**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ИС**

ОТЧЁТ

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и Структуры Данных»**

Тема: Алгоритмы сортировки

|  |  |  |
| --- | --- | --- |
| Студент гр. 1375 |  | Подберёзский А.Д. |
| Преподаватель |  | Пелевин М.С. |

Санкт-Петербург

2022

**Цель работы:**

Ознакомление с существующими алгоритмами сортировки

**Задание:**

Реализовать сортировку [Timsort](https://ru.wikipedia.org/wiki/Timsort). В реализации должны быть выполнены все основные элементы алгоритма:

1. Сортировка вставками
2. Поиск последовательностей run
3. Подсчёт minrun
4. Слияние последовательностей run
5. Режим галопа при слиянии.

**Код программы:**

**Файл Array.h:**

#pragma once

#include<iostream>

template <typename T>

class Array

{

private:

T\* arr = NULL;

int cap = 1;

int size = 0;

//array resizing

void resize() {

cap = int(cap \* 2);

T\* newArr = new T[cap];

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

newArr[i] = arr[i];

}

delete[] arr;

arr = newArr;

}

public:

//default constructor

Array() {

}

//constructor for an array of a specific size

Array(unsigned int N) {

arr = new T[N];

size = N;

cap = N;

}

//element search by value

int findByValue(T value) {

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

if (value == arr[i]) {

return i;

}

}

}

//added element by index

void addByIndex(unsigned int index, T element) {

if (size == cap) {

resize();

}

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

arr[size - i] = arr[size - i - 1];

}

arr[index] = element;

size++;

}

//operator overload[]

T& operator[](unsigned int index) {

return arr[index];

}

//adding an element to the end of an array

void addToTheEnd(T value) {

if (size == cap || size == 0) {

resize();

}

arr[size] = value;

size++;

}

//deleting an element by index

void deleteByIndex(unsigned int index) {

for (unsigned int i = index; i < size - 1; i++) {

arr[i] = arr[i + 1];

}

arr[size - 1] = 0;

size--;

}

//getting the size of an array

int getSize() {

return size;

}

//filling an array with random numbers

void random(int N) {

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

addToTheEnd(rand() % 100 - 50);

}

}

//output array to console

void print() {

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

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

}

std::cout << std::endl;

}

//destructor

~Array() {

delete[]arr;

}

};

**Файл Stack.h:**

#pragma once

template <typename T>

class Stack {

private:

int size = 0;

struct Node {

T element;

Node\* adr = NULL;

};

Node\* next = NULL;

Node\* current = NULL;

Node\* top = NULL;

public:

//getting the element at the beginning of the stack without removing it

T peak() {

return top->element;

}

//operation to remove a new element.

void pop() {

current = top;

top = current->adr;

delete current;

size--;

}

//inserting a new element

void push(T element) {

current = top;

Node\* newStackElement = new Node;

newStackElement->adr = current;

newStackElement->element = element;

top = newStackElement;

size++;

}

//stack destructor

~Stack() {

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

current = top->adr;

delete top;

top = current;

}

}

//checking the stack for the presence of elements in it

bool isEmpty() {

if (size) {

return false;

}

else {

return true;

}

}

//get stack size

int getSize() {

return size;

}

};

**Файл sort.h:**

#pragma once

#include "Array.h"

#include "Stack.h"

//insertion sort

void InsertionSort(int\* arr, int size) {

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

int key = arr[i];

int j = i - 1;

while (j > -1 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

//function to merge subarrays

void Merge(Array<int>& arr, int start\_1, int len\_1, int len\_2) {

int start\_2 = start\_1 + len\_1; //calculating the size of the second temporary array

Array<int> arr\_1(len\_1);

Array<int> arr\_2(len\_2);

for (int i = 0; i < len\_1; i++) { //copying data to temporary arrays

arr\_1[i] = arr[start\_1 + i];

}

for (int i = 0; i < len\_2; i++) {

arr\_2[i] = arr[start\_2 + i];

