Алгоритмы и структуры данных

Сортировки сравнением

Кухтичев Антон

education

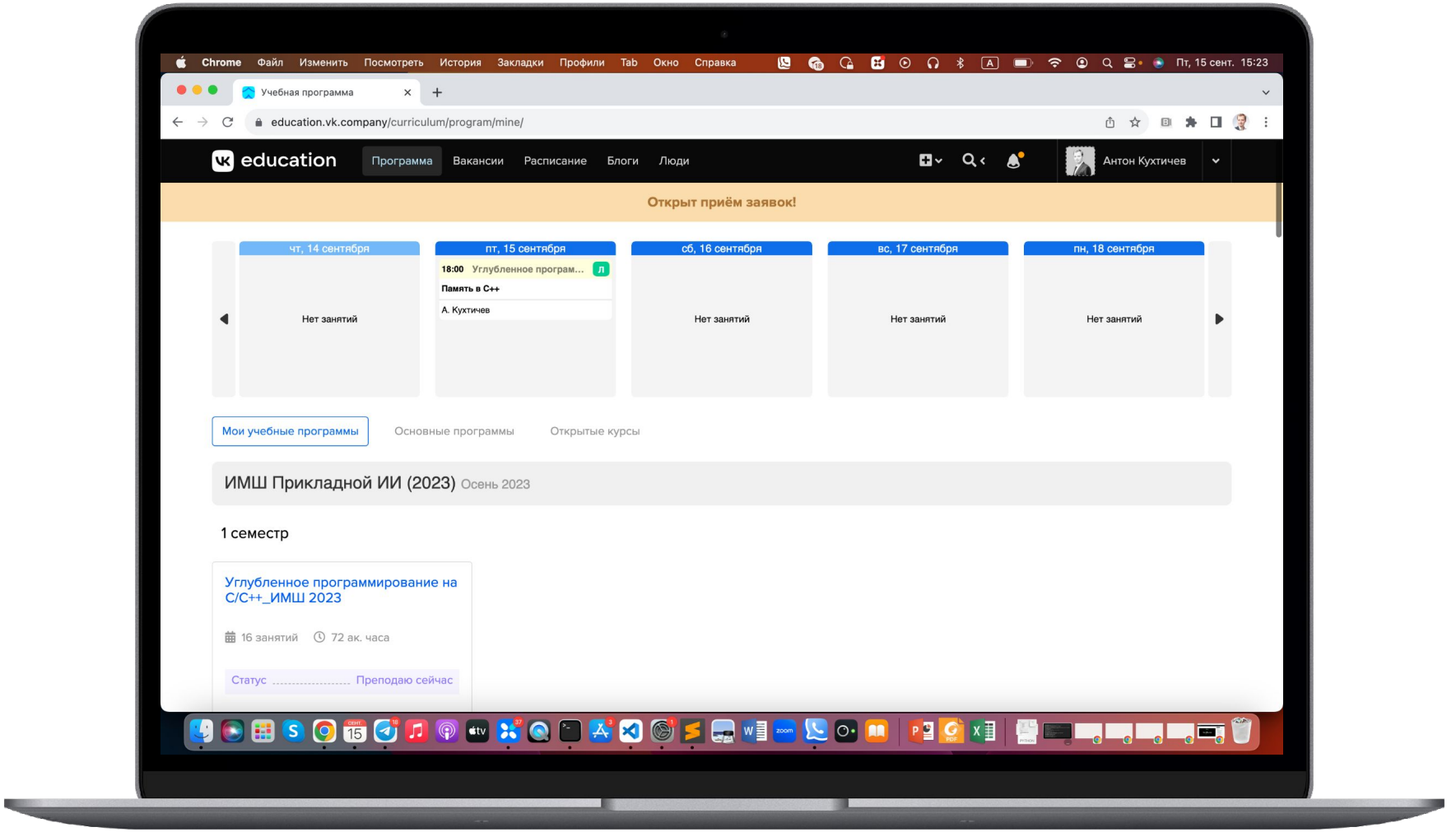
3 марта 2025 года

Содержание занятия

- Квиз
- Гномья сортировка
- Пузырьковая сортировка
- Сортировка вставками
- Сортировка слиянием
- Быстрая сортировка
- Пирамидальная сортировка
- Timsort
- Нижние оценки алгоритмов сортировки сравнением

