Bit Algo
START

Sortowania liniowe

Złożoność sortowania:

sortowanie	złożoność średnia	złożoność pesymistyczna	stabilność
quicksort	O(nlogn)	$O(n^2)$	NIE
merge sort	O(nlogn)	O(nlogn)	TAK
heap sort	O(nlogn)	O(nlogn)	NIE
count sort	O(n + k)	O(n + k)	TAK
radix sort	O(d(n+k))	O(d(n+k))	TAK
bucket sort	O(n)	$O(n^2)$	TAK

Po co to komu?

- **szybkie** - no lepiej niż O(n) się nie da
- **stabilne** - jak potrzebujemy stabilności, np. “stackujemy” sortowania (najpierw sortowanie po nazwisku, a potem po imieniu)
- **wykorzystują wiedzę o danych** - kiedy wiemy, że klucze (to, co sortujemy) są z niewielkiego zakresu i/lub mają rozkład jednostajny
- **możemy zaprojektować system tak, żeby ich używać** - decydując się na wybór formatu ID, kluczy itp. możemy z nich zrobić stringi, binary stringi lub integery z ograniczonego zakresu i z rozkładem jednostajnym (podobna długość, zakres znaków) i je później szybko sortować

Counting sort

- **wymaganie:** zakres wartości w tablicy **ograniczony** do liczb naturalnych [0, k]
- **złożoność:** Θ(k+n), dla k=O(n) daje to Θ(n) -> k musi być proporcjonalne do rozmiaru tablicy!
- **stabilny**
- **idea:** nie porównujemy elementów - zliczamy, ile jest elementów o wartości 0, 1, 2, …, k oraz później wrzucamy je do wynikowej tablicy

