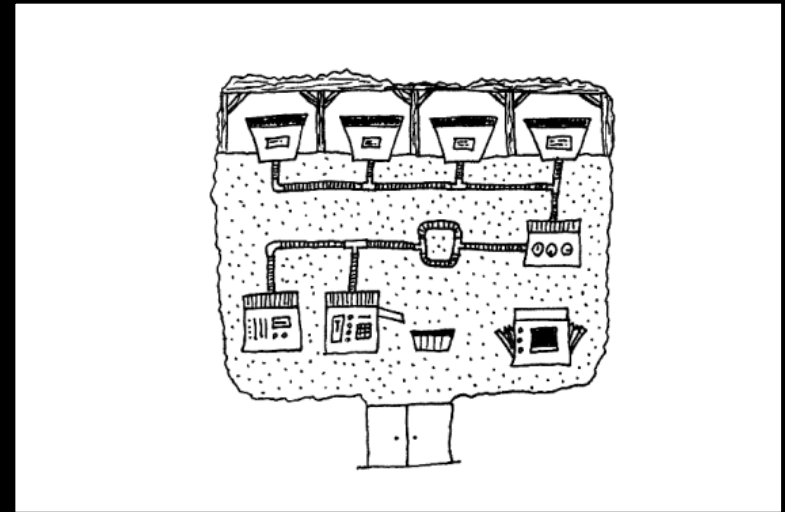




Алгоритмы и структуры данных



Сортировки

- **Сортировка** – процесс упорядочивания элементов массива
- Многие программы используют алгоритмы сортировки.
Часто время их работы определяется временем сортировки
- Данные часто упорядочены каким либо
- Многие задачи существенно быстрее решаются на предварительно упорядоченных данных



1. Простые сортировки
2. Предел скорости
3. Ускорение
4. Хорошие сортировки
5. Особые свойства
6. Поразрядная сортировка
7. Сравнение

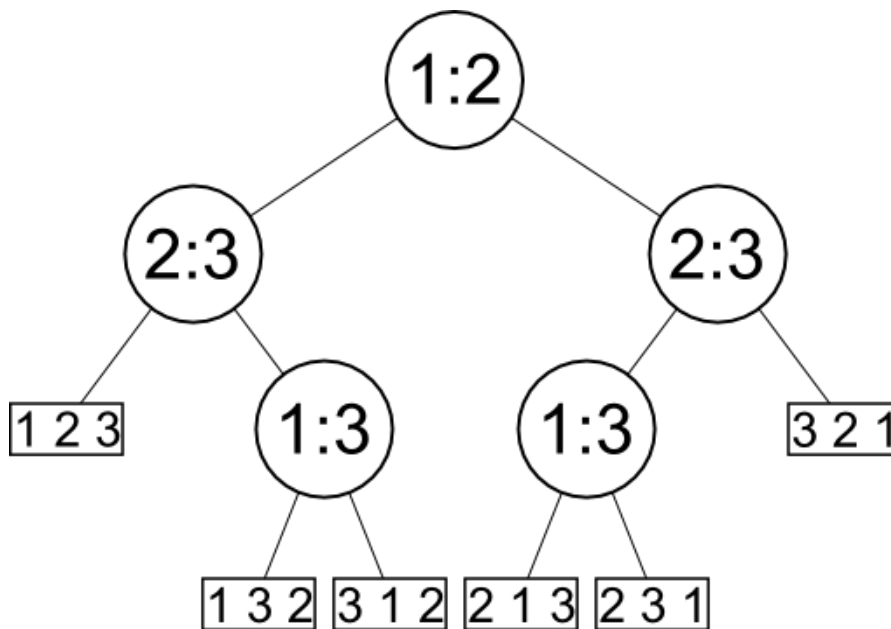


Сортировка 1 и 2х элементов



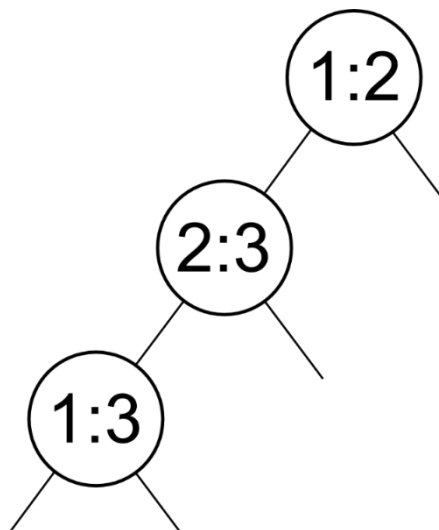
```
void sort_1(int *a) {  
    return;  
}  
  
void sort_2(int *a) {  
    if (a[1] < a[0]) {  
        swap(a[1], a[0]);  
    }  
}
```

Сортировка 3х элементов



```
void sort_3(int *a) {  
    if (a[1] < a[0]) {  
        if (a[2] < a[1]) {  
            // 2 1 0  
        } else {  
            if (a[2] < a[0]) {  
                // 1 2 0  
            } else {  
                // 1 0 2  
            }  
        }  
    } else {  
        if (a[2] < a[0]) {  
            // 2 0 1  
        } else {  
            if (a[2] < a[1]) {  
                // 0 2 1  
            } else {  
                // 0 1 2  
            }  
        }  
    }  
}
```