Напоминание отметиться на портале

и оставить отзыв
после лекции



Квиз #2

Определения

Определение сортировки

Вход: последовательность из n числе $<a_1, a_2, \ldots, a_n>$

Выход: перестановка (изменения порядка) $<\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_n>$ входной последовательности таким образом, что для её членов выполняется соотношение $\hat{a}_1 \leq \hat{a}_2 \leq \ldots \leq \hat{a}_n$

Устойчивая сортировка — сортировка, которая не меняет относительный порядок сортируемых элементов, имеющих одинаковые ключи, по которым происходит сортировка.

A = <{1, ‘A’}, {0, ‘B’}, {5, ‘’C}, {0, ‘D’}>
A’ = <{0, ‘B’}, {0, ‘D’}, {1, ‘A’}, {5, ‘C’}>

Гномья сортировка (Gnome sort)

Это метод, которым садовый гном сортирует линию цветочных горшков:

- Гном смотрит на следующий и предыдущий садовые горшки:
  - если они в правильном порядке, он шагает на один горшок вперёд
  - иначе он меняет их местами и шагает на один горшок назад.

Граничные условия: если нет предыдущего горшка, он шагает вперёд; если нет следующего горшка, он закончил

- Сортировка выполняется на месте (in-place) – без использования дополнительной памяти. Сложность по памяти – O(1);

Сложность по времени:

- O(n) – в лучшем случае;
- $O(n^2)$ - в среднем и в худшем.

Гномья сортировка (Gnome sort)

```
gnome_sort(A):
      i = 1
      len = длина(A)
      while i < len; do
            if A[i] >= A[i-1]; then
                  i += 1
            else;
                  Обменять( A[i], A[i-1] )
                  i -= 1
                  if i <= 1; then
                        i += 1
            end if
      end while
```

Сортировка пузырьком

- Работает "на месте" (in-place);
- Стабильная сортировка (сохраняет порядок равных элементов);
- Сложность:
  - O(n) – в лучшем случае;
  - $O(n^2)$ - в среднем и в худшем
- Сложность по памяти:
  - O(1)

Сортировка пузырьком

```
bubble_sort(A):
  for i = 0 to n - 2; do
    for j = 0 to n - 2; do
      if a[j] > a[j + 1] ; then
        Обменять(a[j], a[j + 1])
      end if
    end for
  end for
```

Сортировка пузырьком (визуализация)

Сортировка вставками

- Алгоритм:
  - На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан.
- Сложность:
  - O(n) – в лучшем случае;
  - $O(n^2)$ - в среднем и в худшем
- Сложность по памяти:
  - O(1)

Сортировка вставками

```
insertion_sort(A):
    len = length(A)
    for j = 2 to len; do
        key = A[j]
        i = j-1
        while (i >= 0 and A[i] > key); do
            A[i + 1] = A[i]
            i = i - 1
        end while
        A[i+1] = key
    end for
```

Сортировка слиянием

Алгоритм был изобретён Джоном фон Нейманом в 1945 году.

- Алгоритм:
  - Сортируемый массив разбивается на две части примерно одинакового размера;
  - Каждая из получившихся частей сортируется отдельно;
  - Два упорядоченных массива половинного размера соединяются в один.
- Сложность:
  - O(n log n) – в лучшем случае;
  - O(n log n) - в среднем и в худшем
- Сложность по памяти:
  - O(n)

Сортировка слиянием

```
merge_sort(A):
    if длина(A) <= 1
        return A
    middle = длина(A) / 2
    L = A[0 ... middle-1]
    R = A[middle ... длина(A)-1]
    L = merge_sort(L)
    R = merge_sort(R)
    return merge(L, R)
```