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Counting sort

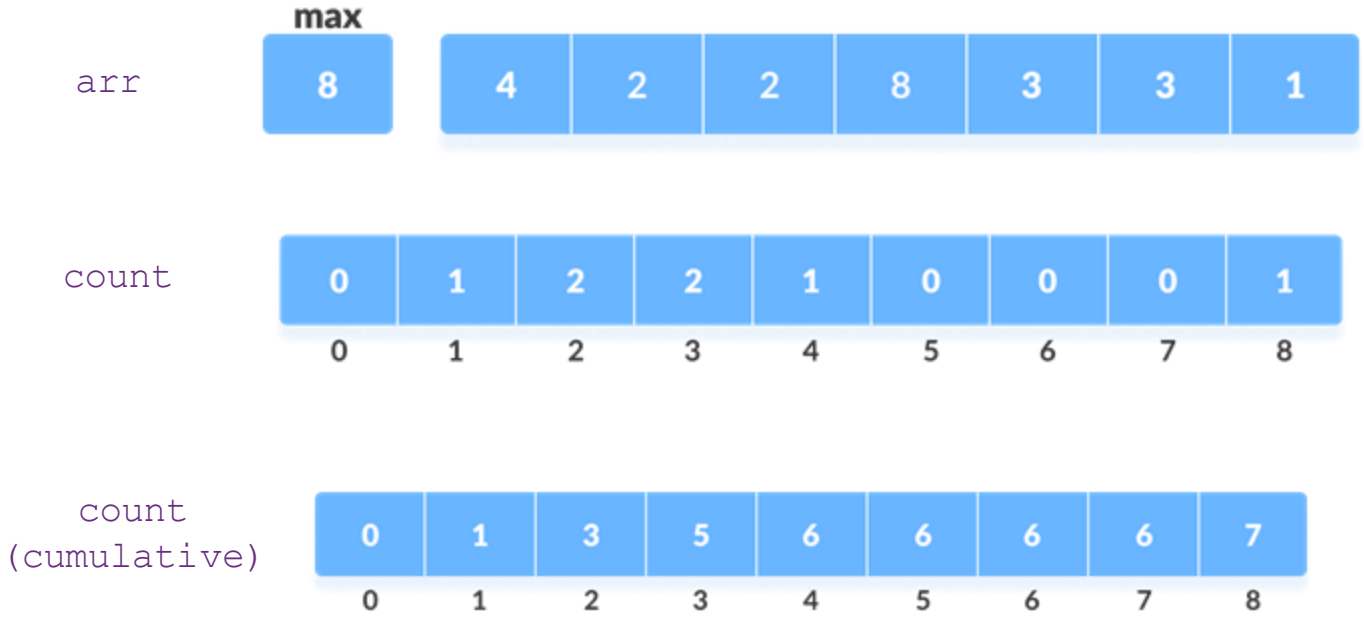
`A - arr, B - output, C - count`

0. Wyznaczamy k: `k = max(arr)`
   1. Robimy tablicę pomocniczą `count` z k zerami - w niej będziemy zliczać wystąpienia 0 (pod indeksem 0), 1 (pod indeksem 1), …, k (pod indeksem k)
   2. Zliczamy elementy - iterujemy po naszej tablicy `arr` i inkrementujemy odpowiednią wartość w `count`
   3. Obliczamy **cumulative sum** - dla każdej komórki w `count` dodajemy sumę wartości poprzednich
   4. Robimy tablicę wynikową `output` tak dużą, jak `arr`
   5. Iterujemy **od końca** (stabilność!) po `arr` i wykonujemy

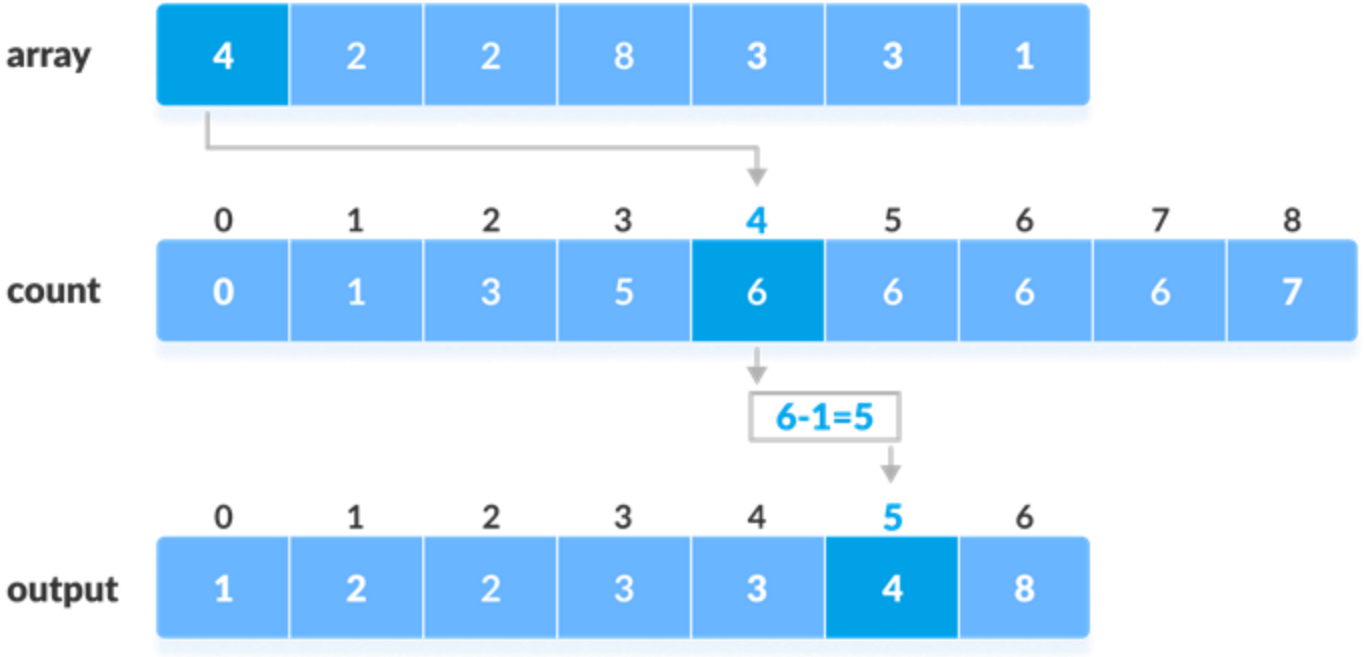
```
output[count[arr[j]] - 1] = arr[j]; count[arr[j]] -= 1
```

- `arr[j]` - wartość elementu w j-tej komórce tablicy wejściowej
- `count[arr[j]]` - tyle “sztuk” elementu o tej wartości nam zostało (dlatego po wstawieniu -1)
- `output[count[arr[j]] - 1]` - “wrzucamy” nasz element na to miejsce; -1, bo indeksujemy od zera (Cormen indeksuje od 1)

Counting sort



Counting sort



Counting sort

```
def counting_sort(arr):
    n = len(arr)
    k = max(arr)
    output = [0] * n
    counts = [0] * k

    for i in range(n):  # count
        count[arr[i]] += 1

    for i in range(k):  # cumulative sum
        count[i] += count[i - 1]

    for i in range(n - 1, -1, -1):  # place elements in output
        output[count[arr[i]] - 1] = arr[i]
        count[arr[i]] -= 1

    return output
```