}

int x = 0, y = 0, i = start\_1; //array indexes

int count\_1 = 0, count\_2 = 0, max = 7, //for gallop mode

plusIndex\_1 = 1, plusIndex\_2 = 1;

while (x + y < len\_1 + len\_2) {

if (arr\_1[x] <= arr\_2[y] && (x < len\_1)) {

arr[i] = arr\_1[x];

i++;

x++;

count\_2 = 0;

plusIndex\_2 = 1;

count\_1++;

if (count\_1 > max) { //gallop mode

while (arr\_1[x + plusIndex\_1] <= arr\_2[y] && x + plusIndex\_1 < len\_1 && y < len\_2) {

for (int j = x; j < x + plusIndex\_1; j++) {

arr[i] = arr\_1[j];

i++;

}

x += plusIndex\_1;

plusIndex\_1 += 1;

//

if (x + plusIndex\_1 > len\_1) {

plusIndex\_1 = len\_1 - x;

}

//

}

count\_1 = 0;

plusIndex\_1 = 1;

}

}

else if (y < len\_2)

{

arr[i] = arr\_2[y];

i++;

y++;

count\_1 = 0;

plusIndex\_1 = 1;

count\_2++;

if (count\_2 > max) { //gallop mode

int iterat\_2 = y + plusIndex\_2;

while (arr\_1[x] > arr\_2[y + plusIndex\_2] && y + plusIndex\_2 < len\_2 && x < len\_1) {

for (int j = y; j < y + plusIndex\_2; j++) {

arr[i] = arr\_2[j];

i++;

}

y += plusIndex\_2;

plusIndex\_2 += 1;

}

count\_2 = 0;

plusIndex\_2 = 1;

//

if (y + plusIndex\_2 > len\_2) {

plusIndex\_2 = len\_2 - y;

}

//

}

}

if (x >= len\_1) {

while (y < len\_2) {

arr[i] = arr\_2[y];

i++;

y++;

}

}

if (y >= len\_2) {

while (x < len\_1) {

arr[i] = arr\_1[x];

i++;

x++;

}

}

}

}

//getting minrun

int getMinrun(int N) {

int r = 0;

while (N >= 64) {

r |= N & 1;

N >>= 1;

}

return N + r;

}

//function to get the smallest of two numbers

template <class T>

T min(T x, T y) {

if (x < y) {

return x;

}

else {

return y;

}

}

//storage structure RUNs

struct RUN {

int size = 0;

int start = 0;

};

void TimSort(Array<int>& userArr, int N) {

Stack<RUN> runStack; //for storing ordered subarrays

RUN run = {};

run.size = 1;

RUN X = {}, Y = {}, Z = {};

int sizeOfStack = 0;

int minrun = getMinrun(N); //getting minrun

int i = 1;

while (i < N) { //splitting into RUNs and ordering

if (userArr[i - 1] <= userArr[i]) {

while (true) {

if (userArr[i - 1] <= userArr[i] && i < N) {

run.size++;

i++;

}

else {

if (run.size < minrun) {

while (run.size < minrun && i < N) {

run.size++;

i++;

}

InsertionSort(&userArr[run.start], run.size);

}

runStack.push(run);

run.start = i;

run.size = 1;

i++;

break;

}

}

}

if (userArr[i - 1] > userArr[i]) {

while (true) {

if (userArr[i - 1] > userArr[i] && i < N) {

run.size++;

i++;

}

else {

for (int i = 0; i < (run.size) / 2; i++) { //rearrange elements in ascending order

userArr[i] = userArr[i] + userArr[(run.size - 1) - i];

userArr[(run.size - 1) - i] = userArr[i] - userArr[(run.size - 1) - i];

userArr[i] = userArr[i] - userArr[(run.size - 1) - i];

}

if (run.size < minrun) {

while (run.size < minrun && i < N) {

run.size++;

i++;

}

InsertionSort(&userArr[run.start], run.size);

}

runStack.push(run);

run.start = i;

run.size = 1;

i++;

break;

