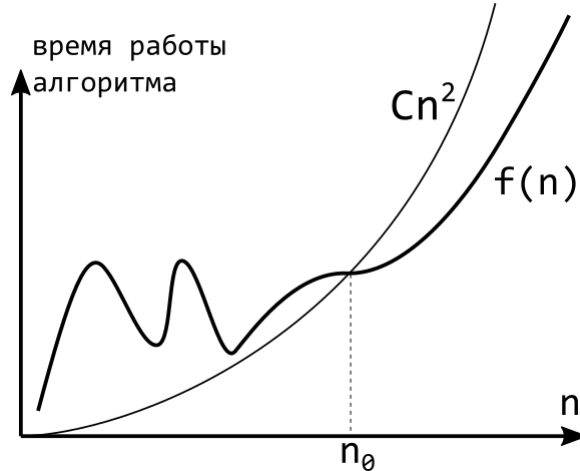


Семинар #7: Алгоритмы сортировок.

Часть 1: Вычислительная сложность. $O(n)$

Говорят, что $f(n) = O(n^2)$, если найдутся числа C и n_0 такие, что $f(n) < C \cdot n^2$, начиная с некоторого n_0 .



Другими словами: при больших n , $f(n)$ растёт не быстрее чем n^2 .

В информатике за n принимают количество символов во входных данных, а по y лежит время работы или количество элементарных операций алгоритма.

1. Расположите следующие функции в порядке увеличения скорости роста при больших n :

- | | | |
|--------------|--------------|---------|
| • $\log n$ | • 1.001^n | • n^2 |
| • 1 | • $n!$ | • 2^n |
| • \sqrt{n} | • n^n | |
| • n | • $n \log n$ | |

2. Отметьте все функции равные $O(n^2)$

- | | |
|---------------------|--|
| • $10n^2$ | • $n^2 + 10^{-1000} \cdot n^{2.00000000001}$ |
| • e^n | • $\log(n^9)$ |
| • $4n^2 + 10n + 50$ | • $n \log n$ |
| • $n!$ | • $n^3/(1+n)$ |

3. Чему равна средняя вычислительная сложность следующих операций?

- Нахождение минимального элемента в массиве размера N
- Нахождение минимального элемента в отсортированном массиве размера N
- Поиск элемента в массиве размера N
- Поиск элемента в отсортированном массиве размера N (бинарный поиск)
- Добавление элемента в начало массива размера N
- Добавление элемента в конец массива размера N (в предположении, что место под него есть)
- Сортировка пузырьком массива размера N
- Сортировка выбором массива размера N
- Быстрая сортировка массива размера N
- Добавление элемента в стек с динамическим выделением памяти размера N
- Сложение матриц размера $N \times N$
- Простой алгоритм умножения матриц размера $N \times N$

