**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**



**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**ВЫСШАЯ ШКОЛА ПЕЧАТИ И МЕДИАИНДУСТРИИ**

***Институт Принтмедиа и информационных технологий***

***Кафедра Информатики и информационных технологий***

**направление подготовки**

**09.03.02 «Информационные системы и технологии»**

**ЛАБОРАТОРНАЯ РАБОТА № 18**

**Дисциплина:** Основы алгоритмизации и программирования

**Тема:** Вычислительная сложность алгоритмов

**Цель:** Получить практические навыки анализа сложности алгоритмов

**Выполнил: студент группы 201-723**

Карпушкин Сергей Евгеньевич

(Фамилия И.О.)



**Дата, подпись** 28.12.2020\_  ***\_\_\_\_\_\_\_\_\_\_\_***

(Дата) (Подпись)

**Проверил: \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_***

(Фамилия И.О., степень, звание) **(Оценка)**

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ***\_\_\_\_\_\_\_\_\_\_\_***

(Дата) (Подпись)

**Замечания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва**

**2020**

Оглавление

[Цель 3](#_Toc60016417)

[Задача 3](#_Toc60016418)

[Алгоритм сортировки “Пузырёк” 3](#_Toc60016419)

[Алгоритм сортировки “Расчёска” 4](#_Toc60016420)

[Алгоритм сортировки “Вставками” 5](#_Toc60016421)

[Алгоритм сортировки “Шелла” 6](#_Toc60016422)

[Алгоритм сортировки “Выбором” 7](#_Toc60016423)

[Алгоритм сортировки “Гномья” 8](#_Toc60016424)

[Алгоритм сортировки “Быстрая” 9](#_Toc60016425)

[Анализ скорости и сложности алгоритмов 11](#_Toc60016426)

# Цель

Получить практические навыки анализа сложности алгоритмов.

# Задача

Необходимо выполнить и оформить описание следующих пунктов:

1. Сформулировать идею алгоритма
2. Выполнить словесное представление алгоритма
3. Выполнить полнить представление алгоритма с помощью блок схем с использованием элемента модификации и без него.
4. Выполнить программную реализацию алгоритмов на языке С с использованием параметрического цикла и цикла с предусловием.

# Алгоритм сортировки “Пузырёк”

Понятие алгоритма: один из наиболее известных и простых алгоритмов сортировки. Он крайне лёгок в понимании, однако эффективен лишь при малых размерах массива. Средняя сложность алгоритма – квадратичная, или же О(n²). Алгоритм состоит в повторяющихся проходах по сортируемому массиву. На каждой итерации последовательно сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами. За каждый проход по массиву как минимум один элемент встает на свое место: меньший по значению элемент (более «лёгкий») продвигается к началу массива, или же «всплывает, как пузырёк», (отсюда и название).

Идея Алгоритма: алгоритм основан на повторяющихся проходах по сортируемому массиву. За каждый проход последовательно сравниваются соседние элементы. Если порядок в паре неверный, то происходит обмен значений элементов. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма). Проходы по массиву повторяются n-1 (где n – размер массива) раз или до тех пор, пока не будет перестановок, т.е. когда массив окажется отсортированным.

Листинг:

Листинг 1 – исходный код программы “пузырёк”

#include <stdio.h>

#include <time.h>

int main()

{

int j = 0, arr[100000];

int n = 100000;

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

arr[i] = rand() % n;

}

Продолжение листинга 1

int i = 0;

clock\_t begin = clock();

while (i < n - 1) {

while (j < n - 1 - i) {

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

int buf = arr[j];

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

arr[j + 1] = buf;

}

j++;

}

j = 0;

i++;

}

clock\_t end = clock();

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

printf("%d ", arr[i]);

double time\_spent = (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("Time: %f", time\_spent);

}

# Алгоритм сортировки “Расчёска”

Понятие алгоритма: это довольно упрощённый алгоритм сортировки, изначально спроектированный в 1980 г. Сортировка расчёской улучшает сортировку пузырьком, и конкурирует с алгоритмами, подобными быстрой сортировке. Основная идея — устранить черепах, или маленькие значения в конце списка, которые крайне замедляют сортировку пузырьком. В сортировке пузырьком, когда сравниваются два элемента, промежуток (расстояние друг от друга) равен 1 Основная идея сортировки расчёской в том, что этот промежуток может быть гораздо больше, чем единица.

