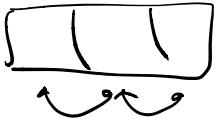
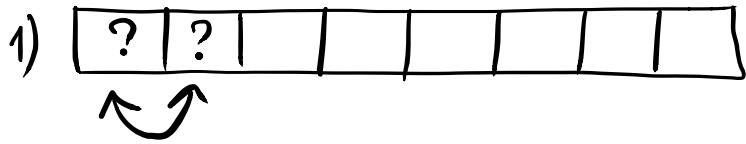


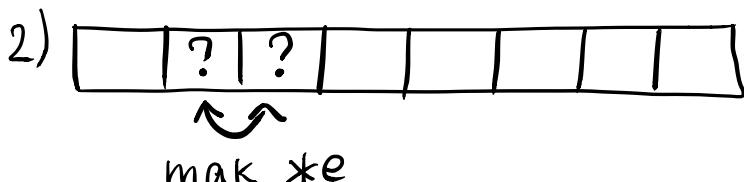
Bubble Sort

n элементов

1 шаг



сравниваем; если $>$, меняем местами



так же
⋮



так же  **Больший элемент**

В 1^{ом} шаге за $(n-1)$ операцию

отправили один элемент на своё место ($\text{arr}[n-1]$)

2 шаг Так как 1 элемент уже отсортирован, нужно сделать $n-2$ операций 

Итог шага: $\text{arr}[n-2]$ заполнен.

$n-1$ шаг Итог шага: $n-1$ элемент на

своим местом \Rightarrow на первом месте элемент тоже отсортирован

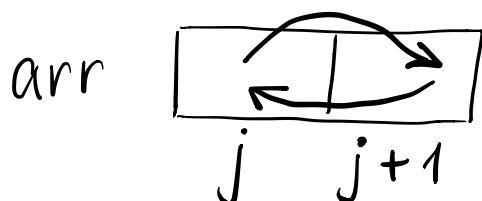
Как реализовать?

```

void bubbleSort(int arr[], int n) {
    for (int i=0; i < n-1; i++) {
        for (int j=0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                swap(&arr[j], &arr[j+1]);
            }
        }
    }
}

```

Подробнее по ф. swap



Проблема

Если массив уже

отсортирован как надо на не
последнем шаге алгоритма, он
не останавливается \Rightarrow фиксим:
добавим логическую нереализованную
bool swapped;

Модернизируем функции:

```
void bubbleSort(int arr[], int n){  
    bool swapped;  
    for (int i=0; i<n-1; i++){  
        swapped = False;  
        for (int j=0; j<n-i-1, j++){  
            if (arr[j] > arr[j+1]){  
                swap(&arr[j], &arr[j+1]);  
                swapped = True; } }  
        if (swapped == False)  $\leftarrow$  м.е.  
            break;  
    }  
}
```

*свопа не
состоилось
за весь шаг*

Функции заполнение массива

```
void fill(int arr[], int n){ \массив и его размер  
    srand(time(NULL)); \что-то для того чтобы норм работал ранд  
    for (int i = 0; i < n; i++)  
        arr[i] = rand();
```

Реализации

Очев., это $O(n^2)$

Время работы функции bubble Sort:

n	<u>без проверки</u>	<u>с проверкой</u>
10	450 нс	425 нс
10^2	22'000 нс	22'000 нс
10^4	0,18 с	0,18 с
10^8	Error	Error

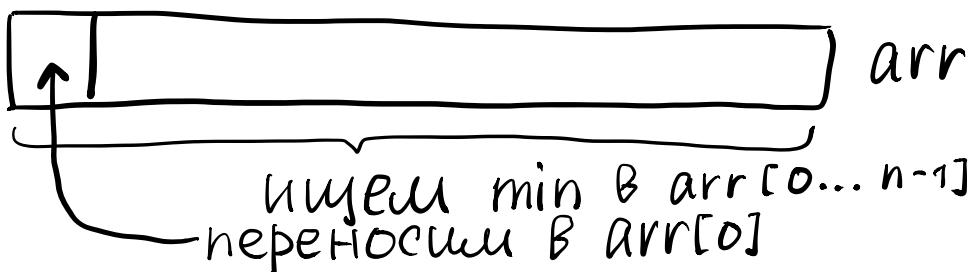
ИТОГ: нет существенной разницы.

