

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА – Российский технологический университет"**

# РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра МОСИТ

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

Исследование сортировок 2

**«АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ»**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил студент группы ИКБО-16-19 . | |  | Павлов С.П. |
| Принял  д.т.н,проф |  |  | Кораблин Ю.П. |
| Лабораторные работы выполнены | «\_\_» \_\_\_\_\_\_\_ 2020 г. |  |
|  |  |  |  |
| «Зачтено» | «\_\_» \_\_\_\_\_\_\_ 2020 г.  Москва 2020 |  |  |
|

|  |  |  |
| --- | --- | --- |
|  |  |  |

Описание алгоритма

Пирамидальная сортировка (или сортировка кучей, HeapSort) — это метод сортировки сравнением, основанный на такой структуре данных как двоичная куча. Она похожа на сортировку выбором, где мы сначала ищем максимальный элемент и помещаем его в конец. Далее мы повторяем ту же операцию для оставшихся элементов.

**Алгоритм пирамидальной сортировки в порядке по возрастанию:**

1. Постройте max-heap из входных данных.
2. На данном этапе самый большой элемент хранится в корне кучи. Замените его на последний элемент кучи, а затем уменьшите ее размер на 1. Наконец, преобразуйте полученное дерево в max-heap с новым корнем.
3. Повторяйте вышеуказанные шаги, пока размер кучи больше 1.

Программа сортировок на с++

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

// Процедура для преобразования в двоичную кучу поддерева с корневым //узлом i, что является

// индексом в arr[]. n - размер кучи

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

{

int max = i;

// Инициализируем наибольший элемент как корень

int l = 2 \* i + 1;

int r = 2 \* i + 2;

// Если левый дочерний элемент больше корня

if (l < n && arr[l] > arr[max])

max = l;

// Если правый дочерний элемент больше, чем самый большой элемент на данный момент

if (r < n && arr[r] > arr[max])

max = r;

// Если самый большой элемент не корень

if (max != i)

{

int temp = arr[i];

arr[i] = arr[max];

arr[max] = temp;

// Рекурсивно преобразуем в двоичную кучу затронутое поддерево

heapify(arr, n, max);

}

}

// Основная функция, выполняющая пирамидальную сортировку

void heapSort(int arr[], int n)

{

// Построение кучи (перегруппируем массив)

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

heapify(arr, n, i);

// Один за другим извлекаем элементы из кучи

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

{

// Перемещаем текущий корень в конец

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

// вызываем процедуру heapify на уменьшенной куче

heapify(arr, i, 0);

}

}

// Быстрая сортировка

void Quick\_sort(int\* a, int first, int last) {

int f, l;

int temp, midl;

f = first;

l = last;

midl = a[(first + last) / 2];

do {

while (a[f] < midl) {

f++;

}

while (a[l] > midl) {

l--;

}

if (f <= l) {

if (a[f] > a[l]) {

temp = a[f];

a[f] = a[l];

a[l] = temp;

}

f++;

if (l > 0) {

l--;

}

}

} while (f <= l);

{

if (f < last) {

Quick\_sort(a, f, last);

}

if (l > first) {

Quick\_sort(a, first, l);

}

}

}

int main()

{

srand(time(NULL));

int n;

cin >> n;

int\* arr = new int[n];

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

{

arr[i] = rand() % 999999;

}

double star\_time1 = clock();

Quick\_sort(arr, 0, n - 1);

double end\_time1 = clock();

cout << "Result time = " << (end\_time1 - star\_time1) / 1000.0 << endl;

double start\_time = clock();

heapSort(arr, n);

double end\_time = clock();

cout << "Result time " << (end\_time - start\_time) / 1000.0;

}

Результаты выполнения программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | 1000 | 10000 | 100000 | 1000000 |
| Quick sort | 0 ms | 1 ms | 11 ms | 112 ms |
| Heap sort | 1 ms | 3 ms | 35 ms | 465 ms |