Избыточное сравнение



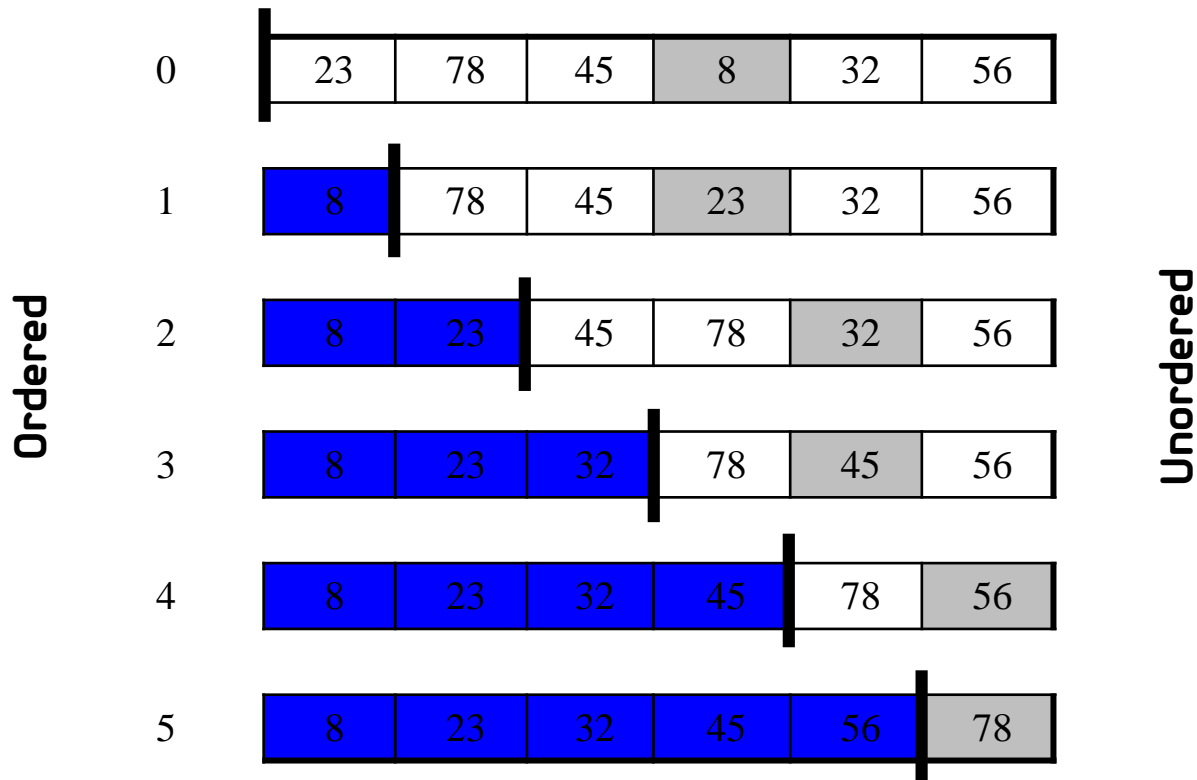
- Существует множество алгоритмов сортировка
 - Сортировка выбором - Selection Sort
 - Сортировка вставками - Insertion Sort
 - Пузырьковая сортировка - Bubble Sort

Сортировка выбором



- Разделим массив на две части: левую - упорядоченную и правую – неупорядоченную
- Будем гарантировать, что элементы в правой части больше чем в левой
- Выберем наименьший элемент в правой части и переставим его в её начало
- После каждого выбора и перестановки, будем смещать границу между частями массива на **1** вправо
- Выбор каждого нового элемента требует прохода по правой части
- Для сортировки массива из **N** элементов требуется **$N-1$** проход

Сортировка выбором



Сортировка выбором



```
void selection_sort(int *a, int n) {
    for (int i = 0; i < n - 1; ++i) {
        int min_index = i;
        for (int j = i + 1; j < n; ++j) {
            if (a[j] < a[min_index]) min_index = j;
        }
        swap(a[i], a[min_index]);
    }
}

void swap(int &a, int &b ) {
    int tmp = a;
    a = b;
    b = tmp;
}
```

Сортировка выбором: Анализ



- В общем случае алгоритм сортировки состоит из сравнения ключей и перестановки элементов
- Время работы алгоритма пропорционально количеству сравнений и количеству перестановок
- Внешний цикл совершает $n-1$ итерацию
- В каждой итерации 1 перестановка
- В 1й итерации $n-1$ сравнение, во 2й – $n-2$, ... в $n-1$ й – 1
- Ровно $n(n-1)/2$ сравнений
- Ровно $n-1$ перемещений

Сортировка выбором: Анализ



- Количество операций не зависит от состава массива
- Сложность в лучшем и в худшем случае совпадает: $O(n^2)$

- Сортировка вставками – простой алгоритм часто применяемый на малых объёмах данных
 - Самый популярный метод сортировки у игроков в покер
- Массив делится на две части, упорядоченную - левую и неупорядоченную - правую
- На каждой итерации выбираем элемент из правой части и вставляем его на подходящее место в левой части
- Массив из n элементов требует $n-1$ итерацию