Полезно, если нужно сортировать
малого массивов длины < 10

Сортировка выбором

Принцип

1)





ищем \min в $arr[1 \dots n-1]$
переносим в $arr[1]$



$O(n^2)$

Функции

```
void selectionSort(int arr[], int n){
    int i, j, min_idx; ← индекс мин. элемента
    for (i=0; i<n-1; i++) { ← все шаги
        min_idx = i;
        for (j=i+1; j<n; j++)
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
                swap (&arr[min_idx], &arr[i]);
            }
    }
}
```

ищем инд.
мин элемента
в

← переносим
мин в начало
подмассива

Реализации

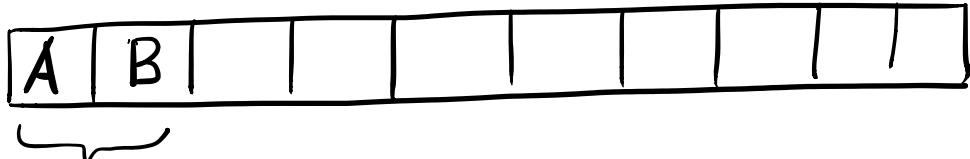
n	t
10	450 нс
10^2	15 000 нс
10^4	0,11 с

Эффективнее пузырька!

СОРТИРОВКА ВСТАВКАМИ

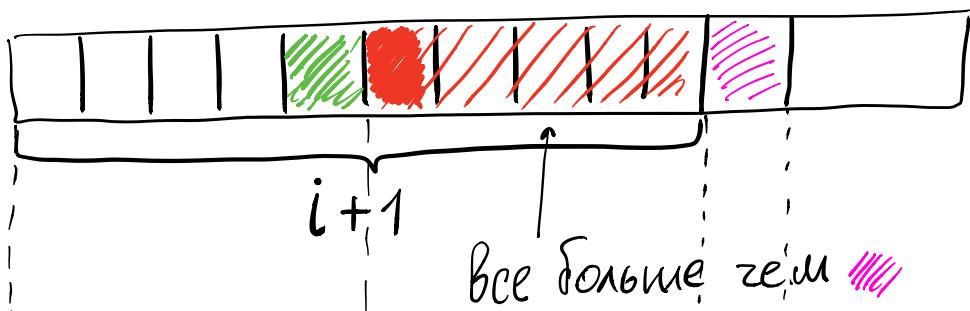
Принцип

1)



Идем справа налево элемент, который впервые меньше В. Ставим В левее этого элемента. Если такого нет, идем дальше

i)

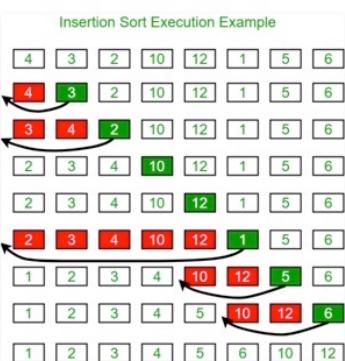


① оставляем без изм.

③ сдвигаем
на 1 адрес

② освобождаем
ячейку

④ ставим В в красное значение



Функции

```
void insertionSort(int arr[], int n){  
    int i, key, j;      key = █  
    for (i=1; i<n; i++){ █ or 1 do n-1  
        key = arr[i]; █  
        j = i-1;  
        while (j >= 0 && arr[j] > key) {  
            arr[j+1] = arr[j];  
            j = j-1; }  
        arr[j+1] = key; }  
    }  
}
```

