include <iostream>

using namespace std;

// Вспомогательная функция для печати массива

void printArray(int arr[], int size) {

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

cout << arr[i] << " ";

}

cout << endl;

}

// Основная функция пузырьковой сортировки

void bubbleSort(int arr[], int n) {

bool swapped;

// Основной цикл

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

swapped = false;

// Внутренний цикл для перебора элементов

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

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

// Обмен значений

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

swapped = true;

}

}

// Если на данном этапе не было обменов, значит массив уже отсортирован

if (!swapped) {

break;

}

}

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

int n = sizeof(arr)/sizeof(arr[0]);

bubbleSort(arr, n);

cout << "Отсортированный массив: ";

printArray(arr, n);

return 0;

}

1. **Алгоритм сортировки вставками**

#include <iostream>

using namespace std;

// Вспомогательная функция для печати массива

void printArray(int arr[], int size) {

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

cout << arr[i] << " ";

}

cout << endl;

}

// Основная функция сортировки вставками

void insertionSort(int arr[], int n) {

int key, j;

// Перебираем элементы начиная со второго

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

key = arr[i];

j = i - 1;

// Смещение элементов вправо, пока не найдём позицию для текущего элемента

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j]; // сдвигаем больший элемент вправо

j--;

}

// Вставка ключа в правильную позицию

arr[j + 1] = key;

}

}

int main() {

int arr[] = {12, 11, 13, 5, 6};

int n = sizeof(arr)/sizeof(arr[0]);

insertionSort(arr, n);

cout << "Отсортированный массив: ";

printArray(arr, n);

return 0;

}

**Отсортированный массив: [3, 9, 10, 27, 38, 43, 82]**

1. **Алгоритм сортировки Шелла**

#include <iostream>

using namespace std;

// Основная функция сортировки Шелла

void shellSort(int arr[], int n) {

for (int gap = n / 2; gap > 0; gap /= 2) {

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

int temp = arr[i];

int j = i;

// Поднимаемся вверх по последовательности с шагом 'gap'

while (j >= gap && arr[j - gap] > temp) {

arr[j] = arr[j - gap];

j -= gap;

}

arr[j] = temp;

}

}

}

// Вспомогательная функция для печати массива

void printArray(int arr[], int size) {

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

cout << arr[i] << " ";

}

cout << endl;

}

int main() {

int arr[] = {9, 8, 3, 7, 5, 6, 4, 1};

int n = sizeof(arr)/sizeof(arr[0]);

shellSort(arr, n);

cout << "Отсортированный массив: ";

printArray(arr, n);

return 0;

}

Отсортированный массив: [1, 3, 4, 5, 6, 7, 8, 9]

## 6. **Быстрая сортировка (Quick Sort)**

#include <iostream>

using namespace std;

// Вспомогательная функция для разделения массива относительно опорного элемента

int partition(int arr[], int low, int high) {

int pivot = arr[(low + high) / 2]; // Опорный элемент посередине

int i = low - 1;

int j = high + 1;

while(true){

do{

i++;

}while(arr[i] < pivot);

do{

j--;

}while(arr[j] > pivot);

if(i >= j)

return j;

swap(arr[i], arr[j]); // Меняем местами элементы

}

}

// Основная функция быстрой сортировки

void quickSort(int arr[], int low, int high) {

if(low < high){

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

quickSort(arr, low, pi); // Левая сторона

quickSort(arr, pi + 1, high); // Правая сторона

}

}

// Функция для печати массива

void printArray(int arr[], int size) {

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

cout << arr[i] << " ";

cout << endl;

}

int main(){

int arr[] = {10, 7, 8, 9, 1, 5};

int n = sizeof(arr)/sizeof(arr[0]);

quickSort(arr, 0, n-1);

cout << "Отсортированный массив: ";

printArray(arr, n);

return 0;

}

Отсортированный массив: [1, 5, 7, 8, 9, 10]

7. **Пирамидальная сортировка (Heapsort)**

#include <iostream>

using namespace std;

// Поддерживаем структуру max-кучи

void heapify(int arr[], int n, int i) {

int largest = i; // Индекс наибольшего элемента

int left = 2\*i + 1; // Левый сын

int right = 2\*i + 2; // Правый сын

// Проверка существования левого сына и его максимальности

if (left < n && arr[left] > arr[largest])

largest = left;

// То же самое для правого сына

if (right < n && arr[right] > arr[largest])

largest = right;

// Если наибольшее отличается от корневого узла

if (largest != i) {

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

heapify(arr, n, largest); // Рекурсивно восстанавливаем кучу

}

}

// Основная функция пирамидальной сортировки

void heapsort(int arr[], int n) {

// Создаем max-кучу

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

heapify(arr, n, i);