Сортировка вставками

Ordered

23	78	45	8	32	56
----	----	----	---	----	----

23	78	45	8	32	56
----	----	----	---	----	----

23	45	78	8	32	56
----	----	----	---	----	----

8	23	45	78	32	56
---	----	----	----	----	----

8	23	32	45	78	56
---	----	----	----	----	----

8	23	32	45	56	78
---	----	----	----	----	----

Unordered

Сортировка вставками



```
void insertion_sort(int *a, int n) {  
    for (int i = 1; i < n; ++i) {  
        int tmp = a[i];  
        for (int j = i; j > 0 && tmp < a[j-1]; --j) {  
            a[j] = a[j-1];  
        }  
        a[j] = tmp;  
    }  
}
```

Сортировка вставками: Анализ



- Время работы алгоритма зависит не только от размера массива, но и от порядка элементов

- **Лучший случай:** $\rightarrow O(n)$

- Массив упорядочен по возрастанию
- Внутренний цикл делает 0 итераций
- Количество копирований: $2 \cdot (n-1) \rightarrow O(n)$
- Количество сравнений: $(n-1) \rightarrow O(n)$

- **Худший случай:** $\rightarrow O(n^2)$

- Массив упорядочен в порядке убывания:
- Внутренний цикл работает $i-1$ итерацию, для $i = 2, 3, \dots, n$
- Количество копирований: $2 \cdot (n-1) + (1+2+\dots+n-1) = 2 \cdot (n-1) + n \cdot (n-1) / 2 \rightarrow O(n^2)$
- Количество сравнений: $(1+2+\dots+n-1) = n \cdot (n-1) / 2 \rightarrow O(n^2)$

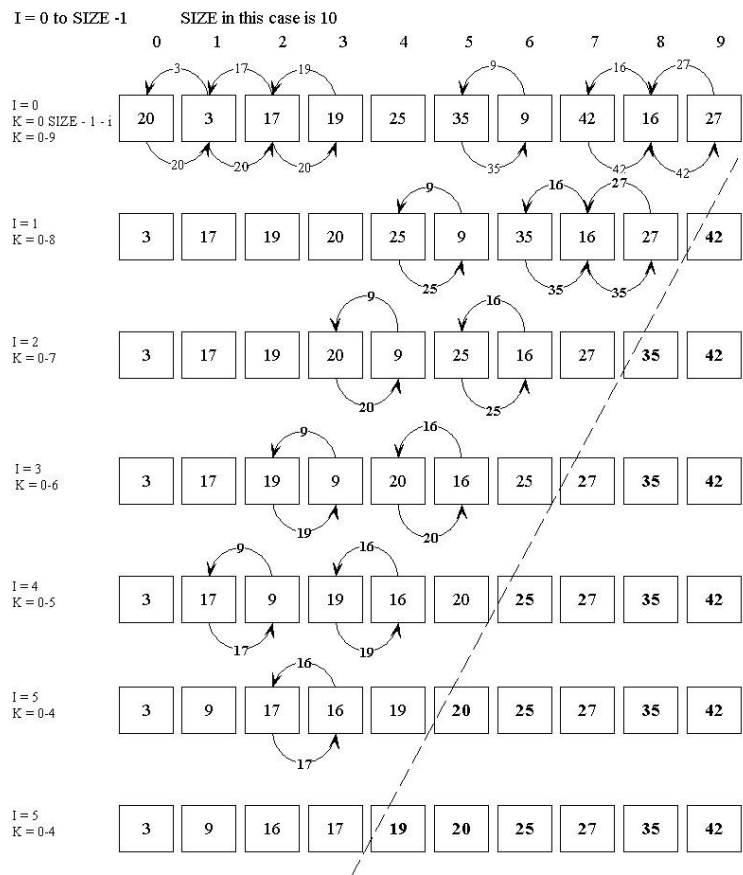
- **В среднем:** $\rightarrow O(n^2)$

Сортировка пузырьком



- Сортировка пузырьком – частный случай сортировки выбором
- Массив делится на две части, упорядоченную - левую и неупорядоченную - правую
- На каждой итерации проходим правую часть сравнивая текущий элемент с соседом слева
 - меняем элементы местами если сосед больше
 - Иначе, уменьшаем индекс текущего на **1**
- Наименьший элемент всплывает к границе левой части
- Останавливаемся если не было ни одного обмена
- Массив из n элементов требует максимум **$n-1$** итерацию

