Алгоритми

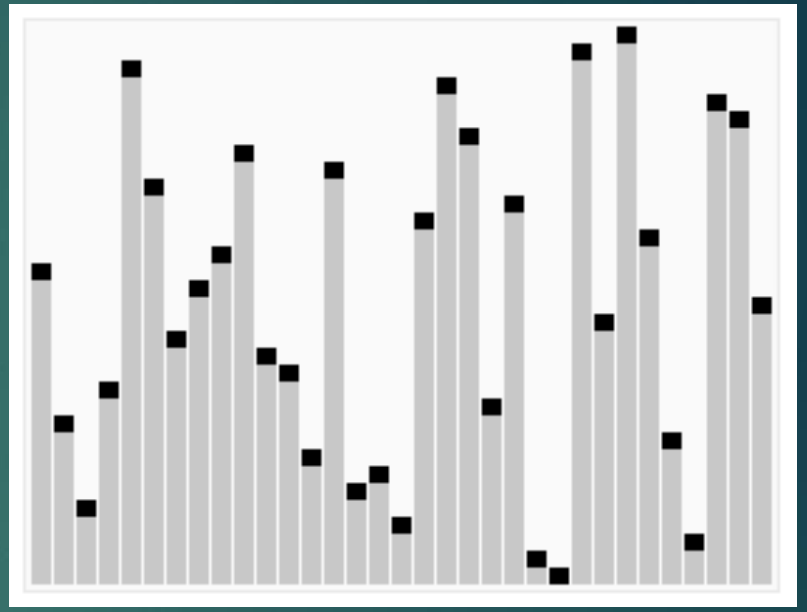
SAP GEEKYCAMP 4.0

Божин Кацарски
6.09.17

Сортиране

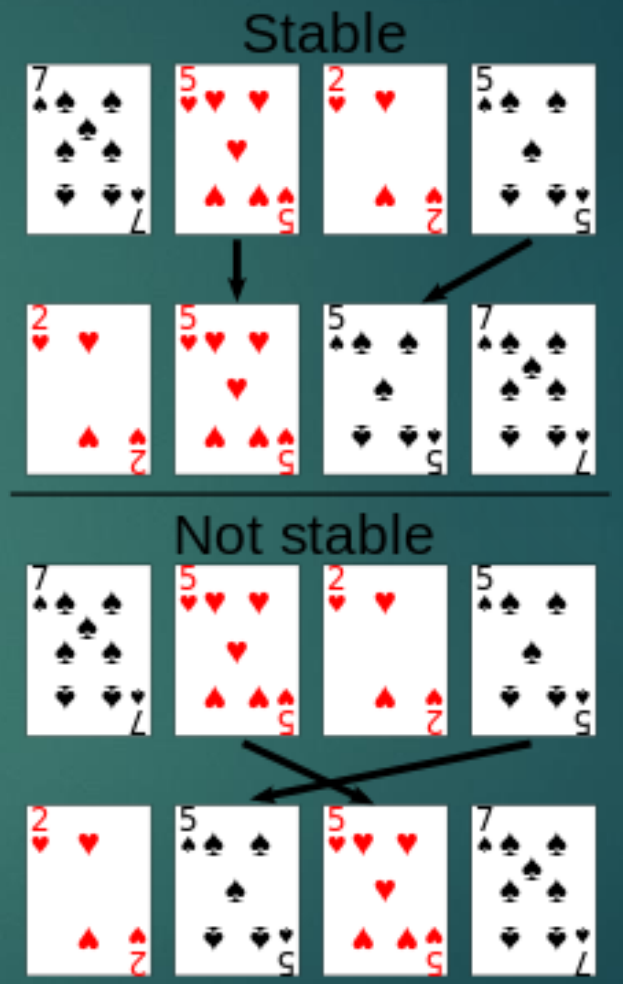
Сортиране

- Основна задача в програмирането
  - Вход: масив от сравними елементи
  - Задача: подреждане на елементите в нарастващ ред
  - Изход: -
- Много различни алгоритми
  - С различни плюсове и минуси
  - Някои подходящи само за учебна цел