// Последовательно извлекаем максимальный элемент

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

swap(arr[0], arr[i]); // Переместили максимум в конец

heapify(arr, i, 0); // Пересоздали кучу без последнего элемента

}

}

// Вспомогательная функция для печати массива

void printArray(int arr[], int size) {

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

cout << arr[i] << " ";

cout << endl;

}

int main() {

int arr[] = {12, 11, 13, 5, 6, 7};

int n = sizeof(arr)/sizeof(arr[0]);

heapsort(arr, n);

cout << "Отсортированный массив: ";

printArray(arr, n);

return 0;

}

**Отсортированный массив: [5, 6, 7, 11, 12, 13]**

**8. последовательный поиск**

#include <iostream>

using namespace std;

// Линейный поиск целевого элемента

int sequentialSearch(int arr[], int n, int target) {

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

if (arr[i] == target) {

return i; // Нашли элемент

}

}

return -1; // Элемент не найден

}

int main() {

int arr[] = {10, 20, 30, 40, 50};

int n = sizeof(arr)/sizeof(arr[0]);

int target = 30;

int result = sequentialSearch(arr, n, target);

if(result != -1) {

cout << "Элемент " << target << " найден на позиции " << result << "." << endl;

} else {

cout << "Элемент " << target << " не найден." << endl;

}

return 0;

}

**Элемент 30 найден на позиции 2.**

**9. бинарный поиск**

#include <iostream>

using namespace std;

// Бинарный поиск целевого элемента

int binarySearch(int arr[], int n, int target) {

int low = 0, high = n - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == target) {

return mid; // Нашли элемент

} else if (arr[mid] < target) {

low = mid + 1; // Продолжаем искать в правой половине

} else {

high = mid - 1; // Продолжаем искать в левой половине

}

}

return -1; // Элемент не найден

}

int main() {

int arr[] = {10, 20, 30, 40, 50};

int n = sizeof(arr)/sizeof(arr[0]);

int target = 30;

int result = binarySearch(arr, n, target);

if(result != -1) {

cout << "Элемент " << target << " найден на позиции " << result << "." << endl;

} else {

cout << "Элемент " << target << " не найден." << endl;

}

return 0;

}

**Элемент 30 найден на позиции 2.**

## **10. Интерполирующий поиск**

#include <iostream>

using namespace std;

// Интерполирующий поиск целевого элемента

int interpolationSearch(int arr[], int n, int target) {

int low = 0, high = n - 1;

while (low <= high && target >= arr[low] && target <= arr[high]) {

// Формула оценки положения элемента

int pos = low + (((double)(target - arr[low]) \* (high - low)) / (arr[high] - arr[low]));

if (arr[pos] == target) {

return pos; // Нашли элемент

} else if (arr[pos] < target) {

low = pos + 1; // Искать дальше в правой половине

} else {

high = pos - 1; // Искать дальше в левой половине

}

}

return -1; // Элемент не найден

}

int main() {

int arr[] = {10, 20, 30, 40, 50, 60, 70};

int n = sizeof(arr)/sizeof(arr[0]);

int target = 30;

int result = interpolationSearch(arr, n, target);

if(result != -1) {

cout << "Элемент " << target << " найден на позиции " << result << "." << endl;

} else {

cout << "Элемент " << target << " не найден." << endl;

}

return 0;

}

**Элемент 30 найден на позиции 2.**

11**. Фибоначчи поиск**

#include <iostream>

#include <vector>

using namespace std;

// Генерация ряда Фибоначчи до указанного предела

vector<int> generate\_fib(int limit) {

vector<int> fib = {0, 1};

while (fib.back() < limit) {

fib.push\_back(fib[fib.size()-1] + fib[fib.size()-2]);

}

return fib;

}

// Функция фибоначчи-поиска

int fibonacciSearch(int arr[], int n, int target) {

vector<int> fibNums = generate\_fib(n);

int fibM = fibNums.size() - 1;

int offset = -1;

while (fibNums[fibM] > 1) {

int i = min(offset + fibNums[fibM - 2], n - 1);

if (arr[i] < target) {

fibM -= 1;

offset = i;

} else if (arr[i] > target) {

fibM -= 2;

} else {

return i;

}

}

if (fibNums[fibM - 1] && arr[offset + 1] == target) {

return offset + 1;

}

return -1; // Элемент не найден

}

int main() {

int arr[] = {10, 20, 30, 40, 50, 60, 70};

int n = sizeof(arr)/sizeof(arr[0]);

int target = 30;

int result = fibonacciSearch(arr, n, target);

if(result != -1) {

cout << "Элемент " << target << " найден на позиции " << result << "." << endl;

} else {

cout << "Элемент " << target << " не найден." << endl;

}

return 0;

}

**Элемент 30 найден на позиции 2.**