Быстрая сортировка

- Основан на парадигме “разделяй и властвуй”
- Массив разбивается на два (возможно, пустых) подмассива
- Подмассивы сортируются путём рекурсивного вызова процедуры быстрой сортировки
- Сложность:
  - O(n log n) – в среднем и в лучшем случае;
  - $O(n^2)$ - в худшем
- Сложность по памяти:
  - O(1)

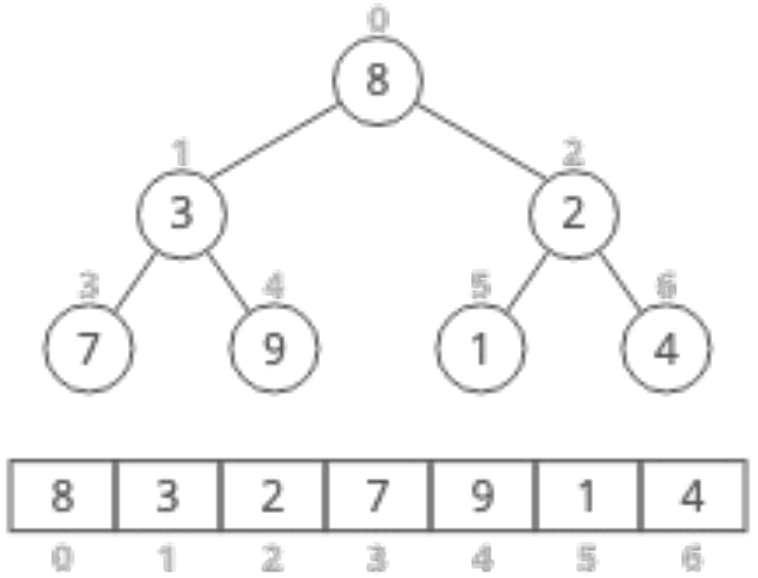
Быстрая сортировка

```
quick_sort(A, p, r)
  if p < r; then
    q = partition(A, p, r)
    quick_sort(A, p, q-1)
    quick_sort(A, q+1, r)
  end if
```

```
partition(A, p, r):
  x = A[r]
  i = p - 1
  for j = p to r - 1 ; do
    if A[j] <= x ; then
      i = i + 1
      Обменять (A[i], A[j])
    end if
  Обменять(A[i+1], A[r])
  return i+1
```

Пирамидальная сортировка

- Основан на структуре heap (пирамида, куча);
- Массив представляется в виде бинарное дерева
  - A[0] – корень;
  - Для A[i] элемента:
    - A[2*i + 1] – левый сын
    - A[2*i + 2] – правый сын
- Сложность:
  - O(n log n) – во всех случаях
- Сложность по памяти:
  - O(1)



