Сортиране, 27.10

# Термини

# Теоретични въпроси

## Въпрос 1

* Като знаем, че quicksort e in-place, каква е сложността относно памет?
* C\*log(n), C\* е заради избирането на pivot, а паметта е заради stack frame-a на рекурсивната функция.

## Въпрос 2

* Pivot vs Shuffling.
* По-добре да шъфълнем масива. Bob Sedgewick

## Въпрос 3

* Възможно ли е quick sort да бъде stable
* Да, като се използва допълнителна памет.

## Въпрос 4

* Worst-Case при Duplicate Keys, QuickSort? (масив от еднакви елементи)
* O(n2)

# Quick sort

## Логика

* Имаме 10 ученика, искаме да ги наредим по височина, 1 е избран (пивот) и всички да се наредят на ляво от него и на дясно от него спрямо височината му. Повтаряме за лявата и дясната част на редицата.
* Нямаме нужда да сравняваме елементи от лявата и дясната половина на пивота.

## Time complexity

* Average O(nlogn)
* Worst O(n2) – лошо избираме пивот, и игнорираме едната част на масива.

## Space complexity

* O(1)

## In-place

* Да

## Number of comparisons

## Number of swaps

## Adaptive

* Anti-adaptive (колкото по-разбъркан, толкова по-добре, разчита на randomization)

## Stable

* Зависи от имплементацията.

## External

## Parallelism

* Има свойството паралелизъм.

## Online

* Не

## Merge vs Quick

* Памет
* Ако паметта не е проблем, quick sort-a пак е за предпочитане.
* Caching
* Cache locality
* Quick sort-а е cache friendly
* L1, L2, L3 Caching
* Локалност (елементите са близо един до друг в паметта)
* Масива при quick sort-а попада в L1 cache-a
* LRU policy (caching стратегия)
* Merge sort (cache missing)

## 3 way Quicksort

* Dutch flag problem

## Dual pivot quicksort

# Binary Search

## Логика

* Идва от монотонността на функцията.
* Монотонна функция.

## Adaptive

* Не

## In place

* Да защото паметта му е по малка от O(n)

## Space complexity

* O(logn) – рекурсивно
* O(1) – итеративно

## Time complexity

* O(logn)

# Exponential search

* Morris pratt
* Идеята е, че няма нужда да знаем дясната граница.
* Индекса върви като степени на 2ката.
* Адаптивно е, защото тръгва от началото и по-бързо ще намери някое число спрямо позицията му.

# Interpolation search

## Логика

* Приема, че имаме добра дистрибуция на числата
* 1 2 3 4 6 8 10 – бързо
* 1, 100, 1 000 000, 20 000 000 – бавно
* Ползваме числата на масива като scaling factor.

## Time complexity

* O(n) – worst case
* O(log(logn))

# Ternary Search

## Логика

* Ще помогне ако имаме унимодална функция
* /\ - функция