Билет 12. Постановка задачи сортировки. Квадратичные сортировки.

Определение

Задача сортировки: Имеется последовательность из n элементов $x_1, x_2, \ldots, x_n$.
Необходимо найти перестановку $p_1, p_2, \ldots, p_n$ такую, что:

$$x_{p_1} \leq x_{p_2} \leq \cdots \leq x_{p_n}$$

Свойства

Классификация сортировок:

- **Основанные на сравнениях:** используют только операцию сравнения элементов
- **Устойчивые (стабильные):** сохраняют относительный порядок равных элементов
- **In-place:** используют $O(1)$ дополнительной памяти

Сортировка пузырьком (Bubble Sort)

Алгоритм

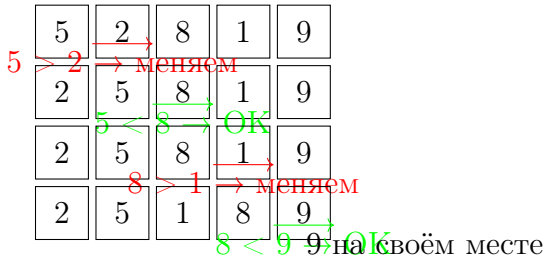
Алгоритм:

1. Выполняем n итераций
2. На каждой итерации проходим по массиву и сравниваем соседние элементы
3. Если элементы стоят в неправильном порядке — меняем их местами
4. После i-й итерации i последних элементов находятся на своих местах

Оптимизации:

- Если на итерации не было обменов — массив отсортирован
- На i-й итерации проходим только до $n - i$-го элемента

Исходный массив: [5, 2, 8, 1, 9]



Сортировка выбором (Selection Sort) - подробно

Алгоритм

Алгоритм сортировки выбором:
На i-й итерации ($i = 0, 1, \ldots, n - 2$):

1. Ищем минимальный элемент на отрезке $[i, n - 1]$
2. Меняем его местами с элементом на позиции i
3. Увеличиваем i и повторяем для оставшейся части

После $n - 1$ итераций массив полностью отсортирован.

```java
public static void instertionSort(int[] arr) {
  int n = arr.length;

  for (int i = 1; i < n; i++) {
    int key = arr[i];
    int j = i - 1;

    while (j >= 0 && arr[j] > key) {
      arr[j + 1] = arr[j];
      j--;
    }

    arr[j + 1] = key;
  }
}
```

Рис. 1: Bubble sort

Сортировка выбором: [5, 2, 8, 1, 9]

Исходный:	5	2	8	1	9
i=0:	5	2	8	1	9
i=1:	1	2	8	5	9
i=2:	1	2	5	8	9
Результат:	1	2	5	8	9

Свойства

Анализ сложности:

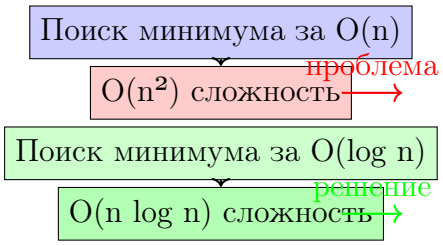
- **Количество сравнений:** $\frac{n(n-1)}{2} = O(n^2)$
- **Количество обменов:** $n - 1 = O(n)$ (максимум)
- **Память:** $O(1)$ (in-place)
- **Устойчивость:** (меняет порядок равных элементов)

Ключевая идея: ускорение через быстрый поиск минимума

Определение

Основное ограничение: Поиск минимума в неотсортированном массиве требует $O(n)$ операций.
Перспектива: Если научиться находить и извлекать минимум быстрее, чем за $O(n)$, можно получить более эффективный алгоритм.

Идея ускорения



Свойства

Как ускорить поиск минимума?

- **Двоичная куча (Heap):** позволяет находить минимум за $O(1)$ и извлекать за $O(\log n)$
- **Результат:** Heapsort со сложностью $O(n \log n)$

Сортировка вставками (Insertion Sort)

Алгоритм

Алгоритм сортировки вставками:
На i-й итерации ($i = 1, 2, \ldots, n-1$):

1. Элемент $arr[i]$ "вставляем"в отсортированный отрезок $[0, i-1]$
2. Сдвигаем элементы большие $arr[i]$ вправо
3. Вставляем $arr[i]$ на правильную позицию

После каждой итерации отрезок $[0, i]$ отсортирован.

```java
public static void instertionSort(int[] arr) {
  int n = arr.length;

  for (int i = 1; i < n; i++) {
    int key = arr[i];
    int j = i - 1;

    while (j >= 0 && arr[j] > key) {
      arr[j + 1] = arr[j];
      j--;
    }

    arr[j + 1] = key;
  }
}
```

Рис. 2: Selection Sort

Сортировка вставками: [5, 2, 8, 1, 9]

Исходный:	5	2	8	1	9
i=1:	5	2	8	1	9
i=2:	2	5	8	1	9
i=3:	2	5	8	1	9
Результат:	1	2	5	8	9

Свойства

Анализ сложности:

- **Лучший случай:** $O(n)$ (массив уже отсортирован)
- **Худший случай:** $O(n^2)$ (массив отсортирован в обратном порядке)
- **Память:** $O(1)$
- **Устойчивость:** да!

```java
public static void instertionSort(int[] arr) {
  int n = arr.length;

  for (int i = 1; i < n; i++) {
    int key = arr[i];
    int j = i - 1;

    while (j >= 0 && arr[j] > key) {
      arr[j + 1] = arr[j];
      j--;
    }

    arr[j + 1] = key;
  }
}
```

Рис. 3: Insertion Sort