Timsort

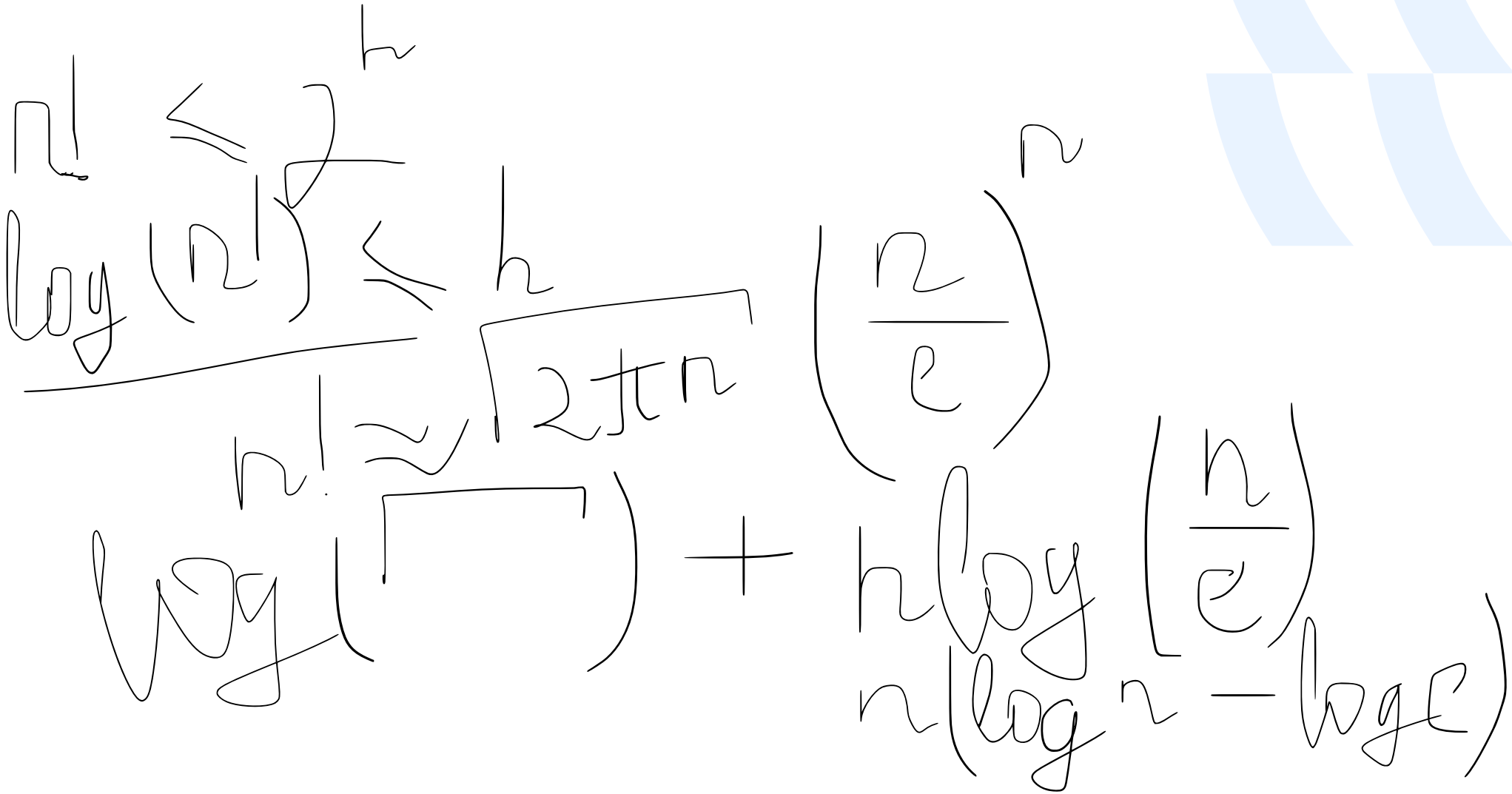
- Опубликованный в 2002 году Тимом Петерсом.
- Алгоритм
  - определение минимального размера подмассива массива;
  - деление входного массива на подмассивы с использованием специального алгоритма;
  - сортировка каждого подмассива с использованием алгоритма сортировки вставками;
  - объединение отсортированных подмассивов в массив с использованием изменённого алгоритма сортировки слиянием.
- Сложность:
  - O(n) – в лучшем случае;
  - O(n log n) - в среднем и в худшем

Нижние оценки алгоритмов сортировки сравнением

- Для оценки нижней границы используется *деревья решений* (decisions trees)
- Для входного массива размера n количество листьев будет $n!$
- Количество листьев дерева для высоты — 2^h
- **Теорема**