Идея алгоритма: Алгоритм является модификацией «пузырька». Отличие алгоритмов состоит в том, что сравниваются не соседние элементы, а отстоящие друг от друга на определённую величину, или шаг (назовём его step). Алгоритм реализован с помощью двух циклов. Окончание внешнего цикла (и алгоритма) происходит тогда, когда step станет меньше 1 На первой итерации расстояние (step) максимально возможное (размер массива – 1), а на последующих итерациях оно изменяется по формуле step /= k (дробная часть отбрасывается). k – это фактор уменьшения, константа, равная 1.2473309 (при написании программы можно использовать примерное значение, равное 1.247). Во внутреннем цикле движение происходит от начала к концу, перемещаясь на step. Если значение текущего элемента больше, чем значение элемента через step шагов от текущего, то сравниваемые элементы меняются местами. Условием продолжения цикла является условие i < n – step (где i – номер текущего элемента).

Листинг:

Листинг 2 – исходный код программы “расческа”

#include <stdio.h>

#include <time.h>

int main()

{

int j = 0, arr[100000];

int n = 100000;

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

arr[i] = rand() % n;

}

float k = 1.247;

int step = n - 1;

clock\_t begin = clock();

for (int step = n - 1; step >= 1; step /= k)

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

if (arr[i] > arr[i + step])

{

int temp = arr[i];

arr[i] = arr[i + step];

arr[i + step] = temp;

}

clock\_t end = clock();

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

printf("%d ", arr[i]);

double time\_spent = (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("Time: %f", time\_spent);

}

# Алгоритм сортировки “Вставками”

Понятие алгоритма: алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

Идея алгоритма: Сортируемый массив можно разделить на две части - отсортированная часть и неотсортированная. В начале сортировки первый элемент массива считается отсортированным, все остальные - не отсортированные. Начиная со второго элемента массива и заканчивая последним, алгоритм вставляет неотсортированный элемент массива в нужную позицию в отсортированной части массива.

Листинг:

Листинг 3 – исходный код программы “вставками”

#include <stdio.h>

#include <time.h>

int main()

{

int i, j, x;

int array[100000], n=100000;

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

array[i] = rand() % n;

}

clock\_t begin = clock();

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

{

for (j = i, x = array[i]; (j > 0) && (array[j - 1] > x); j--){

array[j] = array[j - 1];

}

array[j] = x;

}

clock\_t end = clock();

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

{

printf("%d ", array[i]);

}

double time\_spent = (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("Time: %f", time\_spent);

}

# Алгоритм сортировки “Шелла”

Понятие алгоритма: алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами. Аналогичный метод усовершенствования пузырьковой сортировки называется сортировка расчёской.

Идея алгоритма: Алгоритм сортирует элементы, отстоящие друг от друга на некотором расстоянии. Затем сортировка повторяется при меньших значениях шага, и в конце процесс сортировки Шелла завершается при шаге, равном 1 (а именно обычной сортировкой вставками). Шелл предложил такую последовательность размера шага: N/2, N/4, N/8 …, где N – количество элементов в сортируемом массиве.