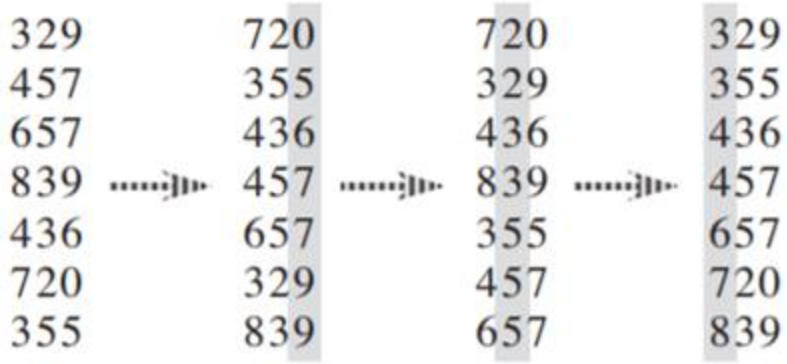
Radix sort

```
RADIX-SORT(A, d)
1  for i = 1 to d
2      use a stable sort to sort array A on digit i
```

- **wymagania:**
  - dane muszą dać się posortować leksykograficznie, np. integery, stringi
  - dane podobne na pierwszych pozycjach (np. 999991, 999992, 99999)
  - najlepiej, żeby były tej samej długości (albo przynajmniej podobnej)
- **złożoność:**
  - `d` - liczba pozycji (np. cyfr), `k` - możliwe wartości na pozycjach (np. 10 dla liczb), `n` - liczba elementów do posortowania
  - typowo $\Theta(d(n+k))$, ogólnie $O(d*f(n, k))$ - $f(n, k)$ to złożoność **sortowania pomocniczego**
- **idea:** sortujemy liczby według kolejnych pozycji, zaczynając od **najmniej znaczącej (ostatniej)**, za pomocą sortowania pomocniczego
- sortowanie pomocnicze stabilne ⇔ radix sort **stabilny** (lub nie...)

Radix sort

```
RADIX-SORT(A, d)
1  for i = 1 to d
2      use a stable sort to sort array A on digit i
```



Bucket sort

```
BUCKET-SORT(A)
1  let B[0..n − 1] be a new array
2  n = A.length
3  for i = 0 to n − 1
4      make B[i] an empty list
5  for i = 1 to n
6      insert A[i] into list B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8      sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
```

- **wymagania:**
  - liczby z zakresu [0, 1] ([0, k] z **normalizacją**)
  - musimy znać liczbę kubełków n (przynajmniej w przybliżeniu)
  - najlepiej rozkład jednostajny
- **złożoność:**
  - O(n) - rozkład jednostajny, $O(n^2)$ - tablica z liczbami jednej wartości
  - im bardziej jednostajny rozkład, tym bardziej liniowe
- **idea:** robimy listę n “kubełków”, czyli zakresów długości 1/n, każdy reprezentujemy jako listę, a następnie “wrzucamy” nasze liczby do odpowiednich “kubełków”, sortujemy je i przepisujemy do tablicy wynikowej
- **normalizacja:** na potrzeby liczenia, do którego kubełka powinien trafić element dzielimy go przez największy element z tablicy - wtedy wszystkie będą w zakresie [0, 1]; wrzucamy do niego jednak **niezmieniony** element!

Bucket sort


11 9 21 8 17 19 13 1 24 12
Scatter
1
9 8
11 13 12
17 19
21 24
0-5
5-10
10-15
15-20
20-25
Sort
1
8 9
11 12 13
17 19
21 24
0-5
5-10
10-15
15-20
20-25
Gather
1 8 9 11 12 13 17 19 21 24

Bucket sort

```
def bucket_sort(arr, n):
    norm = max(arr)
    buckets = [[] for _ in range(n)] # list of n empty lists

    for num in arr:
        norm_num = num / norm  # normalized num
        buck_ind = int(n * norm_num)  # select bucket
        buckets[buck_ind].append(num)  # add num to bucket

    for i in range(n):
        buckets[i] = sorted(buckets[i])

    output = []
    for i in range(n):  # i-th bucket
        for j in range(len(bucket[i])):  # j-th element
            output.append(bucket[i][j])

    return output
```

Zadania

Zadanie 1

Mamy dane n punktów (x, y) w okręgu o promieniu k (liczba naturalna), tzn. $0 <= x^2 + y^2 <= k$, które są w nim równomiernie rozłożone, tzn. prawdopodobieństwo znalezienia punktu na danym obszarze jest proporcjonalne do pola tego obszaru.