минус, потому что
справа налево идёт

swap не используется

Реализации

n	t
10	430 нс
10^2	8000 нс
10^4	0,07с

Эффективнее сортировки выбором

СОРТИРОВКА ШЕЛЛА

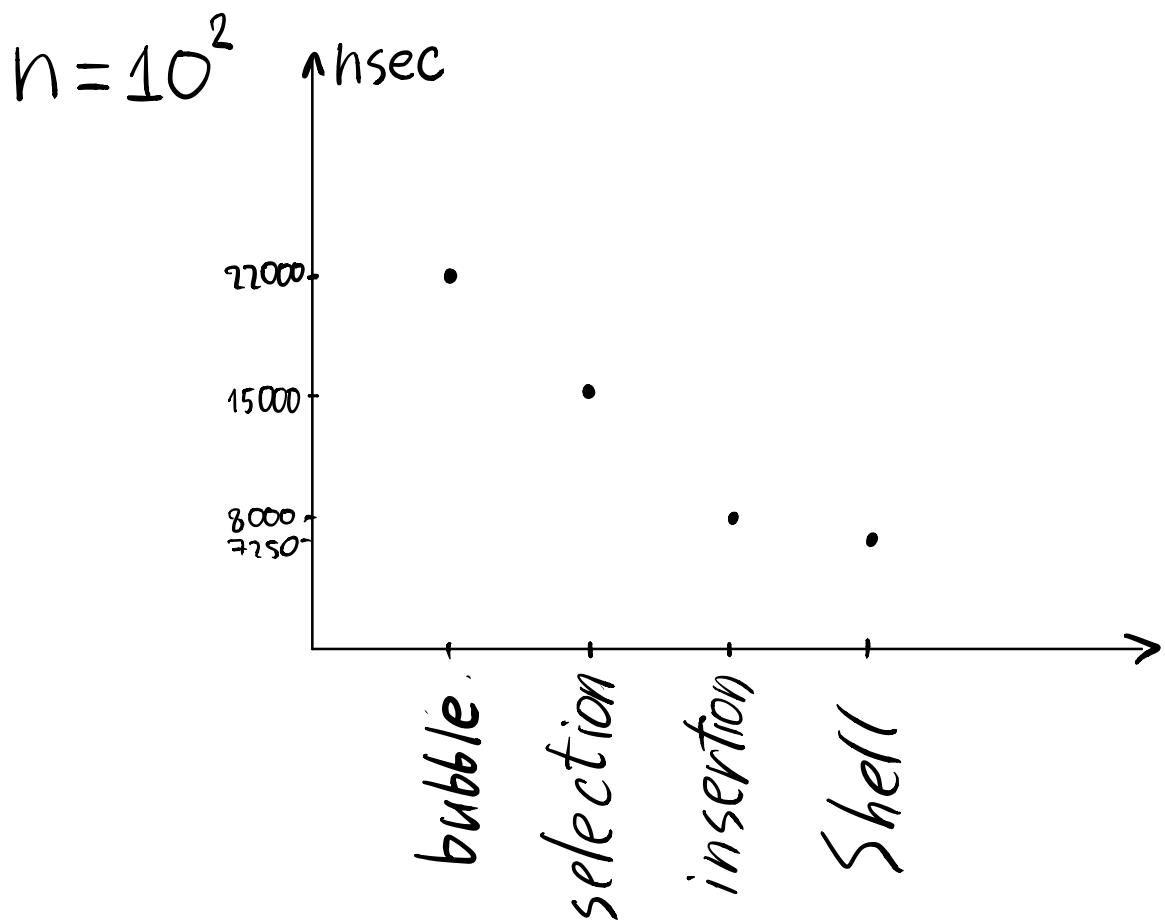
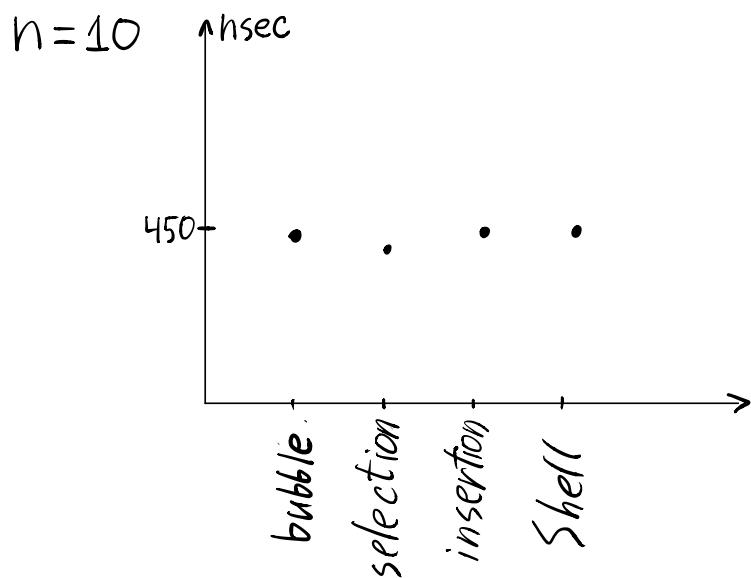
Принцип в видео на youtube Shell Sort Algorithm

