Сортиране, 19.10

# Термини

## Internal/External sort

* Ако имаме 4 ГБ RAM и масива ни заема 4 ГБ RAM не можем да ползваме merge sort.
* External – ако му подадем редица, която не можем да fit-нем в RAM-та може да я сортираме редицата бързо с помощта на харддиска.
* Може да разделим 4 ГБ масив на 2 части от по 2 ГБ. После записваме единия в хард диска, сортираме другия и аналогично сортираме и другия.
* После като имаме 2 сортирани масива в RAM паметта, може да ги merge-нем на хард диска.

## Parallelism

* на 1 нишка може да пуснем едната половина, на друга нишка може да пуснем другата половина

# Теоретични въпроси

## Въпрос 1

* Aко пуснем insertion sort за масив от еднакви елементи, каква ще е сложността.
* О(n)

## Въпрос 2

* Може ли merge sort-a да се напише итеративно.
* Да, всеки рекурсивен алгоритъм може да се напише итеративно.

## Въпрос 3

* Aко сте дизайнери на бъдещата версия на Java 11, какъв подход бихте избрали за вградения sort?
* Нито merge, нито quick sort.
* Плюс на merge – stable, worst case – nlogn
* Минус на merge – паметта.

# Insertion sort (метода на картоиграча)

## Логика

* Почваме с една карта, теглим карти и ги напасваме, там където им е мястото.
* Ключът към най-бързите алгоритми.
* Добре се държи за почти сортирани масиви.

## Time complexity

* O(n2)

## Space complexity

* O(1)

## In-place

* Да, не използва допълнителна памет.

## Number of comparisons

* O(n2)

## Number of swaps

* O(n2)

## Adaptive

* Да
* Има адаптация към данните, които му подаваме.

## Stable

* Да

## Online

* Да
* За линейно време като приемем, че имаме сортирана редица, вкарваме нов и я сортираме.
* Много важно свойство на insertion сорт-а.

# Merge sort

## Логика

* Два несортирани масива да ги merge-нем в 1 сортиран. Не можем.
* Бързо/лесно можем да merge-нем два сортирани масива в 1 сортиран.
* Няма как да merge-нем 2 масива без да заделяме памет за друг, нов.
* Всеки масив с 1 елемент, по дефиниция е сортиран.
* Получава се дървовидна структура.
* Сложността идва от logn на брой разбивания и линейното merge-ване за тези logn на брой стъпки.
* Insertion сортировката не се различава от merge sort-a до г/д 64 елемента по брой на операциите.
* При малък брой елементи, е хубаво да използваме insertion сорт.
* Когато разбием елементите в дървото до 64 елемента може да използваме insertion sort.
* Amazon ползват merge sort за намиране на инверсии за препоръка на продукти.
* Tim sort-a

## Time complexity

* O(n logn)

## Space complexity

* O(n)

## In-place

* Не, отнема допълнителна памет.

## Number of comparisons

* O(n logn)

## Number of swaps

* O(n)

## Adaptive

* Не

## Stable

* Да
* Много важно свойство на merge сорта.
* При обекти се предпочита merge пред quick сорт-а.
* Защо стабилността е толкова важна? – Не местим големи обекти без причина.

## External

* Да

## Parallelism

* Да

## Online

* Не

# Тim sort

## Логика

* Първо намиране всички намаляващи под редици и ги обръщаме, така при merge сорта когато разбием редиците на по-малко от 64, insertion сорта ще се справи много по-добре.