Napisz algorytm, który w czasie $\Theta(n)$ posortuje punkty po ich odległości do punktu (0, 0), tzn.

$d = sqrt(x^2 + y^2)$.

Zadanie 2

Mamy daną tablicę stringów, gdzie suma długości wszystkich stringów daje n. Napisz algorytm, który posortuje tę tablicę w czasie O(n).

Można założyć, że stringi składają się wyłącznie z małych liter alfabetu łacińskiego.

Zadanie 3

Mamy daną tablicę A z n liczbami naturalnymi. Proszę zaproponować algorytm o złożoności O(n), który stwierdza, czy w tablicy ponad połowa elementów ma jednakową wartość.

Zadanie 4

Zaproponuj klasę reprezentującą strukturę danych, która w konstruktorze dostaje tablicę liczb naturalnych długości n o zakresie wartości [0, k]. Ma ona posiadać metodę count_num_in_range(a, b) - ma ona zwracać informację o tym, ile liczb w zakresie [a, b] było w tablicy, ma działać w czasie O(1). Można założyć, że zawsze a >= 1, b <= k.

Zadanie 5

Masz daną tablicę zawierającą n (n >= 11) liczb naturalnych w zakresie [0, k]. Zamieniono 10 liczb z tej tablicy na losowe liczby spoza tego zakresu (np. dużo większe lub ujemne). Napisz algorytm, który posortuje tablicę w czasie O(n).

Zadanie 6

Dana jest tablica z n liczbami całkowitymi. Zawiera ona bardzo dużo powtórzeń - co więcej, zaledwie O(log(n)) liczb jest unikatowe (reszta to powtórzenia). Napisz algorytm, który w czasie O(n*log(log(n))) posortuje taką tablicę.

Zadanie 7

Dana jest tablica zawierająca n liczb z zakresu [0...n^2-1]. Napisz algorytm, który posortuje taką tablicę w czasie O(n).

Zadanie 8

Dana jest klasa :

```
class Node:
        val = 0
        next = None
```

reprezentująca węzeł jednokierunkowego łańcucha odsyłaczowego, w którym wartości val poszczególnych węzłów zostały wygenerowane zgodnie z rozkładem jednostajnym na przedziale [a, b].

Napisz procedurę sort(first), która sortuje taką listę. Funkcja powinna być jak najszybsza.

Zadanie 9

Dane jest słowo będące tablicą n znaków z alfabetu E, o rozmiarze |E|. Dana jest również liczba k. Długość słowa wynosi co najmniej |E|^k. Zaproponuj algorytm, który zwróci najczęściej powtarzający się w tym słowie spójny podciąg o długości k. Algorytm ma działać w czasie O(n), wykorzystywać O(1) pamięci. Ponadto zawartość tablicy po wykonaniu algorytmu powinna pozostać niezmieniona.

Hint: zadanie jest trudne :)

Zadanie 10

Dana jest tablica A mająca n liczb naturalnych przyjmujących wartości z zakresu [0...n]. Proszę napisać algorytm znajdujący rozmiar największego podzbioru liczb z A, takiego, że ich GCD jest różny od 1. Algorytm powinien działać jak najszybciej.

Zadanie 11

Zaproponuj algorytm, który w czasie O(nlog(n)) posortuje stos o rozmiarze n. Dozwolone jest tylko wykorzystywanie operacji udostępnionych przez interfejs stosu: push(), pop(), top(), isEmpty(), oraz dodatkowych stosów.

Zadanie 12

Dany jest zawierający n wierzchołków wielokąt, niekoniecznie wypukły. Jest reprezentowany jako tablica par struktur:

class Point:

x

y

w której (p1, p2) oznacza, że obiekty p1 i p2 klasy Point są połączone odcinkiem. Dany jest również punkt q, leżący poza wielokątem. Zaimplementuj/zaproponuj algorytm, który wyznaczy jak należy poprowadzić półprostą, zaczynającą się w punkcie q, tak aby przecięła jak najwięcej odcinków wielokąta. Uwaga!: zakładamy, że jeśli punkt p jest wspólny dla dwóch odcinków, to prosta przechodząc przez ten punkt przecina oba. Algorytm powinien działać w czasie O(nlog(n)).

Bit Algo
START