Сортиране

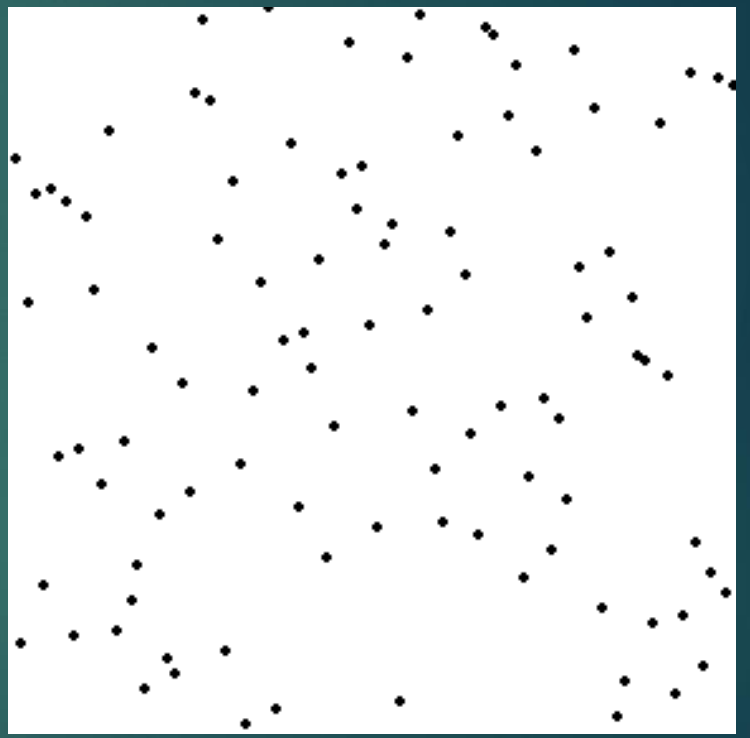
- Сортиране на място (in-place)
  - Изисква константна допълнителна памет
- Стабилно сортиране (stable)
  - Запазва относителния ред на елементите, които считаме за равни



selectionSort

сортиране чрез пряка селекция

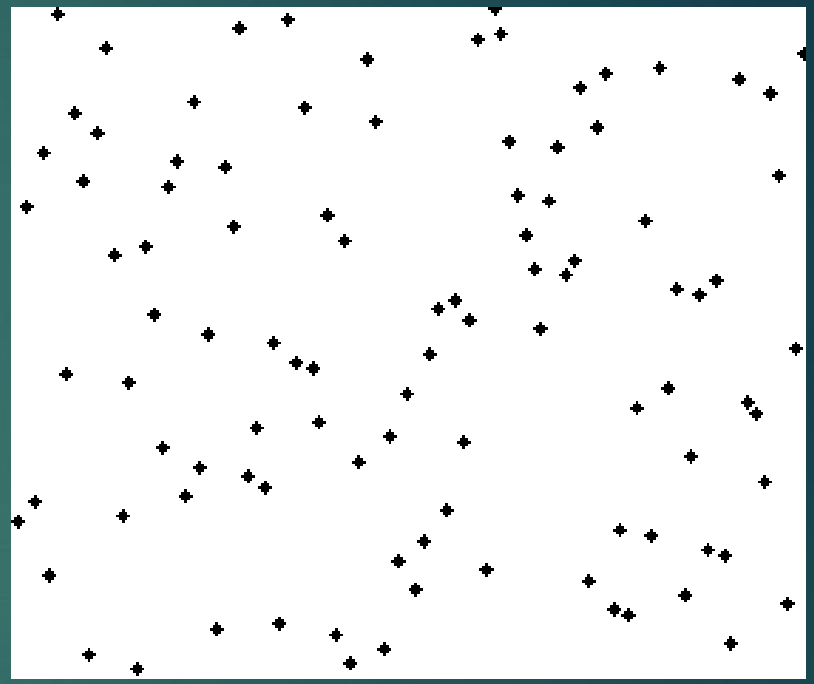
- Алгоритъм:
  - Прави n итерации върху масива
  - На i-тата от тях избира i-ят най-малък елемент
  - Разменя го с този, който в момента е на i-та позиция
- Сортира
  - стабилно
  - на място
- Лесен за имплементация
- Бавен
  - Каква е сложността му?
- Извършва минимален брой размени



mergeSort

сортиране чрез сливане

- Алгоритъм:
  - Разделя масива на две половини
  - Сортира всяка половина рекурсивно
  - Слива сортираните половини
- `*Разделяй и владей*` подход
- Стабилна сортировка
- Използва $O(n)$ допълнително памет*
  - Има вариант, който сортира на място, но е прекалено сложен
- Бърз
  - Каква е сложността му?
- Подходящ за паралелна обработка

