

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

**РТУ МИРЭА**

Отчет по выполнению практического задания № 8

**Тема**: **Алгоритмы сортировок**

Дисциплина: Структуры и алгоритмы обработки данных

|  |  |
| --- | --- |
| Выполнил студент | Дмитриев П. В. |
|  | Фамилия И.О. |
| Группа | ИКБО-33-21 |
|  | Номер группы |

Москва 2022

Вариант № 7

Тема. Определение эффективного алгоритма сортировки

Цель. Получить навыки по анализу вычислительной сложности нескольких алгоритмов сортировки и определение наиболее эффективного алгоритма

Задание

Разработать три алгоритма сортировки, определенные вариантом. Провести анализ вычислительной и емкостной сложности алгоритма на массивах, заполненных случайно. Определить наиболее эффективный алгоритм.

Теоретическое введение

Временная сложность алгоритма – это необходимые для его выполнения затраты по времени. Данная характеристика представляет собой функцию, которая каждой входной длине данных ставит в соответствие максимальное (среди всех отдельных задач той же длины) время, затрачиваемое на решение этих задач.

Эмпирический (теоретический) метод анализа оценки трудоемкости и исследование производительности для каждого алгоритма в отдельности. Оценка количества сравнений, обменов и времени работы алгоритма.

Экспериментальный (практический) метод оценки трудоемкости (сложности) алгоритма. Метод основан на измерении времени выполнения алгоритма на некотором наборе данных (тесте). При этом используются стандартные средства языка программирования, позволяющие определить системное время компьютера.

ЗАДАНИЕ 1

1. **Разработка программы задачи 1**
   1. **Постановка задачи**

Оценить зависимость времени выполнения алгоритма простой сортировки на массиве, заполненном случайными числами (средний случай).

Вариант задачи – СОРТИРОВКА ПРОСТОГО ВЫБОРА

1. Разработать алгоритм сортировки одномерного целочисленного массива A[n] и реализовать его функцией, используя алгоритм согласно варианту, индивидуального задания - Алгоритм задания 1. Провести тестирование программы на исходном массиве, сформированном вводом с клавиатуры, т.е. доказать ее работоспособность.

2. Разработать функцию заполнения рабочего массива A с использованием генератора псевдослучайных чисел.

3. Провести экспериментальную оценку вычислительной сложности алгоритма, для чего выполнить контрольные прогоны программы для размеров массива n = 100, 1000, 10000, 100000 и 1000000 элементов с вычислением времени T(n) выполнения T(n) – (в миллисекундах/секундах). Полученные результаты свести в сводную таблицу Таблица 1.

4. Провести эмпирическую оценку вычислительной сложности алгоритма, определив функцию зависимости времени выполнения алгоритма от размера массива(задачи) и показать ее результат в таблице 1 в столбце Тэт =f(C+M).

5. Провести эмпирическую оценку фактического количества операций сравнения Сф и количества операций перемещения Мф. Полученные результаты вставить в сводную таблицу в столбец Тэп=Сф+Мф.

* 1. **Реализация задания**

void selection\_sort(int x[], int n) – функция сортировки массива:

1. Инициализация переменных
2. Запуск цикла от i=0 до n-1 с шагом i++
3. Присвоение переменной min индекса i
4. Запуск цикла от j=i+1 до n с шагом j++
5. Если элемент больше следующего элемента массива

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n) время в сек** | **Тэт=f(C+M)-функция** | **Т**эп**=Cф+Mф- количество** |
| 100 | 0.0000126 | O(104) | 5049 |
| 1000 | 0.0008842 | O(106) | 500499 |
| 10000 | 0.0866753 | O(108) | 50004999 |
| 100000 | 8.57113 | O(1010) | 705082703 |
| 1000000 | 877.524 | O(1012) | 1784293663 |

Таблица 1 – Сводная таблица результатов

* 1. **Код программы**

#include <iostream>

#include <chrono>

using namespace std;

void selection\_sort(int x[], int n) {

int i, j, min, temp, sravn = 0, replace = 0;

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

min = i;

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

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

min = j;

sravn++;

}

temp = x[i];

x[i] = x[min];

x[min] = temp;

replace++;

}

}

void input\_x(int x[], int n) {

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

cin >> x[i];

}

}

void input\_rand(int x[], int n) {

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

x[i] = rand() % 1000 + 1;

}

}

void output\_x(int x[], int n) {

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

cout << x[i] << " ";

}

}

int main() {

setlocale(LC\_ALL, "ru");

int n;

cout << "Введите размер массива: ";

cin >> n;

int\* x = new int[n];

//cout << "Введите " << n << " чисел: ";

//input\_x(x, n);

input\_rand(x, n);

//cout << "Исходный массив: ";

//output\_x(x, n);

auto start = chrono::high\_resolution\_clock::now();

selection\_sort(x, n);

auto end = chrono::high\_resolution\_clock::now();

cout << "\nОтсортированный массив: ";

