СДА

- Алгоритъм - последователност от стъпки за решаване на проблем
- Структура от данни - организиране на данни във формат удобен за работа
- Вид сложност - по време и по памет
- оценка на сложност - best, worst, average case + амортизирана при многократно извикване
- Big O - показва функцията, която се описва когато аргументът клони към или големи стойности
  - сложност : $f(x) = O(g(x))$
  - дефиниция : $\exists(N>0)\exists x_0 \forall(x>x_0)\left(|f(x)| \leq N_{g(x)}\right)$

Поредност -> $O(1)$, $O(\log N)$, $O(N)$, $O(N \log N)$, $O(N^2)$, $O(2^N)$, $O(N!)$, $O(N^N)$

// $O(1)$ - константна, $O(N)$ - линейна

$O(\log N)$ - означава, че на всяка стъпка махаме половината (или повече) от възможностите
- ако изхвърляме половината : $i^* = 2$, тогава базата на log е 2 => $\log_2 N$. Аналогично за $i^* = 5$ ще е $\log_5 N$, но обуо се пише $\log N$
- показваме worst case, махаме константи и по-малки събираеми

Упражнение

- Намиране на сложност
  - $O(1)$ - константна ако операциите не зависят от входа и нямa цикъл зависещ от входа

Пример: `func(int n){ n+=10; cout<<n; и тн }`
`{ for(int i=0; i<10; i++) n+=1; }`

  - $O(N)$ - линейна ако обхождаме всички данни в цикъл

Пример: `for(int i=0; i<N; i++) ...`

  - $O(N^2)$ - квадратична ако обхождаме всички 2 пъти

Пример: `for(i=0; i<N; i++){ for(j=0; j<N; j++) ... }`

  - $O(\log N)$ - логаритмична ако махаме част от операциите (половината поне)

Пример: `for(i=N; i>0; i/=2)` // $\log_2 N$, ако беше i/=3 щеше да е $\log_3 N$, но за нас е $\log N$ защото е с 2

  - $O(N \log N)$

Пример: `for(i=N, i>0, i--){ for(j=N, j>0, j/=2) ...}`

- Ако са отделни for не вложени се събира сложността тест остава колкото е най-сложния
  // for за N и for за N^2 е N^2
- Ако извикваме функ гледаме вътрешната сложност и тя е кол умножение цикъла (все едно е for цикъл) и нас нали остава само най-голямата без конст и събираемите
- Да внимавам ако вътрешния цикъл е до i да не е N вместо N2
- Ако функ извиква функ:
  `for(..cn){ func2(n); }` и func има `for(cn)`
  сиква for в for => n.n => n^2

Сортиращи методи

1 Bubble Sort

- идея: от началото сравняваме всеки 2 последователни елемента и ако левия е по-голям ги разменяме. В края най-големият елемент ще е на правилна позиция и така последователно ще "изплуват" най-големите.

2 Selection Sort

- идея: от началото търси най-малкия елемент и го поставя на правилна позиция. След това гледа за ind 1 и търси най-малкия от останалата част и го слага на правилна позиция и тн

3. Insertion Sort

- идея: от началото започва да прави малки сортирани участъци. Първо сортира 0 и 1 елем. След това 0,1,2 ел. После 0,1,2,3 ел и така малко по-малко сортира всичко като всеки следващ го insert-ва на правилната му позиция като отмества дрте по-големи надясно с 1 поз

Тези бяха бавни сортировки за $O(n^2)$

Допълнително

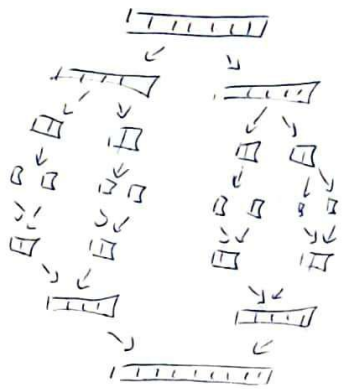
Radix Sort -> като count ама и за големи числа

1 гледа първо единиците и ги сортира по тях, после по десетици, стотици и тн

Бързи сортиращи алгоритми $O(n * \log n)$ mostly

1 Merge Sort

- идея: дели масива на 2. Всяка половина рекурсивно я дели на още 2, докато не останат единични елементи. След това по обратния ред слива (merge-ва) масивите и ги сортира:



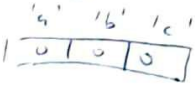
2 Quick Sort

- идея: имаме pivot (най-левия ел) и след него имаме две чертички (последния най-малък и текущия). Сравняваме и ако текущия е по-малък от pivot, то разменяме текущия и този след посл от най-малките накрен. Така в края всички по-малки и всички по-големи ще са разделени и на мястото на последния най-малък ще разменим pivot. Така pivot е на правилното място

3 Counting Sort

- идея: имаме масив с капацитет макс бр разл. ел. Броим всяко срещане на всеки ел като увеличаваме съответстващата му стойност в масива ни с 1

Пр! "aabcacb"



→

→ "aaabbcc

Характеристики на алгоритми за сортиране

1 In-place

- не се нуждае от допълнително памет, освен малко за променливи (разрешено е всичко $\leq n$) най-често $\log n$
- in-place ако swap-ваш, не е in-place ако имаш нов arr[n]

2 Stability

- ако има равни елементи, то в сортирания масив, тези два равни елемента да са в същата последователност както са били в оригиналния

3 Complexity

- time
- space (memory)