Листинг:

Листинг 4 – Исходный код программы “Шелла”

#include <stdio.h>

#include <time.h>

int main()

Продолжение листинга 4

{

int i, j, x, d;

int array[100000], n = 100000;

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

array[i] = rand() % n;

}

clock\_t begin = clock();

for (d = n / 2; d > 0; d /= 2)

for (i = d; i < n; i++)

{

x = array[i];

for (j = i; j >= d; j -= d)

if (x < array[j - d]) {

array[j] = array[j - d];

}

else

break;

array[j] = x;

}

clock\_t end = clock();

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

{

printf("%d ", array[i]);

}

double time\_spent = (double)(end-begin) / CLOCKS\_PER\_SEC;

printf("Time: %f", time\_spent);

}

# Алгоритм сортировки “Выбором”

Понятие алгоритма: может быть как устойчивый, так и неустойчивый. На массиве из n элементов имеет время выполнения в худшем, среднем и лучшем случае Θ(n2), предполагая, что сравнения делаются за постоянное время. Это возможно, самый простой в реализации алгоритм сортировки. Как и в большинстве других подобных алгоритмов, в его основе лежит операция сравнения. Сравнивая каждый элемент с каждым, и в случае необходимости производя обмен, метод приводит последовательность к необходимому упорядоченному виду.

Идея алгоритма: пусть имеется массив A размером N, тогда сортировка выбором сводится к следующему:

• берем первый элемент последовательности A[i], здесь i – номер элемента, для первого i равен 1;

• находим минимальный (максимальный) элемент последовательности и запоминаем его номер;

• если номер первого элемента и номер найденного элемента не совпадают, тогда два этих элемента обмениваются значениями, иначе никаких манипуляций не происходит;

• увеличиваем i на 1 и продолжаем сортировку оставшейся части массива.

С каждым последующим шагом размер подмассива, с которым работает алгоритм, уменьшается.

Листинг:

Листинг 5 – Исходный код программы “Выбором”

#include <stdio.h>

#include <time.h>

int main()

{

int i, j, min, buf;

int array[100000], N = 100000;

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

array[i] = rand() % N;

}

clock\_t begin = clock();

for (i = 0; i < N - 1; i++)

{

for (j = i + 1, min = i; j < N; j++)

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

min = j;

buf = array[i];

array[i] = array[min];

array[min] = buf;

}

clock\_t end = clock();

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

printf("%d ", array[i]);

double time\_spent = (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("Time: %f", time\_spent);

}

# Алгоритм сортировки “Гномья”

Понятие алгоритма: алгоритм сортировки, похожий на сортировку вставками, но в отличие от последней перед вставкой на нужное место происходит серия обменов, как в сортировке пузырьком. Название происходит от предполагаемого поведения садовых гномов при сортировке линии садовых горшков. Алгоритм концептуально простой, не требует вложенных циклов. Время работы O(n²). На практике алгоритм может работать так же быстро, как и сортировка вставками.

Идея алгоритма: пусть имеется массив A размером N, тогда сортировка выбором сводится к следующему:

• Смотрим на текущий и предыдущий элемент массива:

 если они в правильном порядке, шагаем на один элемент вперед,

 иначе меняем их местами и шагаем на один элемент назад.

• Граничные условия:

 если нет предыдущего элемента, шагаем вперёд;

 если нет следующего элемента, стоп.

Это оптимизированная версия с использованием переменной j, чтобы разрешить прыжок вперёд туда, где он остановился до движения влево, избегая лишних итераций и сравнений.

Листинг:

Листинг 6 – Исходный код программы “Гномья”

#include <stdio.h>

#include <time.h>

int main()

{

int i = 1, j = 2, buf;

int arr[100000], n = 100000;

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

arr[i] = rand() % n;

}

clock\_t begin = clock();

for (i = 1; i < n;)

{

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

buf = arr[i];

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

arr[i - 1] = buf;

i--;

if (i > 0) continue;

}

i = j++;

}

clock\_t end = clock();

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

printf("%d ", arr[i]);

double time\_spent = (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("Time: %f", time\_spent);

}

# Алгоритм сортировки “Быстрая”

Понятие алгоритма: один из самых известных и широко используемых алгоритмов сортировки. Среднее время работы O(n log n), что является асимптотически оптимальным временем работы для алгоритма, основанного на сравнении. Хотя время работы алгоритма для массива из n элементов в худшем случае может составить Θ(n^2), на практике этот алгоритм является одним из самых быстрых. Для этого алгоритма применяется рекурсивный метод. Рекурсией называется ситуация, когда программа вызывает сама себя непосредственно или косвенно (через другие функции).

