Softverski algoritmi u sistemima automatskog upravljanja

# Algoritmi sortiranja

Danilo Kaćanski, Ivan Radman

# Bubble sort

## 🔁 Ideja algoritma:

- U svakom prolasku **poredi dva susedna elementa**
- Ako su u pogrešnom redosledu → **zameni mesta**
- Nastavlja se dok **ceo niz ne postane sortiran**

8 5 3 1 4 7 9

```
def bubbleSort(niz):
    for i in range(len(niz)):
        for j in range(0, len(niz) - i - 1):
            if niz[j] > niz[j + 1]:
                niz[j], niz[j + 1] = niz[j + 1], niz[j]
```

# Bubble sort

## 🔁 Ideja algoritma:

- U svakom prolasku **poredi dva susedna elementa**
- Ako su u pogrešnom redosledu → **zameni mesta**
- Nastavlja se dok **ceo niz ne postane sortiran**

## 📉 Nema optimizacije:

- Čak i ako je niz **već sortiran** → i dalje se vrši sveukupan broj poređenja

🧠 Jednostavan ali **neefikasan za veće nizove**

## 🧮 Kompleksnost:

- **Najgori slučaj:** $O(n^2)$
- **Najbolji slučaj:** $O(n^2)$ (*uvek radi sve itaracije*)

```python
def bubbleSort(niz):
    for i in range(len(niz)):
        for j in range(0, len(niz) - i - 1):
            if niz[j] > niz[j + 1]:
                niz[j], niz[j + 1] = niz[j + 1], niz[j]
```

8 5 3 1 4 7 9

# Bubble sort — Optimizovana verzija

## 🔍 Problem:

- Ako je niz **već sortiran** → nepotrebno prolazi kroz niz

## 🛠️ Rešenje:

- Uvesti **indikator**
- Ako se u jednom prolazu ne izvrši **ni jedna** izmena:
  - **Niz je sortiran**
  - **Prekini izvršavanje**

## 📊 Kompleksnost:

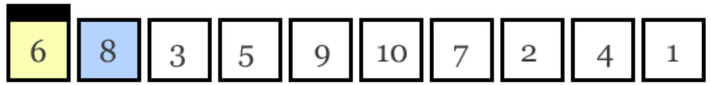
- **Najgori slučaj:** $O(n^2)$
- **Najbolji slučaj:** $O(n)$ (*niz već sortiran*)

```python
def bubbleSortOptimized(niz):
    for i in range(len(niz)):
        izvrsenaIzmena = False
        for j in range(0, len(niz) - i - 1):
            if niz[j] > niz[j + 1]:
                niz[j], niz[j + 1] = niz[j + 1], niz[j]
                izvrsenaIzmena = True
        if not izvrsenaIzmena:
            break
```

# Selection sort

## 🧠 Ideja algoritma:

- U svakom prolasku kroz niz:
  - Nađe se **najmanji element**
  - Zameni se sa **prvim nesortiranim elemntom**
- Ponavlja se dok ceo niz ne postane sortiran




Yellow is smallest number found
Blue is current item
Green is sorted list

```
def selectionSort(niz):
    for index in range(len(niz)):
        min_index = index
        for j in range(index + 1, len(niz)):
            if niz[j] < niz[min_index]:
                min_index = j
        niz[index], niz[min_index] = niz[min_index], niz[index]
```

# Selection sort

## 🧠 Ideja algoritma:

- U svakom prolasku kroz niz:
  - Nađe se **najmanji element**
  - Zameni se sa **prvim nesortiranim elemntom**
- Ponavlja se dok ceo niz ne postane sortiran

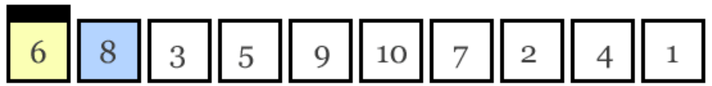
## ⚠️ Nema optimizacije:

- Uvek se **traži minimum** do kraja niza, bez obzira na stanje elemenata
- Broj **poređenja** je uvek isti, bez obzira na to da li je niz skoro sortiran

## 🧮 Kompleksnost:

- **Najgori slučaj:** $O(n^2)$
- **Najbolji slučaj:** $O(n^2)$ (*i dalje se traži minimum*)

```python
def selectionSort(niz):
    for index in range(len(niz)):
        min_index = index
        for j in range(index + 1, len(niz)):
            if niz[j] < niz[min_index]:
                min_index = j
        niz[index], niz[min_index] = niz[min_index], niz[index]
```




