ЛЕКЦИЯ 10
АЛГОРИТМЫ ПОИСКА И СОРТИРОВКИ

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

НЕМНОГО О КОНСТАНТНОСТИ

```
int a; // a можно изменить далее в программе
const int b = 2; // b пообещали не изменять
const int c = 2;
const int * pca = &a; // указатель на константный int
*pca = 3; // FAIL
pca = &c; // OK
int const * pcb = &b; // same as pca
int * const cpa = &a; // константный указатель на int
*cpa = 3; // OK
cpa = &c; // FAIL
```

- Модификатор const означает константность того, что слева от него. Но если слева ничего нет, то он распространяется на то, что справа от него.

ДАВНИЙ ХОЛИВАР

`const` стоящая в нормативной позиции (справа от того, что должно быть константным) называется **east const** (восточный const, похоже на East Coast).

```
int const x; // неизменяемый int (east const)
const int y; // неизменяемый int (west const)
int const *x; // указатель на неизменяемый int (east const)
const int *x; // указатель на неизменяемый int (west const)
int *const x; //  неизменяемый указатель на int (east const)
```

ДАВНИЙ ХОЛИВАР

`const` стоящая в нормативной позиции (справа от того, что должно быть константным) называется **east const** (восточный const, похоже на East Coast).

```
typedef int * pint_t; // теперь p_int это int*
pint_t const x; // тоже, что int * const x
const pint_t x; // не тоже, что  const int * x
```

В случаях выше west const может ввести в заблуждение.

Увы, в существующем коде таких констант много и многие любят их писать и пишут даже принципиально

ОБЪЯВЛЕНИЕ ФУНКЦИЙ С МАССИВАМИ

Очень часто const используется в аргументах функций

```
// Эта функция может прочитать и изменить массив
void foo(int* arr, unsigned sz);

// Эта функция может только прочитать массив
void bar(const int* arr, unsigned sz);
```

Также двойственность между массивами и указателями позволяет писать

```
void foo(int arr[], unsigned sz);
void bar(const int arr[], unsigned sz);
```

ПОИСК В МАССИВАХ

На входе функции указатель на первый элемент некоторого массива и длина массива, а также искомый элемент.

```
int search(const int *parr, unsigned sz, int elem);
```

Необходимо вернуть позицию элемента от 0 до `sz - 1`, если он есть и `-1`, если его нет

```
int arr[6] = {1, 4, 10, 12, 54, 67};
int p10 = search(arr, 6, 10);
int p50 = search(arr, 6, 50);
assert(p10 == 2 && p54 == -1);
```

ЛИНЕЙНЫЙ ПОИСК

```
int search(const int *parr, unsigned sz, int elem) {
    for (int i = 0; i < sz; ++i)
        if (parr[i] == elem) return i;
    return -1;
}
```

Какая асимптотика у этого алгоритма?

ПОИСК НАИБОЛЬШЕГО И НАИМЕНЬШЕГО

На входе есть массив целых чисел. Ваша задача посчитать его наибольший и наименьший элементы.
Сможете ли вы сделать это за один проход по массиву?

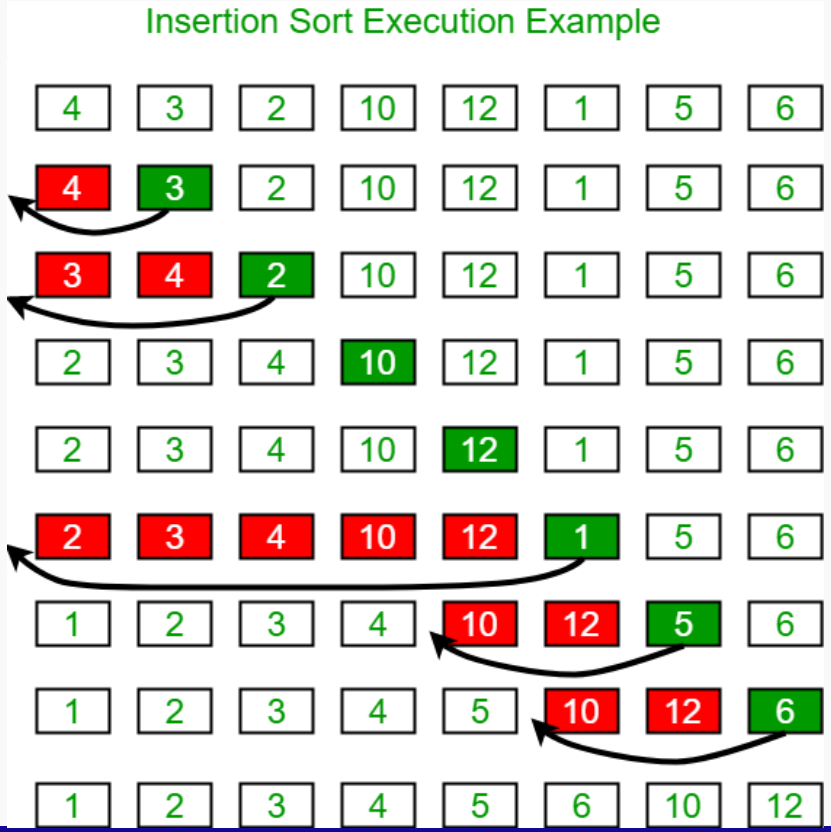
СОРТИРОВКА. НАИВНЫЙ ПОДХОД

У вас есть 300 немаркированных гирек разного веса, но одинаковой формы, и весы

Вас попросили разложить эти гири по весу в порядке возрастания

Как вы это будете делать?