}

}

}

}

X = runStack.peak();

runStack.pop();

sizeOfStack++;

while (!runStack.isEmpty()) { //merge RUNs

if (!runStack.isEmpty() && Y.size == 0) {

Y = runStack.peak();

runStack.pop();

sizeOfStack++;

}

if (!runStack.isEmpty() && Z.size == 0) {

Z = runStack.peak();

runStack.pop();

sizeOfStack++;

}

while ((Z.size <= X.size + Y.size || X.size >= Y.size) && sizeOfStack > 1) {

if (sizeOfStack >= 2 && X.size >= Y.size) {

Merge(userArr, Y.start, Y.size, X.size);

X.start = min(X.start, Y.start);

X.size = X.size + Y.size;

if (Z.size != 0) {

Y = Z;

}

else

{

break;

}

sizeOfStack--;

if (!runStack.isEmpty()) {

Z = runStack.peak();

runStack.pop();

sizeOfStack++;

}

}

if (sizeOfStack >= 3 && Z.size <= X.size + Y.size) {

if (X.size <= Z.size) {

Merge(userArr, Y.start, Y.size, X.size);

X.start = min(X.start, Y.start);

X.size = X.size + Y.size;

Y = Z;

Z.size = 0;

sizeOfStack--;

if (!runStack.isEmpty()) {

Z = runStack.peak();

runStack.pop();

sizeOfStack++;

}

}

else

{

Merge(userArr, Z.start, Z.size, Y.size);

Y.start = min(Y.start, Z.start);

Y.size = Y.size + Z.size;

Z.size = 0;

sizeOfStack--;

if (!runStack.isEmpty()) {

Z = runStack.peak();

runStack.pop();

sizeOfStack++;

}

}

}

}

}

}

**Файл main.cpp:**

#include <iostream>

#include <ctime>

#include "Stack.h"

#include "Array.h"

#include "Sort.h"

int main() {

int N = 0; //number of elements in custom array

std::cout << "Enter the number of array elements:" << std::endl;

std::cin >> N;

std::cout << std::endl << std::endl;

if (N > 1) {

Array<int> userArr; //custom array

userArr.random(N);

std::cout << "Your array has " << N << " elements:" << std::endl; //custom array output

userArr.print();

std::cout << std::endl;

unsigned int start\_time = clock();

TimSort(userArr, N); //sorting TimSort

unsigned int end\_time = clock() - start\_time;

std::cout << "Your sorted array:" << std::endl; //output sorted custom array

userArr.print();

std::cout << std::endl << "Algorithm worktime is " << end\_time << "ms" << std::endl;

}

else {

std::cout << "Invalid input" << std::endl;

}

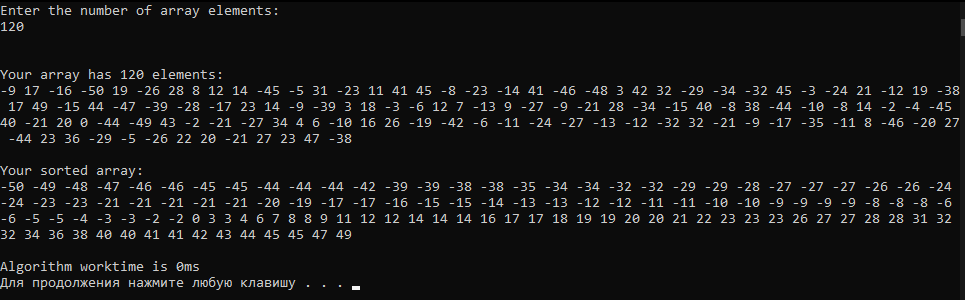
system("pause");

return 0;

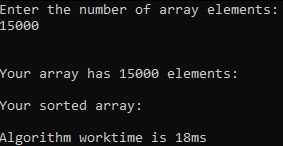
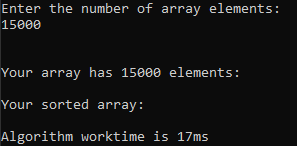
}

**Пример и время работы программы:**

При 120 элементов:



При 15000 элементов(для удобства было убрано выведение массива):



Время работы алгоритма варьируется от 17 до 20мс