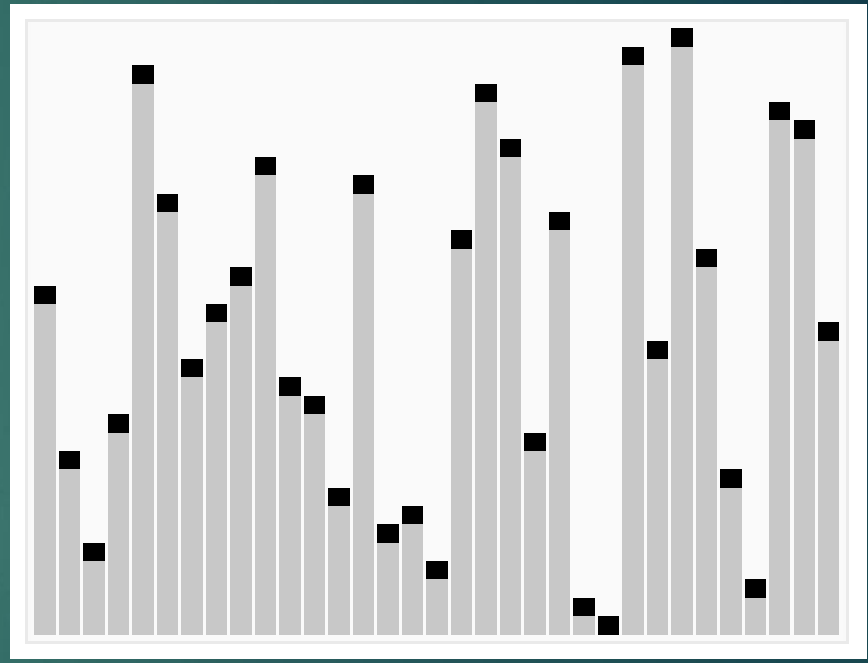


quickSort

бързо сортиране

- Алгоритъм:
  - Избираме *някой* елемент за pivot
  - Разместваме елементите така че:
    - По-малките от pivot са вляво от него
    - По-големите от pivot са вдясно от него
  - Сортираме двете части рекурсивно
- Нестабилна сортировка
- Сортира на място т.е. $O(1)$ допълнително памет
- Бърз в средния случай*
  - Каква е сложността му?
  - Кой е лошият случай?
- Най-добрият* сортиращ алгоритъм

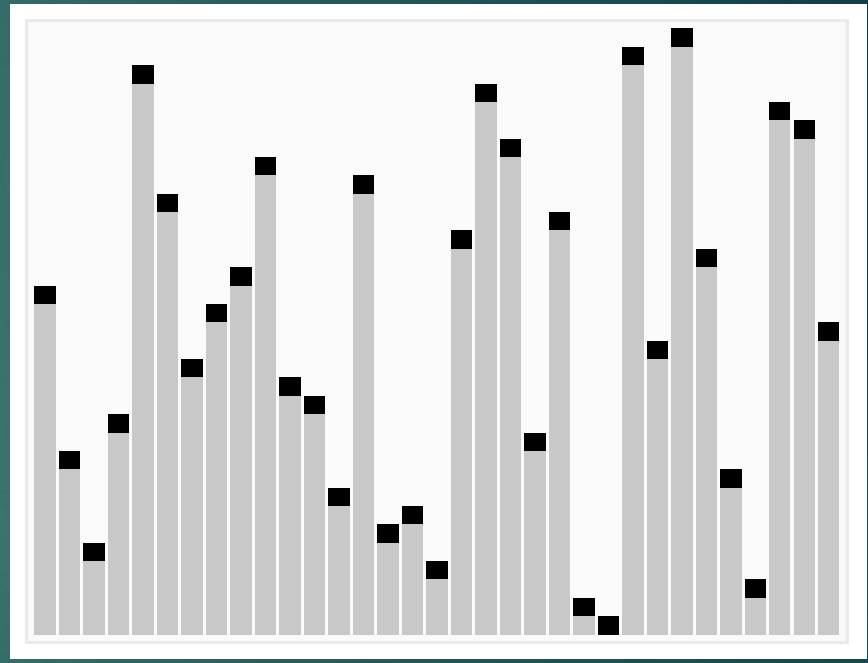
* в комбинация с друг алгоритъм за базовите случаи