Сортировка пузырьком



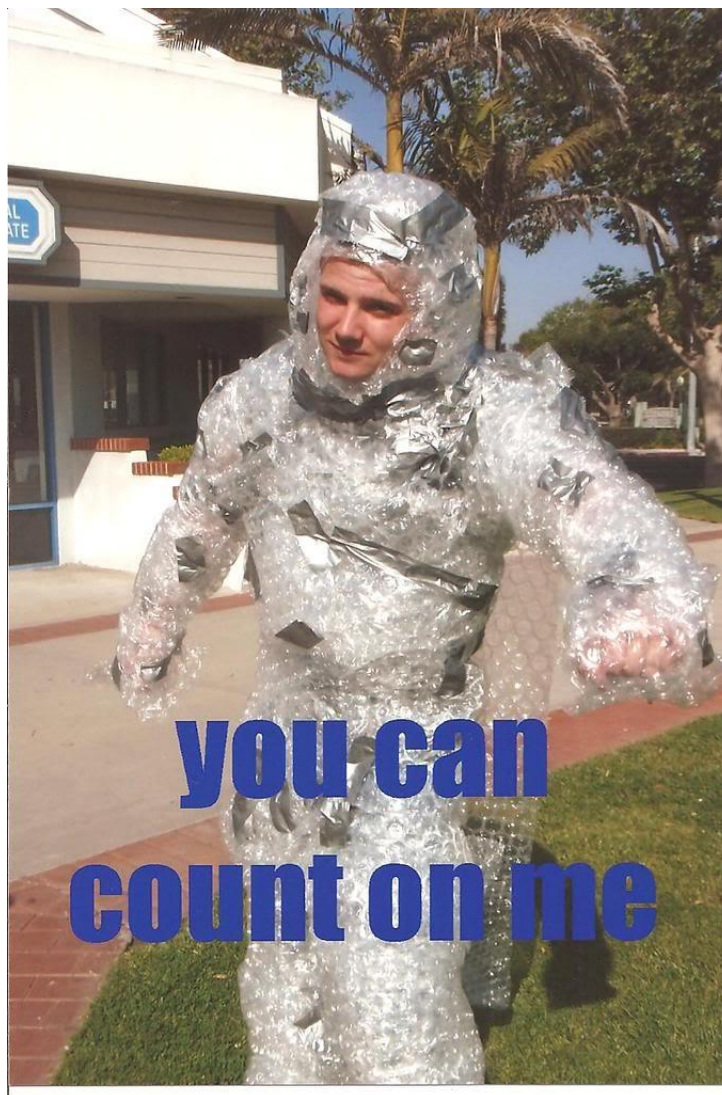
Because nothing was found out of order on the last pass we quit.

Сортировка пузырьком: Анализ



- Лучший случай
 - 1 проход, $N-1$ сравнение, 0 обменов $\Rightarrow O(N)$
- Худший случай
 - $N-1$ проход, $N(N-1)/2$ сравнений, $N-1$ обменов $\Rightarrow O(N^2)$

Сортировка пузырьком



**you can
count on me**

- Оценка сложности

- Количество перестановок N элементов: $N!$
- 1 сравнение = 1 бит информации
- Для записи номера перестановки нужно

$$\log_2(N!) \cong N \log(N) \text{ бит}$$

Сортировка вставками. $n(\log(n))$?



- Будем искать позицию вставки в упорядоченную часть массива бинарным поиском
- $O(n \log(n))$ - операций сравнения
- $O(n^2)$ – операций перемещения
- Используем `memcpy` для уменьшения константы C_2
- $T(n) \leq C_1 n \log(n) + C_2 n^2$

Сортировка вставками. $n(\log(n))$?



```
void memmove(char *dst, char *src, int size);

void insertionSortFast(int *a, int n) {
    for (int i = 1; i < n; ++i) {
        int new_pos = binary_search(a, i, a[i]);

        if (new_pos < i) {
            int tmp = a[i];
            memmove(&a[new_pos + 1], &a[new_pos],
                (i - new_pos)*sizeof(int));

            a[new_pos] = tmp;
        }
    }
}
```


Сортировка вставками. $n(\log(n))$?



- Будем искать позицию вставки за время $O(\log(i - k))$
- $O(n \log(n))$, $\theta(n)$ - операций сравнения
- $O(n^2)$, $\theta(1)$ - операций перемещения

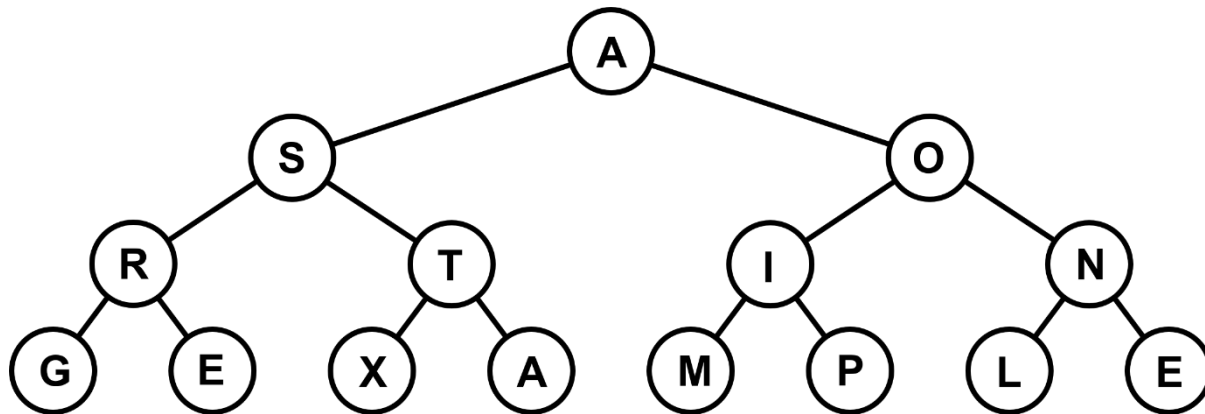
Хорошие сортировки



- Пирамидальная сортировка - Heap Sort
- Сортировка слиянием - Merge Sort
- Быстрая сортировка - Quick Sort