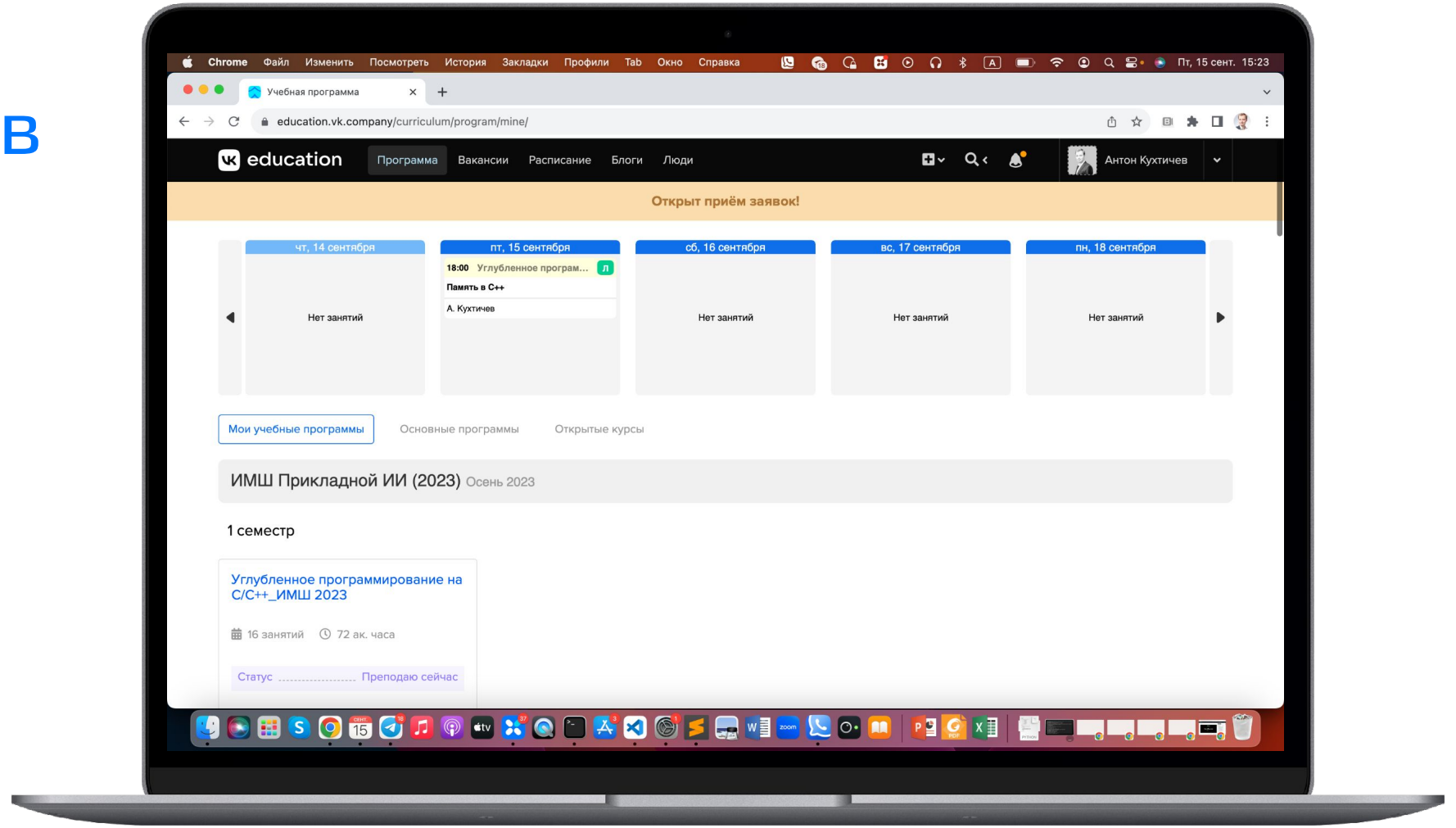
  Любой алгоритм сортировки в наихудшем случае требует выполнения $\Omega(n \log n)$ сравнений

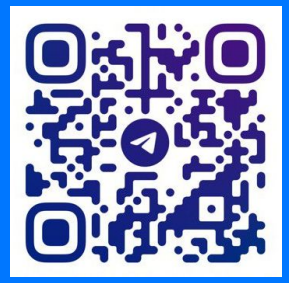
Нижние оценки алгоритмов сортировки сравнением



Напоминание оставить отзыв

Это правда важно





Спасибо
за внимание!