Часть 2: Быстрая сортировка - Quicksort

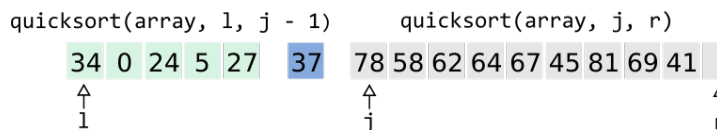
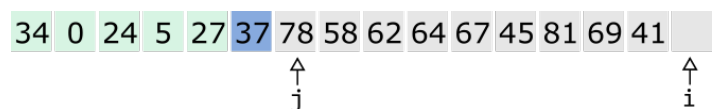
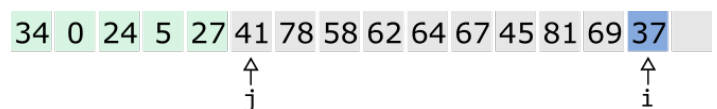
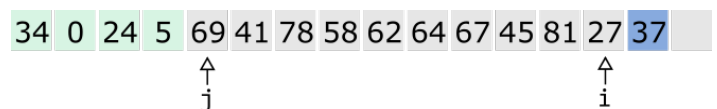
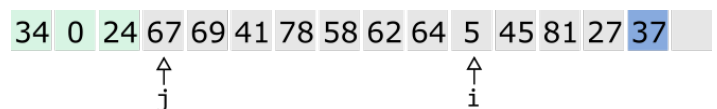
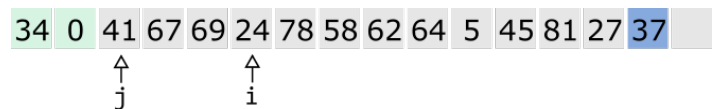
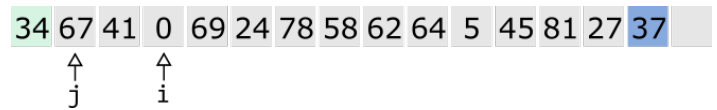
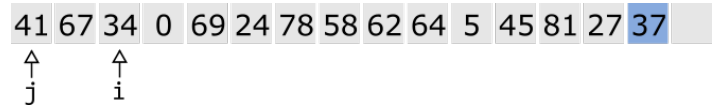
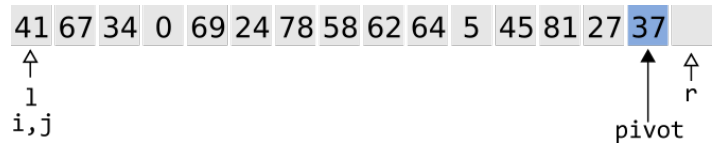
```
#include <stdio.h>
#include <stdlib.h>
#define N 30
```

```
void quicksort(int array[], int l, int r) {
    if (r - l <= 1) {
        return;
    }
    int j = l;
    int pivot = array[r - 1];
    for (int i = l; i < r; i++) {
        if (array[i] <= pivot) {
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
            j++;
        }
    }
    quicksort(array, l, j - 1);
    quicksort(array, j, r);
}
```

```
void print(int array[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", array[i]);
    printf("\n");
}
```

```
int main() {
    int numbers[N];
    for(int i = 0; i < N; i++)
        numbers[i] = rand() % 100;

    print(numbers, N);
    quicksort(numbers, 0, N);
    print(numbers, N);
}
```



- **Случайные числа:** Напишите программу, которая будет генерировать 10 случайных чисел от -10 и до 10. При каждом запуске программы числа должны быть разными. Как создать случайные числа смотрите в файлах `0random.c` и `1seedrandom.c`.
- **Сравнение сортировок:** Проведите сравнения скорости работы сортировки выбором $O(n^2)$ и быстрой сортировки $O(n \cdot \log(n))$. Для этого засекайте время работы обеих сортировок при следующих размерах $N = 100, 1000, 10^4, 10^5, 10^6, 10^7$. Код сортировок доступен в файле `2quicksort.c`. Для замера времени используйте утилиту `time` в терминале:
`time ./a.out`

При этом нужно закоментировать печать на экран, так как печать может занимать много времени. Если у Вас не работает утилита `time`, то есть альтернатива - функция `clock` из библиотеки `time.h`: cplusplus.com/reference/ctime/clock

- **По убыванию:** Измените сортировку `quicksort`, так, чтобы эта функция сортировала по убыванию.
- **По последней цифре:** Измените сортировку `quicksort`, так, чтобы эта функция сортировала по возрастанию последней цифры.
- **Сортировка вещественных чисел:** Измените сортировку `quicksort`, так, чтобы она сортировала вещественные числа. Проверьте работу сортировки. Для этого сгенерируйте массив из `N` случайных вещественных чисел от 0 до 100.
- **Города Мира:** В файле `worldcities.txt` содержится информация о различных городах мира. В каждой строке - информация об одном городе: название города, координаты - широта(`latitude`) и долгота(`longitude`), название страны и население города. В первой строке - общее количество городов.

– Опишите структуру `City`, которая будет предназначена для хранения информации об одном городе.