Пирамидальная сортировка

- N вставок в кучу: $N \cdot O(\log(N))$
- N Извлечение минимума из кучи: $N \cdot O(\log(n))$
- Построение кучи из N элементов: $O(2N \cdot \log(N))$



Пирамидальная сортировка .



```
void heap_insert(int *a, int n, int x)
{
    a[n+1] = x;
    for (int i = n+1; i > 1;) {
        if (a[i] > a[i/2]) {
            swap(a[i], a[i/2]);
            i = i/2;
        } else {
            break;
        }
    }
}
```

Пирамидальная сортировка ..



```
void heap_pop(int *a, int n) {  
    swap(a[n], a[1]);  
  
    for (int i = 1; 2*i < n;) {  
        i *= 2;  
        if (i+1 < n && a[i] < a[i+1]) {  
            i += 1;  
        }  
        if (a[i/2] < a[i]) {  
            swap(a[i/2], a[i]);  
        }  
    }  
}
```

Пирамидальная сортировка ..!



```
void heap_sort(int *data, int n) {  
    int *buff = new int[n+1];  
    for (int i = 0; i < n; ++i) {  
        heap_insert(buff, i, data[i]);  
    }  
    for (int i = 0; i < n; ++i) {  
        data[n-1-i] = buff[1];  
        heap_pop(buff, n - i);  
    }  
    delete [] buff;  
}
```

Пирамидальная сортировка



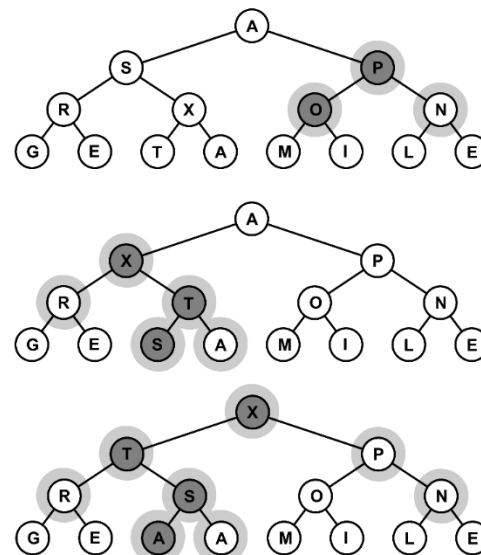
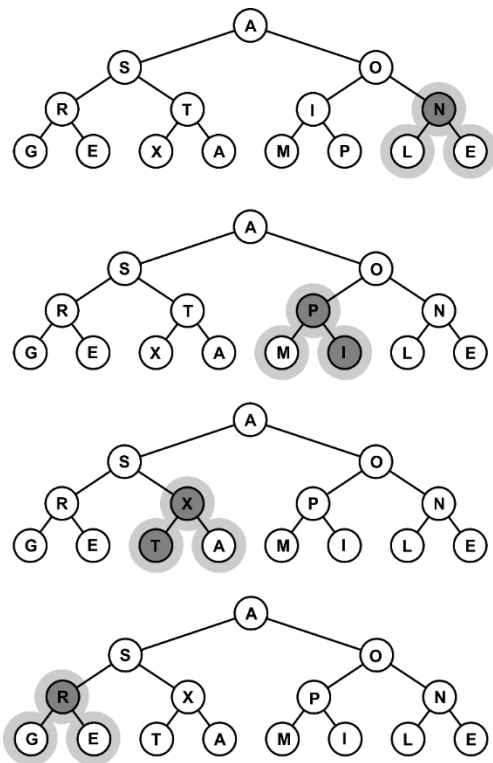
- Построить пирамиду за линейное время: $O(N)$
- N раз достать максимальный элемент: $O(N \cdot \log(N))$
- Не использовать дополнительную память

Пирамида за линейное время



- Для каждого внутреннего элемента восстановить порядок
- По 2 сравнения на уровень дерева
- $N/2 \cdot 2$

