ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | М. А. Мурашова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «АЛГОРИТМЫ СОРТИРОВКИ» |
| по курсу: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134к |  |  |  | Иванов И.В. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

* 1. **Цель работы**

Целью работы является изучение алгоритмов внутренней сортировки и получение практических навыков их использования, и анализа их сложности.

**1.2 Задание на лабораторную работу** Использовать неупорядоченный массив A, содержащий n целочисленных элементов. Величина n определяется по согласованию с преподавателем. Дополнительно в программе должны быть реализованы следующие функции: 1) Поиск элемента либо по его порядковой позиции, либо по его содержимому; 2) Добавление/удаление элемента с последующей пересортировкой последовательности; 3) В программе должен быть реализован подсчет количества сравнений и перестановок, при осуществлении сортировки.

Вариант 4



**Расчет сложности алгоритма**

void A::sort()

{

int\* mas = new int[size];

curr = top;

int index = 0;

for (index; index < size; index++) {

mas[index] = curr->value;

curr = curr->next;

if (curr == nullptr) {

break;

}

}

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

int maxind = 1;

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

general\_comparison += 1;

if (mas[j] > mas[maxind])

{

maxind = j;

}

}

int tmp = mas[i];

mas[i] = mas[maxind];

mas[maxind] = tmp;

general\_permutation += 1;

}

curr = top;

for (index=0; index < size; index++) {

curr->value= mas[index];

curr = curr->next;

if (curr == nullptr) {

break;

}

}

delete [] mas;

}

**Теоретическая временная сложность**

Простое извлечение во всех случаях имеет временную сложность О(n2) (два вложенных цикла, зависящих от n линейно).

|  |  |  |
| --- | --- | --- |
| Лучший случай | Средний случай | Худший случай |
| O() | O() | O() |

**Затраты по памяти**

2 int, n int массив

**Теоретическая пространственная** сложность составляет

**Листинг**

#include <iostream>

#include <ctime>

class A

{

public:

A();

~A();

class el

{

public:

el(double val) { value = val; }

~el();

void edit(int val) { value = val; };

el\* next = nullptr;

int value;

};

void add(int val);

void del(int index);

void del\_with\_sort(int index);

void show();

void show\_info();

void sort();

double get(int index);

double get\_data(int val);

int get\_size() { return size; };

el\* get\_el(int index);

void add\_with\_sort(int val);

void sent();

private:

el\* top = nullptr;

el\* curr = top;

int size = 0;

int general\_comparison = 0;

int general\_permutation = 0;

};

void A::sent() {

int\* arr = new int [size];

curr = top;

int index = 0;

for (index; index < size;index++) {

arr[index] = curr->value;

curr = curr->next;

if (curr == nullptr) {

break;

}

}

int k = 0;

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

bool fl = true;

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

if (i != j && arr[i] == arr[j]) {

fl = false;

break;

}

if (fl) k++;

}

std::cout << k;

delete[] arr;

}

void A::add(int val)

{

if (top == nullptr)

{

top = new el(val);

size++;

}

else

{

el\* current = top;

while (current->next != nullptr)

{

current = current->next;

}

current->next = new el(val);

size++;

}

}

void A::del(int index)

{

el\* current = top;

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

{

if (current == nullptr)

{

std::cout << "Error, out of range\n";

return;

}

current = current->next;

}

current->next = current->next->next;

size--;

}

void A::add\_with\_sort(int val) {

add(val);

if (size > 1)

{

sort();

}

}

void A::del\_with\_sort(int index)

{

del(index);

if (size > 1)

{

sort();

}

}

void A::show()

{

for (el\* current = top; current != nullptr; current = current->next)

{

std::cout << current->value << " ";

}

std::cout << '\n';

}

void A::show\_info()

{

std::cout << "general\_comporisons: " << general\_comparison << "\ngeneral\_permutations: " << general\_permutation << "\n";

general\_comparison = 0;

general\_permutation = 0;

}

void A::sort()

{

int\* mas = new int[size];

curr = top;

int index = 0;

for (index; index < size; index++) {

mas[index] = curr->value;

curr = curr->next;

if (curr == nullptr) {

break;

}

}

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

int maxind = 1;

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

general\_comparison += 1;

if (mas[j] > mas[maxind])

{

maxind = j;

}

}

int tmp = mas[i];

mas[i] = mas[maxind];

mas[maxind] = tmp;

general\_permutation += 1;

}

curr = top;

for (index=0; index < size; index++) {

curr->value= mas[index];

curr = curr->next;

if (curr == nullptr) {

break;

}

}

delete [] mas;

}

double A::get(int index)

{

el\* current = top;

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

{

current = current->next;

}

return current->value;

}

double A::get\_data(int val)

{

int i = 0;

for (el\* current = top; current != nullptr; current = current->next)

{

if (current->value == val)

{

return i;

}

i++;

}

return -1;

}

A::el\* A::get\_el(int index)

{

el\* current = top;

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

{

current = current->next;

}

return current;

}

A::A()

{

}

A::~A()

{

}

int main()

{

A array;

std::srand(time(0));

for (int i = 0; i < 14; i++)

{

array.add(rand() % 1000);

}

array.show();

std::cout << "1) to sort array\t2)to delete index\t3)to delete index with sort\n4)to find by index\t5)to find by value\n6) add element\n7) make a task\n";

int var = 0;

int index;

int val;

while (var < 8)

{

std::cin >> var;

switch (var)

{

case 1:

array.sort();

array.show();

array.show\_info();

std::cout << '\n';

break;

case 2:

std::cout << "Enter index: ";

std::cin >> index;

array.del(index);

array.show();

std::cout << '\n';

break;

case 3:

std::cout << "Enter index: ";

std::cin >> index;

array.del\_with\_sort(index);

array.show();

array.show\_info();

std::cout << '\n';

break;

case 4:

std::cout << "Enter index: ";

std::cin >> index;

std::cout << array.get(index) << '\n';

std::cout << '\n';

break;

case 5:

std::cout << "Enter value: ";

std::cin >> index;

std::cout << array.get\_data(index);

std::cout << '\n';

break;

case 6:

std::cin >> val;

array.add\_with\_sort(val);

array.show();

array.show\_info();

std::cout << '\n';

break;

case 7:

array.sent();

break;

default:

break;

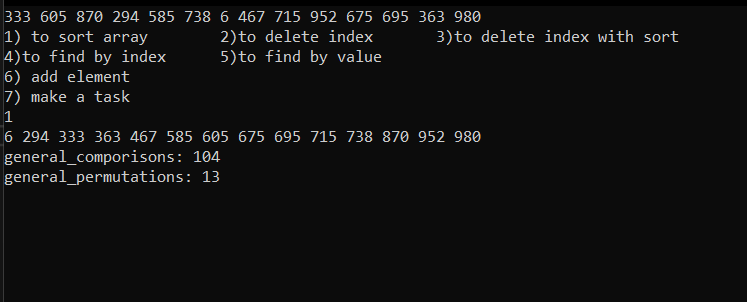
}

}

std::cout << array.get(13);

}

**Результат**



**Вывод**

Я освоил метод сортировки простым извлечением