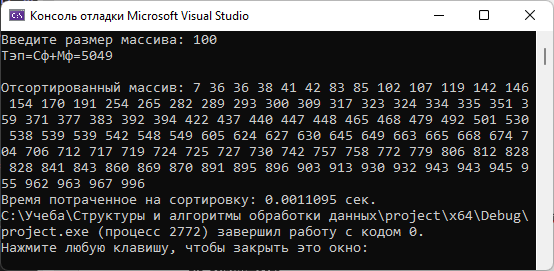
//output\_x(x, n);

chrono::duration<float> duration = end - start;

cout << "\n" << "Время потраченное на сортировку: " << duration.count() << " сек.";

}

* 1. **Скриншоты**

****

ЗАДАНИЕ 2

1. **Разработка программы задачи 2**
   1. **Постановка задачи**

Оценить зависимость времени выполнения алгоритма усовершенствованной сортировки на массиве

Вариант задача – ПИРАМИДАЛЬНАЯ СОРТИРОВКА

Требования по выполнению задания

1. Разработать алгоритм усовершенствованной сортировки (задача 2), определенной в варианте, реализовать алгоритм. Сформировать таблицу Таблица 4 результатов сортировки по формату Таблица 1 для массива, заполненного случайными числами. Определить емкостную сложность алгоритма. Определить асимптотическую сложность алгоритма.

2. Провести дополнительные прогоны программы для оценки эффективности алгоритмов в наилучшем и наихудшем случаях. Сформировать таблицы 5 и 6.

3. Выполнить анализ полученных результатов по таблицам 4, 5, 6.

4. Определить эффективный из алгоритмов (задача 1) и (задача 2) по временной и емкостной сложности.

5. Представить график зависимости Сф+Мф для анализируемых алгоритмов.

* 1. **Реализация задания**

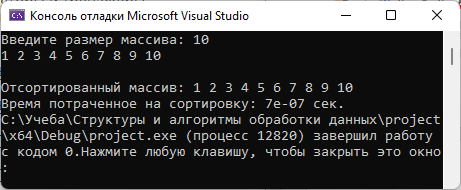
|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n) время в сек** | **Тэт=f(C+M)-функция** | **Т**эп**=Cф+Mф- количество** |
| 100 | 0.0000075 | O(460,5) |  |
| 1000 | 0.0000986 | O(6907,7) |  |
| 10000 | 0.0012574 | O(92103,4) |  |
| 100000 | 0.0159177 | O(1,15\*106) |  |
| 1000000 | 0.184502 | O(1,38\*107) |  |

Таблица 2 – Сводная таблица результатов

При наилучшем случае время работы программы при n = 10: 0.00000077 сек.

При среднем случае при n = 10: 0.0000008 сек.

При наихудшем: 0.0000008 сек.



* 1. **Код программы**

#include <iostream>

#include <chrono>

using namespace std;

// Функция "просеивания" через кучу - формирование кучи

void siftDown(int\* numbers, int root, int bottom)

{

int maxChild; // индекс максимального потомка

int done = 0; // флаг того, что куча сформирована

// Пока не дошли до последнего ряда

while ((root \* 2 <= bottom) and (!done))

{

if (root \* 2 == bottom) // если мы в последнем ряду,

maxChild = root \* 2; // запоминаем левый потомок

// иначе запоминаем больший потомок из двух

else if (numbers[root \* 2] > numbers[root \* 2 + 1])

maxChild = root \* 2;

else

maxChild = root \* 2 + 1;

// если элемент вершины меньше максимального потомка

if (numbers[root] < numbers[maxChild])

{

int temp = numbers[root]; // меняем их местами

numbers[root] = numbers[maxChild];

numbers[maxChild] = temp;

root = maxChild;

}

else

done = 1; // пирамида сформирована

}

}

// Функция сортировки на куче

void heapSort(int\* numbers, int array\_size)

{

// Формируем нижний ряд пирамиды

for (int i = (array\_size / 2); i >= 0; i--)

siftDown(numbers, i, array\_size - 1);

// Просеиваем через пирамиду остальные элементы

for (int i = array\_size - 1; i >= 1; i--)

{

int temp = numbers[0];

numbers[0] = numbers[i];

numbers[i] = temp;

siftDown(numbers, 0, i - 1);

}

}

void input\_x(int x[], int n) {

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

cin >> x[i];

}

}

void input\_rand(int x[], int n) {

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

x[i] = rand() % 1000 + 1;

}

}

void output\_x(int x[], int n) {

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

cout << x[i] << " ";

}

}

int main() {

setlocale(LC\_ALL, "ru");

int n;

cout << "Введите размер массива: ";

cin >> n;

int\* a = new int[n];

//cout << "Введите " << n << " чисел: ";

//input\_x(x, n);

input\_rand(a, n);

//cout << "Исходный массив: ";

//output\_x(x, n);

auto start = chrono::high\_resolution\_clock::now();

heapSort(a, n);

auto end = chrono::high\_resolution\_clock::now();

cout << "\nОтсортированный массив: ";

output\_x(a, n);