Пирамида за линейное время



Пирамида за линейное время



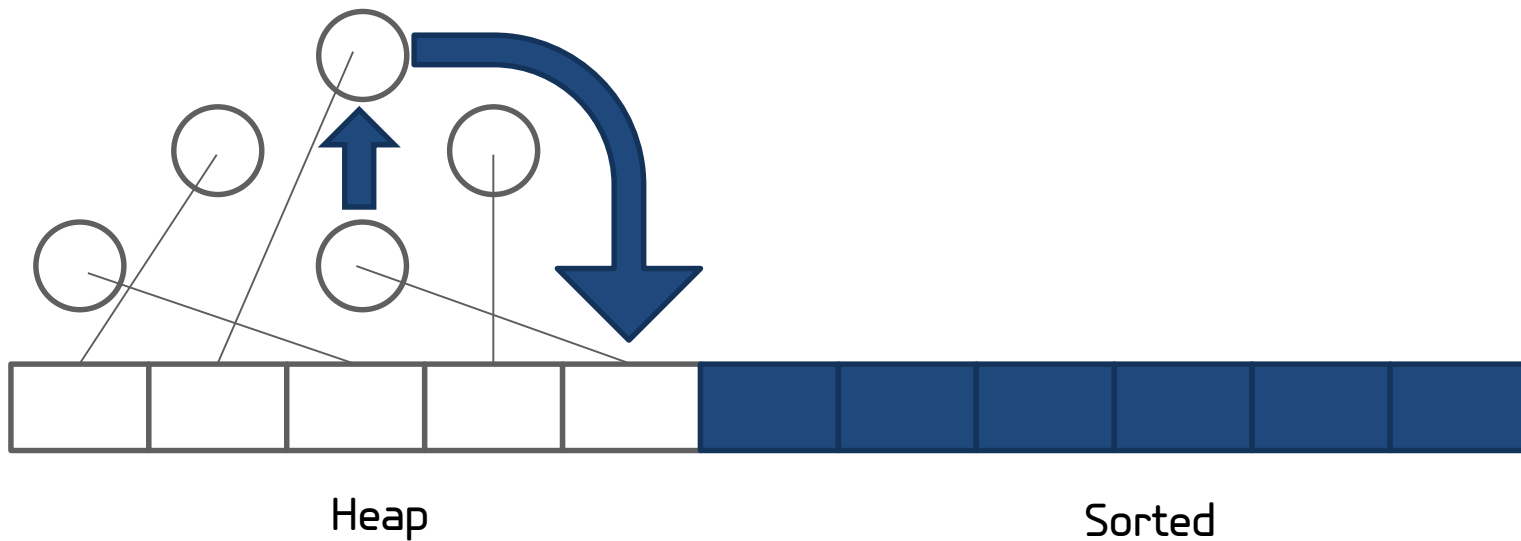
```
void heap_make(int *a, int n) {  
    for (int i = n/2; i >= 1; --i) {  
        for (int j = i; j <= n/2;) {  
            int k = j*2;  
            if (k+1 <= n and a[k] < a[k+1]) {  
                ++k;  
            }  
            if (a[j] < a[k]) {  
                swap(a[k], a[j]);  
                j = k;  
            } else {  
                break;  
            }  
        }  
    }  
}
```

Пирамида за линейное время



```
void heap_sort_fast(int *data, int n)
{
    heap_make(data - 1, n);
    for (int i = 0; i < n; ++i) {
        heap_pop(data - 1, n - i);
    }
}
```

Пирамидальная сортировка как эволюция сортировки выбором



Сортировка Хоара: QuickSort



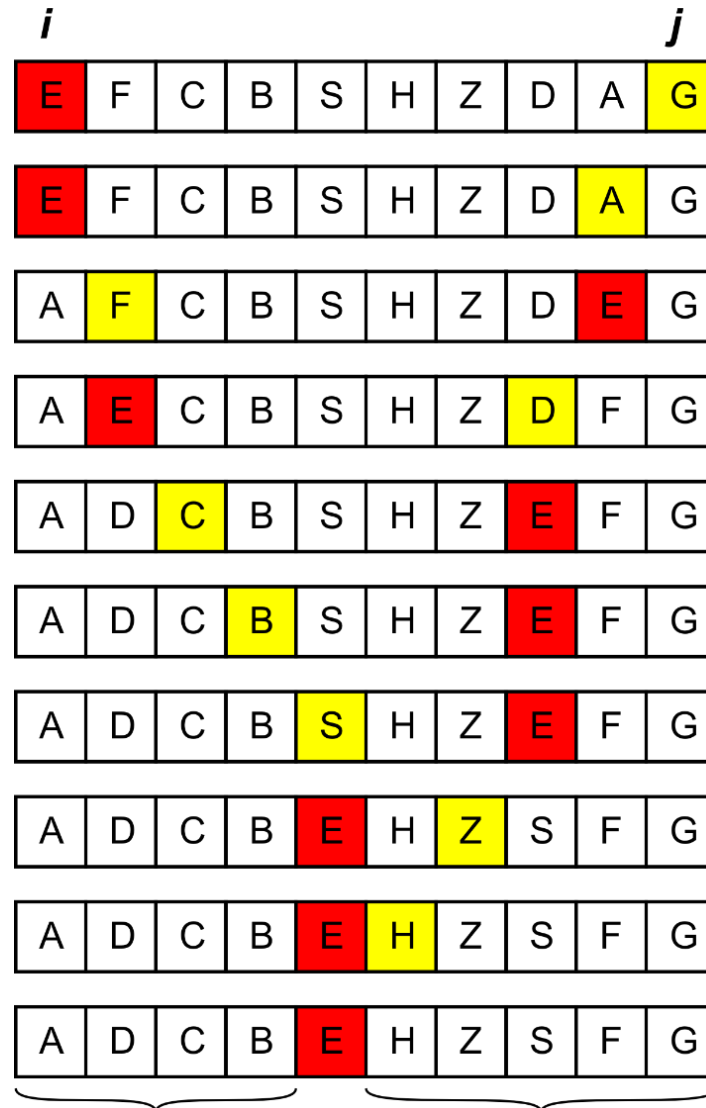
- Разделим массив на 2 части, так что элементы в левой меньше чем в правой
- Применим эту процедуру к левой и правой частям рекурсивно