Yellow is smallest number found
Blue is current item
Green is sorted list

# Insertion sort

**Ideja algoritma:**

- Poseća na **ređanje karata u ruci**
- U svakom koraku:
  - Uzima se **sledeći element** iz nesortiranog dela
  - Proedi se unazad i **ubacuje odgovarajuće mesto** u sortiranom delu niza
- Ponavlja se dok se ceo niz ne sortira


```
def insertionSort(niz):
    for i in range(1, len(niz)):
        trenutni = niz[i]
        j = i - 1
        while j >= 0 and niz[j] > trenutni:
            niz[j + 1] = niz[j]
            j -= 1
        niz[j + 1] = trenutni
```

# Insertion sort

## 🃏 Ideja algoritma:

- Poseća na **ređanje karata u ruci**
- U svakom koraku:
  - Uzima se **sledeći element** iz nesortiranog dela
  - Poredi se unazad i **ubacuje odgovarajuće mesto** u sortiranom delu niza
- Ponavlja se dok se ceo niz ne sortira

```
def insertionSort(niz):
    for i in range(1, len(niz)):
        trenutni = niz[i]
        j = i - 1
        while j ⩾ 0 and niz[j] > trenutni:
            niz[j + 1] = niz[j]
            j -= 1
        niz[j + 1] = trenutni
```

## ⚠️ Nema optimizacije:

- Uvek se vrši **provera unazad** bez obzira da li je niz skoro sortiran
- Efiaksan za male ili skoro sortirane nizove, ali **neefikasan za obrnut redosled**

## 🧰 Kompleksnost:

- **Najgori slučaj:** $O(n^2)$ (*niz je obrnuto sortiran*)
- **Najbolji slučaj:** $O(n^2)$ (*niz sortiran kako treba*)

6 5 3 1 8 7 2 4

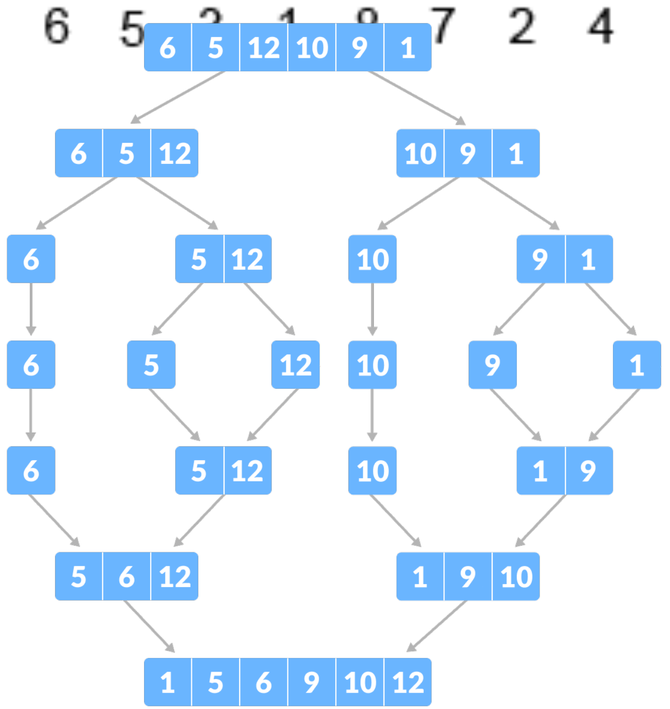
# Merge sort

## 🧠 Ideja algoritma:

- Niz se **rekurzivno deli** na manje podnizove sve dok svaki ne sadrži samo **jedan element**
- Zatim se ti podnizovi **spajaju**, pri čemu se **elementi porede** i postavljaju na odgovarajuće mesto
- Na kraju dobijamo **jedan veliki**, **sortiran niz**

## 📉 Kompleksnost:

- **Najbolji = Prosečan = Najgori slučaj =** $O(n * log\, n)$



# Merge sort — Koraci algoritma

### 1. Podeli niz na dva dela:

```
sredina = len(niz) // 2
L = niz[:sredina]
R = niz[sredina:]
```

### 2. Pozovi rekurzivno `mergeSort` na `L` I `R`:

```
mergeSort(L)
mergeSort(R)
```

### 3. Spoji dva sortirana niza u jedan:

```
i = j = k = 0
while i < len(L) and j < len(R):
    if L[i] < R[j]:
        niz[k] = L[i]
        i += 1
    else:
        niz[k] = R[j]
        j += 1
    k += 1
```

### 4.Dodaj preostale elemente:

```
while i < len(L):
    niz[k] = L[i]
    i += 1
    k += 1

while j < len(R):
    niz[k] = R[j]
    j += 1
    k += 1
```