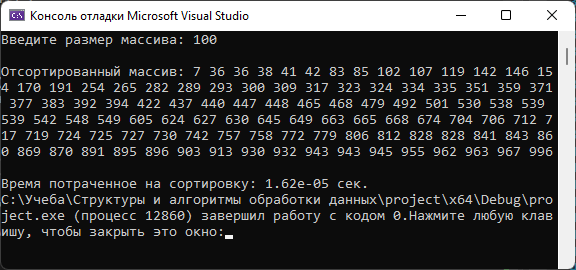
chrono::duration<float> duration = end - start;

cout << "\n" << "Время потраченное на сортировку: " << duration.count() << " сек.";

return 0;

}

* 1. **Скриншоты**



ЗАДАНИЕ 3

1. **Разработка программы задачи 3**
   1. **Постановка задачи**

Оценить зависимость времени выполнения алгоритма усовершенствованной сортировки на массиве

Вариант задача – ПРОСТОЕ СЛИЯНИЕ

Требования по выполнению задания

1. Разработать алгоритм ускоренной сортировки (задача 3), реализовать алгоритм. Сформировать таблицу Таблица 7 результатов сортировки по формату Таблица 1 для массива, заполненного случайными числами. Определить емкостную сложность алгоритма. Определить асимптотическую сложность алгоритма.

2. Провести дополнительные прогоны программы для оценки эффективности алгоритмов в наилучшем и наихудшем случаях. Сформировать таблицы 8 и 9.

3. Представить график зависимости Сф+Мф для анализируемых алгоритмов (все три алгоритма) для наихудшего случая.

* 1. **Реализация задания**

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n) время в сек** | **Тэт=f(C+M)-функция** | **Т**эп**=Cф+Mф- количество** |
| 100 | 0.0000403 | O(460,5) |  |
| 1000 | 0.0003304 | O(6907,7) |  |
| 10000 | 0.0029537 | O(92103,4) |  |
| 100000 | 0.030657 | O(1,15\*106) |  |
| 1000000 | 0.326381 | O(1,38\*107) |  |

Таблица 3 – Сводная таблица результатов

При наилучшем случае время работы программы при n = 10: 0.0000027 сек.

При среднем случае при n = 10: 0.0000025 сек.

При наихудшем: 0.0000028 сек.

* 1. **Код программы**

#include <iostream>

#include <chrono>

using namespace std;

void Merge(int\* a, int l, int r) {

int i = l;

int mid = l + (r - l) / 2;

int j = mid + 1;

int pos = 0, sravn = 0, replace = 0;

int\* b = new int[r - l + 1];

while (i <= mid and j <= r) {

if (a[i] < a[j]) {

b[pos++] = a[i++];

}

else

b[pos++] = a[j++];

}

while (i <= mid) {

b[pos++] = a[i++];

}

while (j <= r) {

b[pos++] = a[j++];

}

for (int i = l; i <= r; i++) {

a[i] = b[i - l];

}

}

void MergeSort(int\* A, int l, int r) {

int mid = l + (r - l) / 2;

if (r <= l) {

return;

}

MergeSort(A, l, mid);

MergeSort(A, mid + 1, r);

Merge(A, l, r);

}

void input\_x(int x[], int n) {

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

cin >> x[i];

}

}

void input\_rand(int x[], int n) {

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

x[i] = rand() % 100 + 1;

}

}

void output\_x(int x[], int n) {

int j = 0;

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

cout << x[i] << " ";

}

}

void main() {

setlocale(LC\_ALL, "ru");

int i, n;

cout << "Введите размер массива: ";

cin >> n;

int\* x = new int[n];

//cout << "Введите " << n << " чисел: ";

input\_x(x, n);

//input\_rand(x, n);

auto start = chrono::high\_resolution\_clock::now();

MergeSort(x, 0, n);

auto end = chrono::high\_resolution\_clock::now();

cout << "\nОтсортированный массив: ";

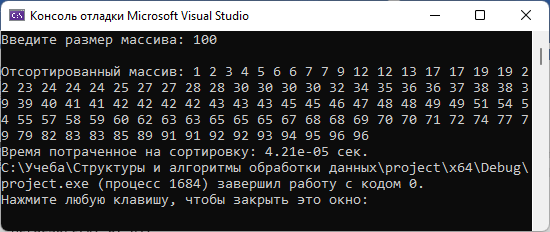
//output\_x(x, n);

chrono::duration<float> duration = end - start;

cout << "\n" << "Время потраченное на сортировку: " << duration.count() << " сек.";

}

* 1. **Скриншоты**



ВЫВОД

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название алгоритма | Асимптотическая сложность алгоритма | | | |
| Наихудший случай | Наилучший случай | Средний случай | Емкостная сложность |
| Простого выбора | O(n2) | O(n2) | O(n2) | 1 |
| Пирамидальная сортировка | O(nlogn) | O(nlogn) | O(nlogn) | 1 |
| Простое слияние | O(nlogn) | O(nlogn) | O(nlogn) | n |

Самая быстрая сортировка – пирамидальная сортировка