QuickSort: Split



- Установим 2 указателя: i в – начало массива, j – в конец
- Будем помнить под каким указателем лежит «пивот»
- Если $a[j] > a[i]$, поменяем элементы массива под i, j
- Сместим на 1 указатель, не указывающий на «пивот»
- Продолжим пока $i \neq j$

QuickSort: Split



Сортировка Хоара

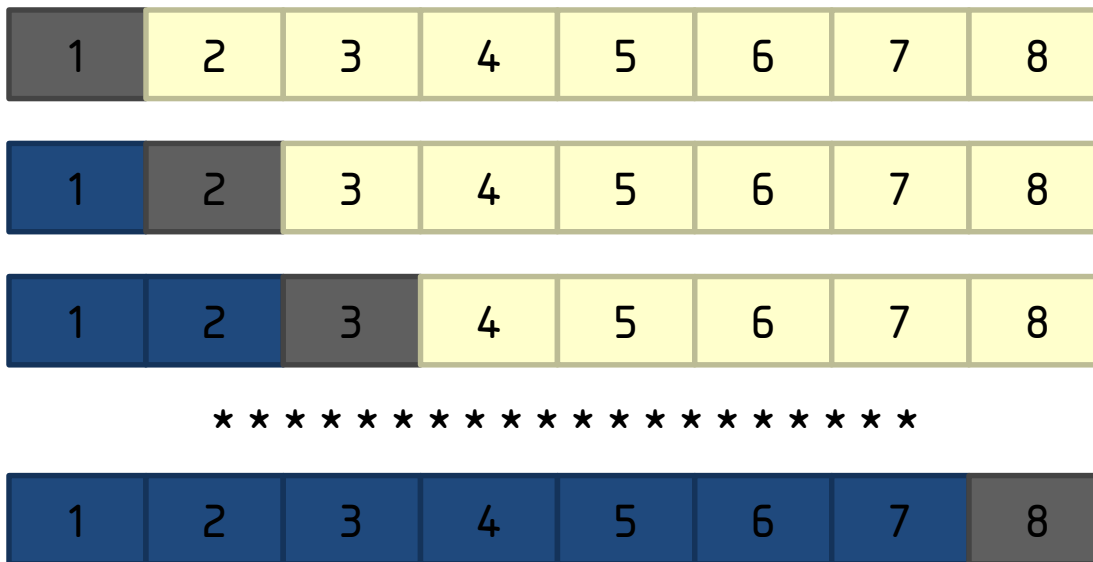


```
void quick_sort(int *a, int n) {  
    int i = 0;  
    int j = n - 1;  
    bool side = 0;  
    while (i != j) {  
        if (a[i] > a[j]) {  
            swap(a[i], a[j]);  
            side = !side;  
        }  
        if (side) {  
            ++i;  
        } else {  
            --j;  
        }  
    }  
    if (i > 1) quick_sort(a, i);  
    if (n > i+1) quick_sort(a + (i+1), n - (i+1));  
}
```


- *Предположим, что split делит массив в соотношении 1:1*
- $$T(n) \leq c_1 + c_2 n + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) =$$
$$= \sum_{k=0}^{\log_2 n} \{c_1 2^k + c_2 n\} = c_1 n + c_2 n \log(n)$$
- $T(n) = O(n \log(n))$

Quicksort: анализ

- *Упорядоченный массив делится в соотношении 1:n-1*
- $T(n) = c_1 + c_2n + T(n - 1) = \frac{1}{2}c_2n^2 + c_1n = O(n^2)$

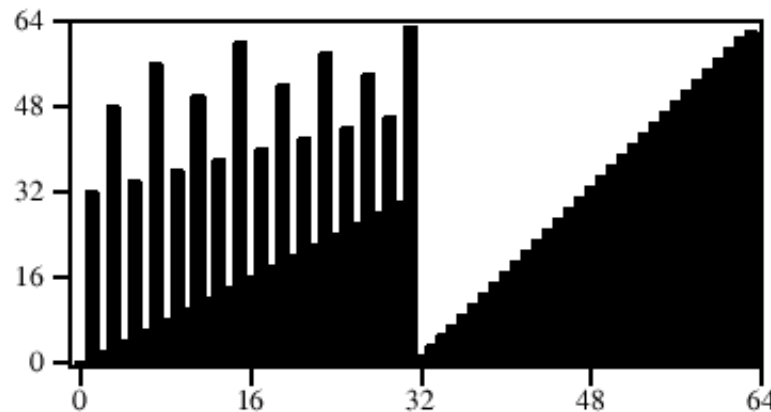


Quicksort: выбор пивота



- **1й элемент**
- Серединный элемент
- Медиана трёх
- Случайный элемент
- Медиана
- Медиана по трём случайным
- ...

- Последовательность, приводящая к времени: $T(n) = O(n^2)$
- $[1, 2, 3, \dots, n] \Leftrightarrow$ первый элемент
- Для любого predetermined порядка выбора пивота, существует **killer**-последовательность



- <http://www.cs.dartmouth.edu/~doug/mdmspe.pdf>

- **Локальная** сортировка сортировка не требует дополнительной памяти
- **Стабильная** сортировка сохраняет порядок следования равных элементов

Стабильность сортировки



- Постоянство порядка элементов с неразличимыми ключами
- Сортировка двузначных чисел по старшему разряду
 $[21, 22, 23, 24, 25, 19, 18, 17, 16] \Rightarrow [19, 18, 17, 16, 21, 22, 23, 24, 25]$

Algorithm	insertion	select	heap	quick	merge
stable	Yep	Yep	No	Possible	???