СОРТИРОВКА ВСТАВКАМИ



Обычно первая идея, которая приходит, отсортировать массив вставками.

Инвариант алгоритма – левая часть массива до n всегда отсортирована

На каждом шаге n увеличивается

При необходимости все красные элементы перемещаются

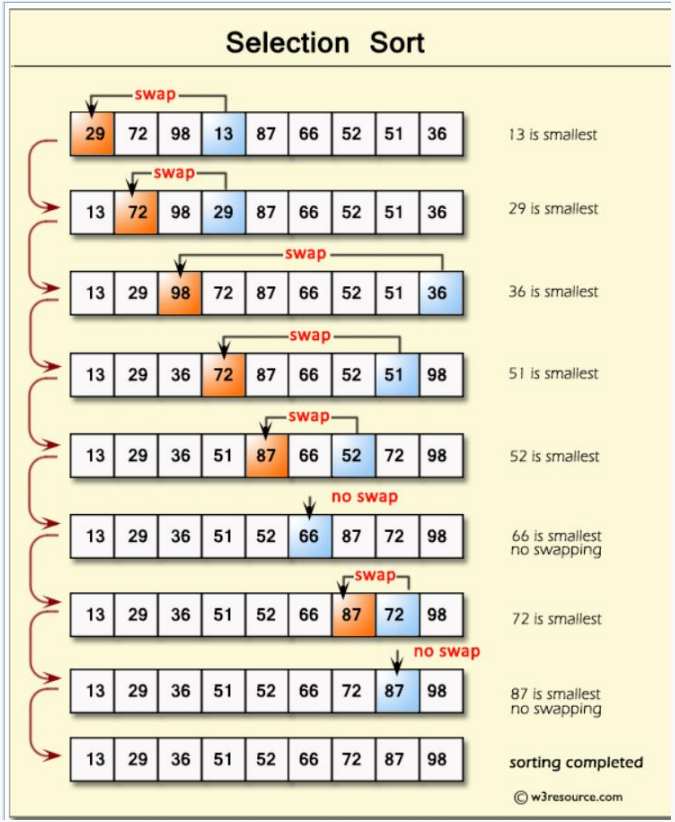
Какая асимптотика у этого алгоритма?

СОРТИРОВКА ВСТАВКАМИ

```
unsigned moveright(int *arr, int key, unsigned last) {
    // TODO напишите код этой функции
}

void insertion_sort(int *arr, unsigned sz) {
    for (unsigned i = 0; i < sz; ++i) {
        int key = arr[i];
        unsigned pos = moveright(arr, key, i);
        arr[pos] = key;
    }
}
```

СОРТИРОВКА ВЫБОРОМ



Вторая частая идея это сортировка выбором.

Найдем в массиве минимальный элемент и обменяем его местами с текущим.

Переместимся к следующему элементу.
Какая асимптотика у этого алгоритма?

СОРТИРОВКА ВЫБОРОМ

```
int search(const int *parr, unsigned sz, int elem);

void swap(unsigned *v1, unsigned *v2) {
    unsigned temp = *v1;
    *v1 = *v2;
    *v2 = temp;
}

void selection_sort(int *arr, unsigned sz) {
    // TODO Напишите код этой функции
}
```

СОМНИТЕЛЬНАЯ ИДЕЯ

Может показаться заманчивым сортируя сравнивать соседние элементы, обменивая их местами, если их взаимный порядок неправильный.

```
do {
    int sw = 0;
    for (int j = sz - 1; j > 0; --j)
        if (arr[j - 1] > a[j]) {swap(&arr[j - 1],
&a[j]); sw = 1;}
} while (sw == 1)
```

Это называется сортировка пузырьком (bubble sort)

BUBBLE VS SELECTION

Посмотрим на простом примере, почему одинаковая асимптотика не означает одинаковое быстродействие

0	1	7	4	3	6	5	2

Selection сразу найдет нужный элемент и обменяет ег с нужной позицией

Bubble тоже это сделает, но по дороге она сделает обмены со всеми остальными элементами

Формально оба сделают для одного шага $O(N)$ сравнений, но в реальности речь идет о разнице **в разы**.

ДОМАШНЕЕ ЗАДАНИЕ

1. Дан целочисленный массив, содержащий не менее четырех элементов. Напишите функцию, которая принимает указатель на этот массив, его размер и возвращает сумму двух самых маленьких по модулю чисел в нём.
2. Дан целочисленный массив, содержащий не менее четырех элементов. Напишите функцию, которая принимает указатель на этот массив, его размер и возвращает разность между самым большим и самым маленьким значениями в нем.
3. Дан целочисленный массив, содержащий не менее четырех элементов. Напишите функцию, которая принимает указатель на этот массив, его размер и индекс элемента и возвращает индекс самого маленького элемента, который больше заданного. Если такого нет, то верните -1.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Дорохова Т.Ю., Основы алгоритмизации и программирования : учебное пособие для СПО / Т.Ю. Дорохова, И.Е. Ильина. – Саратов, Москва : Профобразование, Ай, Пи Ар Медиа, 2022. – 139 с.
2. Кудинов Ю.И., Основы алгоритмизации и программирования : учебное пособие для СПО / Ю.И. Кудинов, А.Ю. Келина. – 2-е изд. – Липецк, Саратов: Липецкий государственный технический университет, Профообразование, 2020. – 71 с.
3. Дональд Кнут, Искусство программирования. Том 1. Основные алгоритмы / Ю.В. Козаченко. - 3-е изд – Москва, Санкт-Петербург: ВИЛЬЯМС, 2018. – 721 с.