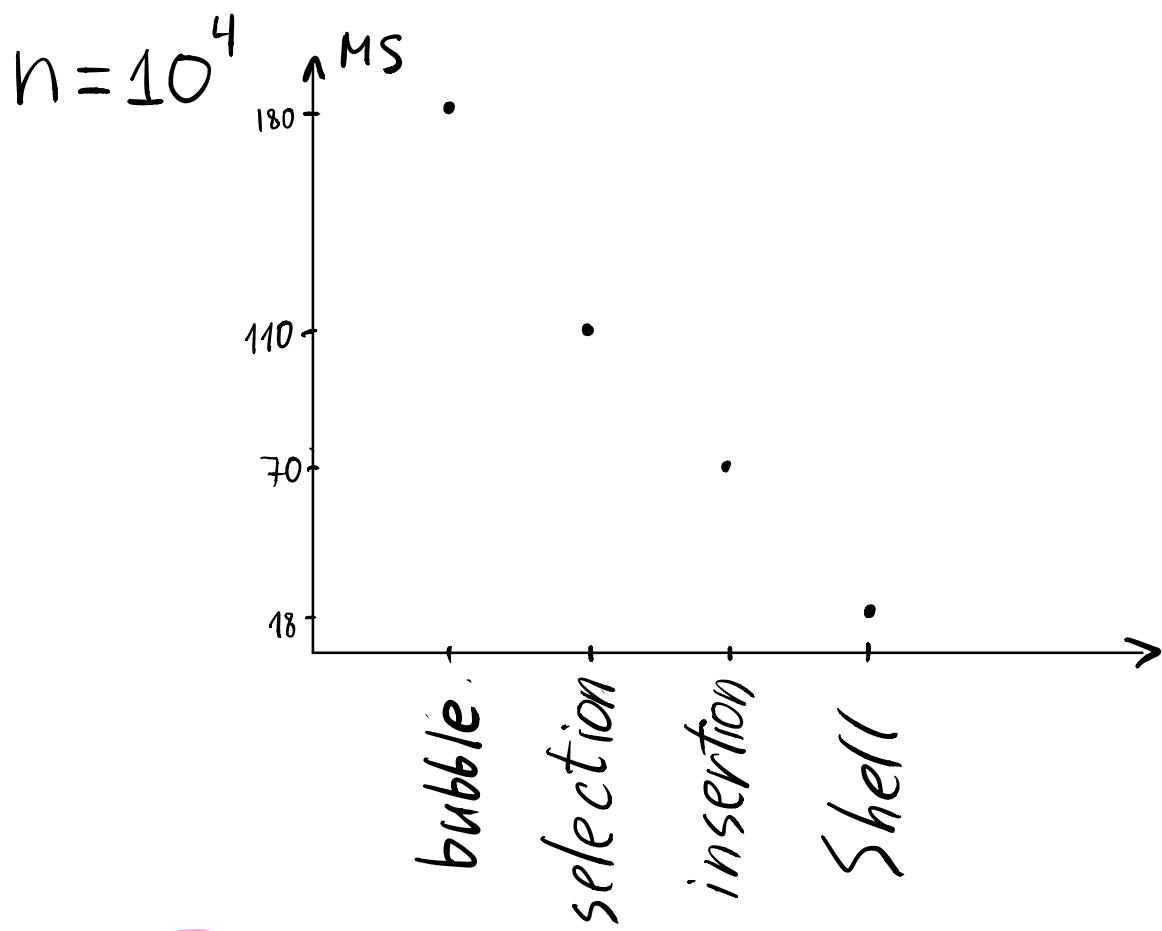
```
int shellSort(int arr[], int n)
{
    for (int gap = n/2; gap > 0; gap /= 2) менеджем gap'ов
    {
        for (int i = gap; i < n; i += 1)
        {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];
            arr[j] = temp;
        }
    }
    return 0;
}
```

некоторые шаги обрабатываются
на расстоянии шага gap
 между сравниваемыми

n	t
10	450 нс
10^2	7250 нс
10^4	0,018 с

СРАВНЕНИЕ





Вывод: Еёврость эффективности существенно заметен при $n > 10^2$. Самый эф. алгоритм - Шелла (нет) Эффективнее пузырька при $n = 10^4$ в 10 раз! При $n = 10$ разница при работе со случайными массивами не наблюдается.