Идея алгоритма: выбираем из массива элемент, называемый опорным, и запоминаем его значение. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Далее начинаем двигаться от начала массива по возрастающей, а потом от конца массива по убывающей. Цель: переместить в правую часть элементы больше опорного, а в левую – элементы меньше опорного. Если во время движения по возрастающей находится элемент со значением больше опорного, то мы выходим из цикла, прибавляем единицу к индексу элемента, на котором остановились, и переходим к циклу с движением по убывающей. В этом цикле мы остаемся до тех пор, пока не находится элемент со значением меньше опорного. Как только такой элемент найден, мы отнимаем единицу от его индекса, и меняем значение элемента со значением элемента, на котором мы остановились в предыдущем цикле. Делаем так до тех пор, пока индекс левого элемента (найденного в первом цикле) меньше либо равен индексу правого элемента (найденного во втором цикле). В итоге получаем два подмассива (от начала до

индекса правого элемента и от индекса левого элемента до конца). С этими подмассивами мы рекурсивно проделываем все то же самое, что и с большим массивом до тех пор, пока все элементы окончательно не отсортируются.

Листинг:

Листинг 7 – Исходный код программы “Быстрая”

#include <stdio.h>

#include <time.h>

void qsort(int\* arr, int b, int e) {

if (b < e) {

int buf, l = b, r = e, piv = arr[(b + e) / 2];

while (l <= r) {

while (arr[l] < piv)

l++;

while (arr[r] > piv)

r--;

if (l <= r) {

int t = arr[l];

arr[l] = arr[r];

arr[r] = t;

l++;

r--;

}

}

qsort(arr, b, r);

qsort(arr, l, e);

}

}

Продолжение листинга 7

int main()

{

int arr[100000], n=100000;

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

arr[i] = rand() % n;

}

clock\_t begin = clock();

qsort(arr, 0, n - 1);

clock\_t end = clock();

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

printf("%d ", arr[i]);

double time\_spent = (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("Time: %f", time\_spent);

}

# Анализ скорости и сложности алгоритмов

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Имя | Пузырёк | Расческа | Вставками | Шелла | Выбором | Гномья | Быстрая |
| Массив на 100 000 элементов | | | | | | | |
| Время (тест 1, сек) | 23,640 | 0,016 | 5,148 | 0,020 | 10,366 | 7,193 | 0,012 |
| Время (тест 2, сек) | 24,286 | 0,015 | 5,242 | 0,021 | 10,465 | 7,212 | 0,015 |
| Время (тест 3, сек) | 22,707 | 0,017 | 5,006 | 0,023 | 10,415 | 7,346 | 0,013 |
| Время (тест 4, сек) | 22,684 | 0,015 | 5,052 | 0,020 | 10,477 | 7,155 | 0,013 |
| Время (тест 5, сек) | 22,644 | 0,014 | 5,170 | 0,021 | 10,465 | 7,162 | 0,014 |
| Время (среднее, сек) | 23,1922 | 0,0154 | 5,124 | 0,021 | 10,438 | 7,214 | 0,013 |
| Сложность алгоритма | O(n^2) | O(n^2) | O(n^2) | O(n^1,25) | O(n^2) | O(n^2) | O(n \* log n) |
| Количество перестановок | ~ 2,5 млрд | ~ 713,5 тыс | ~ 2,5 млрд | ~ 2,7 млн | ~ 100 тыс | ~ 2,5 млрд | ~ 437 тыс |