quickSort – как избираме pivot

бързо сортиране

- Медианата на масива
- Най-левият елемент
- Елементът в средата
- Произволен елемент
- Медиана от 3 елемента

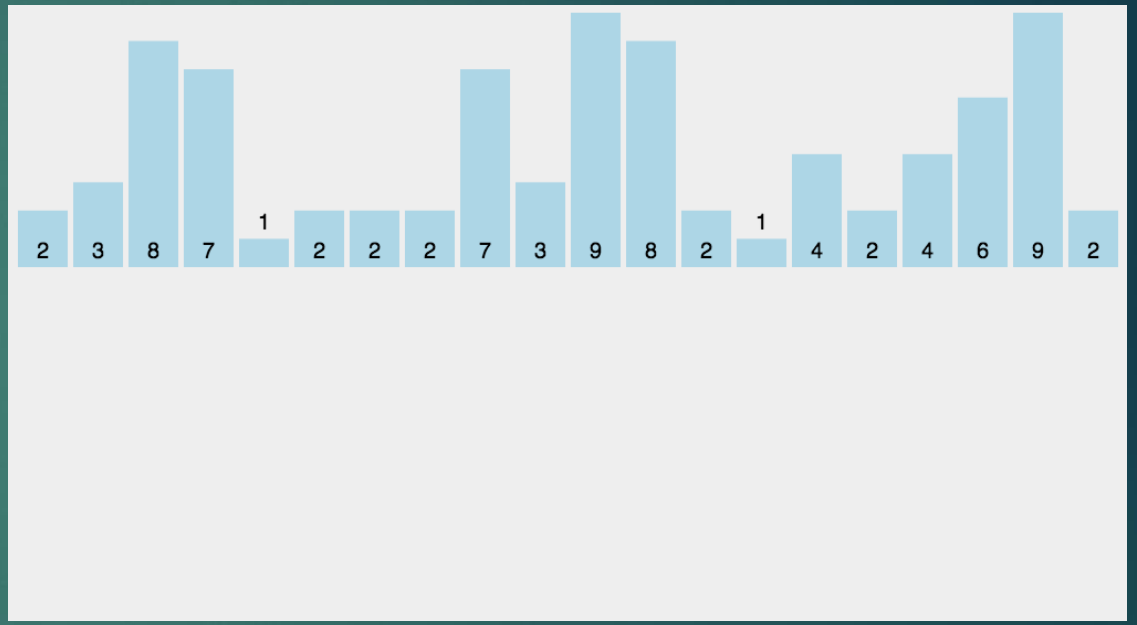


countingSort

сортиране чрез броене

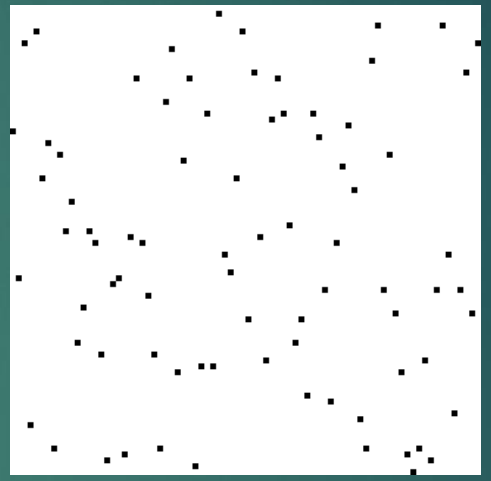
- За разлика от другите, не е базиран на сравнения
- Приложим, когато елементите за сортировка са в ограничен* интервал