– **Считываем города:**

Создайте массив из структур `City` подходящего размера (нужно создать массив динамически) и считайте все данные из файла в массив. Для считывания используйте функцию `fscanf` из библиотеки `stdio.h`. Пример считывания:

```
#include <stdio.h>

int main()
{
    // Открываем файл input.txt на чтение("r"). Для открытия на запись - "w"
    FILE* f = fopen("input.txt", "r");
    fscanf(f, < тут всё то же самое, что и у обычного scanf >)
    // ...
    fclose(f);
}
```

Учтите, что спецификатор `%s` считывает строку до пробела. Чтобы считать строку до запятой используйте спецификатор `%[^,]` - при этом `s` на конец спецификатора ставить не надо.

Вся строка для `fscanf` будет выглядеть следующим образом: `"%[^,],%f,%f,%[^,],%d\n"`

– **Сохраняем города:**

Написать функцию `void save_cities(char filename[], City array[], int n)`, которая будет сохранять города из массива `array` в файл, чьё название хранится в переменной `filename`. Например, при вызове `save_cities("output.txt", cities, n)`; массив `cities` должен сохраниться в файл `output.txt`.

– **По населению:**

Создайте функцию `quicksort_population`, чтобы она принимала на вход массив из структур `City` и сортировала их по возрастанию населения. Проверьте функцию в `main`, отсортировав структуру и сохранив её в файл `sorted_by_population.txt` с помощью функции `save_cities`.

– **По широте:**

Создайте функцию `quicksort_latitude`, чтобы она сортировала массив городов по широте - от самого южного города к самому северному. Проверьте функцию в `main`, отсортировав структуру и сохранив её в файл `sorted_by_latitude.txt`.

– **По названию:**

Создайте функцию `quicksort_name`, чтобы она сортировала массив городов по названию города. Проверьте функцию в `main`, отсортировав структуру и сохранив её в файл `sorted_by_name.txt`. Используйте функцию `strcmp` из библиотеки `string.h`.

Другие сортировки

Bogosort

Bogosort - крайне неэффективная сортировка, состоящая из двух шагов:

1. Перемешать массив случайным образом.
2. Проверить, отсортирован ли массив, и, если нет, то перейти к шагу 1.

В файле `3bogosort.c` содержится реализация этой сортировки.

- Скомпилируйте эту программу (`gcc -std=c99 3bogosort.c`) и запустите. Попытайтесь отсортировать массив из 10-ти чисел с помощью этой сортировки.
- Какова вычислительная сложность этой сортировки? Какое среднее число итераций нужно произвести, чтобы отсортировать 10 чисел.

Сортировка подсчётом - Counting Sort

Быстрая сортировка и некоторые другие хорошие алгоритмы сортировки работают за $O(n \cdot \log(n))$. Такая сложность является оптимальной, если ничего больше неизвестно о сортируемых числах. Однако, если о числах что-нибудь известно, то можно добиться лучшего результата. Предположим, что наш массив состоит только из нулей и единиц и имеет примерно такой вид:

```
int array[] = {0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1,
0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,
1, 1, 1, 1};
```

Как отсортировать массив, состоящий только из нулей и единиц, за $O(n)$?

- Напишите сортировку подсчётом для такого массива (смотрите файл `4countsort_simple.c`).
- Напишите сортировку подсчётом для произвольного массива (смотрите файл `5countsort.c`).

Какие недостатки при сортировке таким методом массивов, содержащих большие числа?

Какова будет вычислительная сложность данного алгоритма для сортировки n положительных чисел, которые могут принимать значения от 0 до k .

Цифровая сортировка - Radix Sort

Цифровая сортировка заключается в последовательном применении сортировок подсчётом к цифрам числа. То есть, сначала сортируем весь массив подсчётом по последней цифре, потом по предпоследней и т. д. При этом, важно, чтобы сортировка подсчётом сохраняла порядок элементов с одинаковыми цифрами.

- Какова вычислительная сложность данного алгоритма.
- Реализуйте алгоритм цифровой сортировки и сравните его с быстрой сортировкой.

При реализации данного алгоритма нужно следить за тем, чтобы при проходе одной итерации сортировки подсчётом порядок чисел с одинаковой цифрой в текущем разряде не нарушался.