- Любую сортировку можно сделать стабильной, за $O(n)$ дополнительной памяти

Локальность сортировки



- Отсутствие необходимости во внешней памяти
- Тривиальные сортировки локальны
- Скорость – плата за локальность

Algorithm	insertion	select	heap	quick	merge
Inplace	$O(n^2)$	$O(n^2)$	$O(n \log(n))$	$O(n \log(n))??$	$O(n \log(n))??$
$O(\log(n))$				$O(n \log(n))$	
$O(n)$					$O(n \log(n))$

- http://akira.ruc.dk/~keld/teaching/algoritmedesign_f04/Artikler/04/Huang88.pdf <= inplace merge

- **К-й порядковой статистикой называется элемент, который окажется на k-й позиции после сортировки массива**

Медиана \Leftrightarrow $\text{RandSelect}(A[1, N], k); k = N/2$

- Случайно выберем элемент из A : $A[j]$
- Разобьём A на 2 части: меньше/больше $A[j]$
- Пусть позиция $A[j]$ в разделённом массиве: k
- $k < j$?
 - Найдём: $\text{RandSelect}(A[1, j], k)$
 - Иначе: $\text{RandSelect}(A[j+1, N], k-j)$

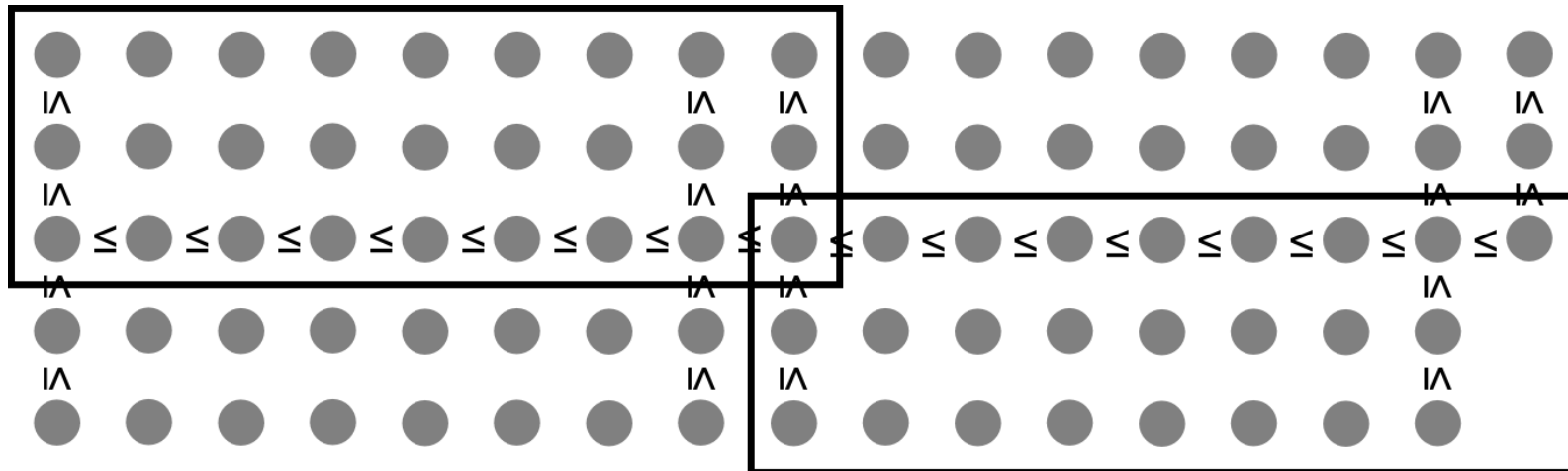
Quicksort: медиана за линейное время



Медиана \Leftrightarrow *SELECT*(*A*[1,*N*], *k*); $k = N/2$

- Разобьём массив на пятёрки
- Отсортируем каждую пятёрку
- Найдём медиану середин пятёрок
- Разобьём массив на 2 группы: меньше/больше медианы пятёрок
- Пусть индекс медианы в массиве j
- $j > k$?
 - найдём: *SELECT*(*A*[1, j], k)
 - иначе: *SELECT*(*A*[$j+1$, *N*], $k-j$)

Медиана за линейное время



Медиана за линейное время: анализ



- Разобьём массив на пятёрки
- Отсортируем каждую пятёрку $c_1 N$
- Найдём медиану середин пятёрок $T(N/5)$
- Разобьём массив на 2 группы \leftrightarrow медианы медиан: $c_2 N$
 - найдём: $SECTECT(A[1, j], k) \Rightarrow T(j)$
 - иначе: $SELECT(A[j+1, N], k-j) \Rightarrow T(N - j); 0.3N \leq j \leq 0.7N;$
- $T(N) \leq T\left(\frac{N}{5}\right) + cN + T(0.7N); \Rightarrow \mathbf{T(N) = O(N)};$



?

? *Вопросы* ?

?



Спасибо за внимание!

Иванов Георгий

Поиск.mail.ru