  * Искаме разликата между най-големия и най-малкия елемент да е <n
- $O(n)$ време
- $O(n)$ допълнително памет

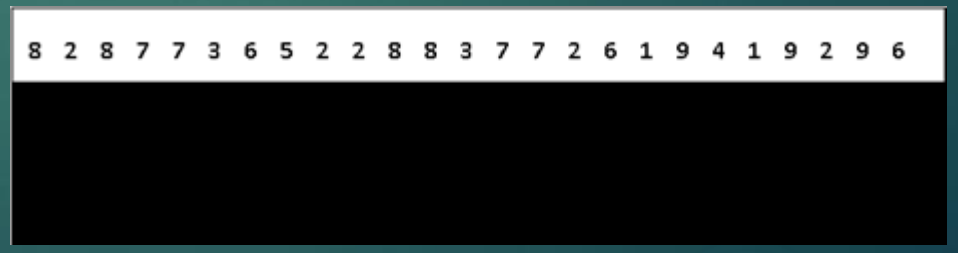


Безполезни, но забавни сортировки

- bogoSort



- sleepSort



Търсене

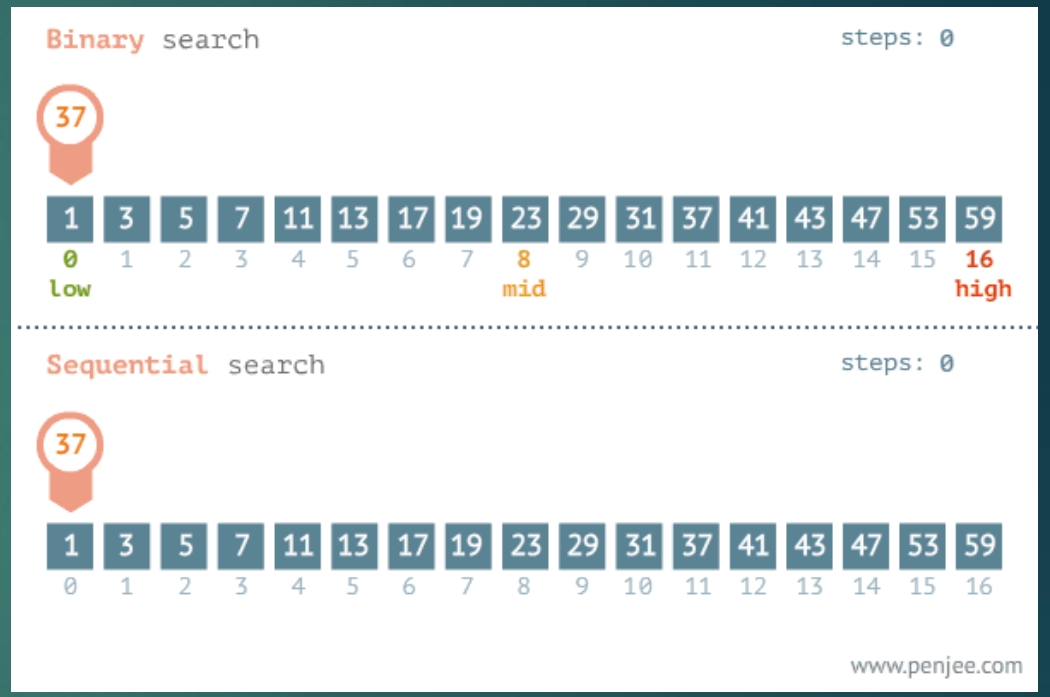
Линейно търсене

- Задачата:
  - вход:
    - Линейна последователност от елементи
    - Елемент, който търсим
  - изход: последователността съдържа ли елемента?

- Очевидното решение – линейно търсене:
  - обхождаме всички елементи и проверяваме дали са равни на търсения
  - $O(n)$ време

Двоично търсене
binary search

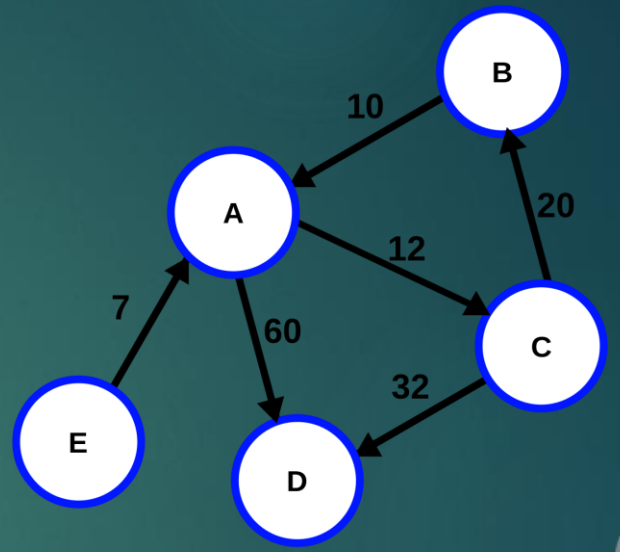
- Задачата:
  - вход:
    - Линейна сортирана последователност от елементи
    - Елемент, който търсим
  - изход: последователността съдържа ли елемента?
- Решението - двоично търсене:
  - Елементът в средата по-голям ли е от търсения?
  - Да – тогава търси рекурсивно в лявата половина
  - Не – тогава търси рекурсивно в дясната половина
- Каква е сложността по време?

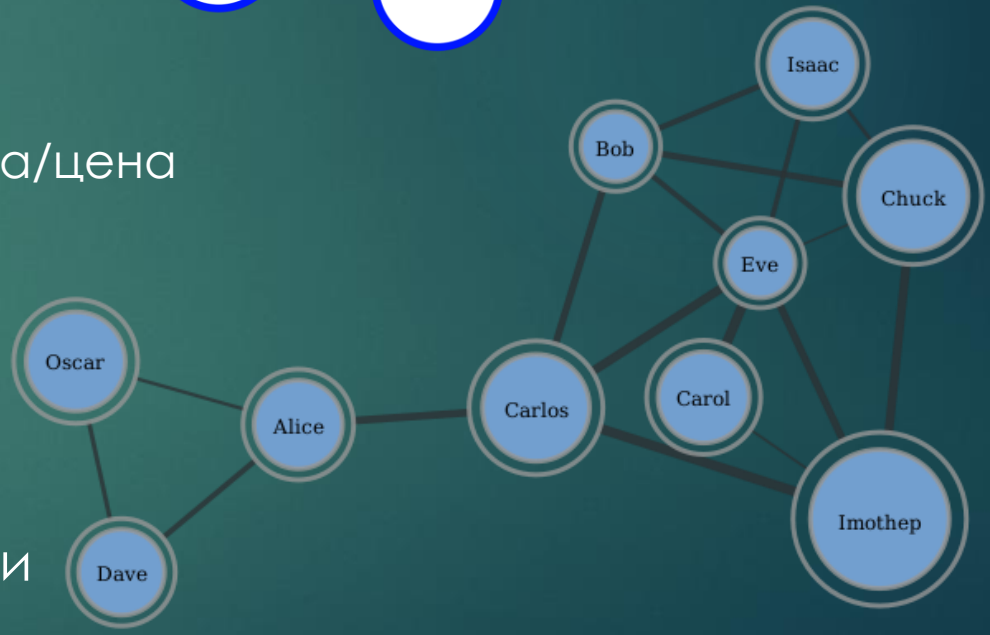


Графи

Какво е граф

- Множество от върхове (vertices / nodes)
- Множество от ребра (edges), свързващи някои върхове
  - Насочен граф – ребрата имат посока
  - Тегловен граф – ребрата имат тегло/дължина/цена
- С графи могат да се моделират
  - географски карти
  - компютърни мрежи
  - състояния на някаква система
- Съществуват много и най-различни задачи и алгоритми, свързани с графи





Как представяме граф в паметта

- Ако върховете се идентифицират с нещо, различно от числата 0, …, n
  - map<IdentifierType, int> vertices;
  - Защото ще ползваме номерата на върховете като индекси в масив
- Ребрата можем да представим като:
  - Матрица на съседство
  - Списък на съседите
  - Списък на ребрата