# Merge sort — Zajedno

```python
def mergeSort(niz):
    if len(niz) > 1:
        sredina = len(niz) // 2
        L = niz[:sredina]
        R = niz[sredina:]

        mergeSort(L)
        mergeSort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                niz[k] = L[i]
                i += 1
            else:
                niz[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            niz[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            niz[k] = R[j]
            j += 1
            k += 1
```

✅ **Pogodan za paralelizaciju** — nazivsne rekurzije na podnizovima

✅ **Stabilan algoritam** — ne menja redosled elemenata koji su jednaki

✅ **Deterministički** — vreme izvršavanja ne zavisi od redosleda elemenata

✅ **Efikasan za rad sa velikim strukturama** — pogodan za eksterno sortiranje (*fajlovi na disku*)

🔁 Postoji više varijanti:

- Top-down merge sort (*klasičan rekurzivni*)
- Bottom-up merge sort (*iterativni*)
- Natural merge sort (*iskorišćava već sortirane delove*)
- Ping-pong merge sort (*dva niza koji se “prelivaju”*)

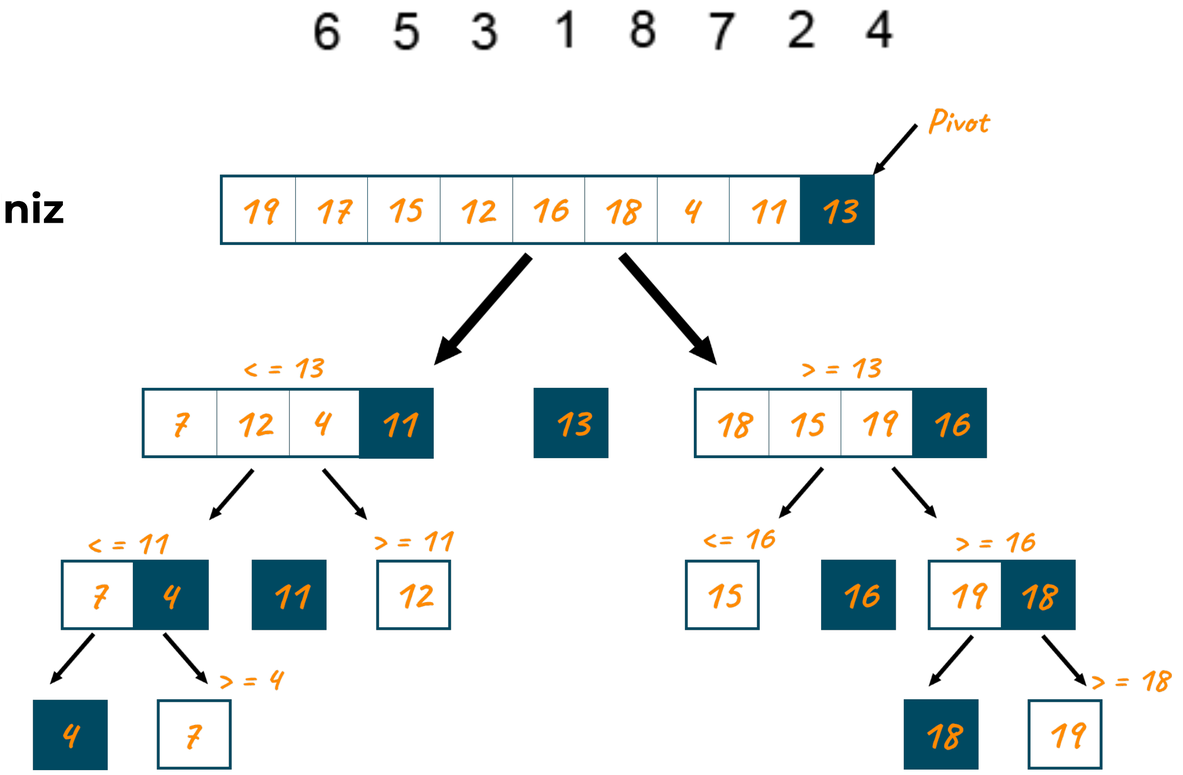
# Quick sort

## 🧠 Ideja algoritma:

- Odavrati **pivot** element
- Podeliti niz u dva dela:
  - **Levi deo:** elementi **manji od pivota**
  - **Desni deo:** elementi **veći od pivota**
- Rekurzivno primeniit isti postupak na **svaki podniz**

## ⚙️ Strategije za isbor pivota:

- **Prvi element** niza
- **Poslednji element** niza
- **Nasumični element**
- **Median-of-three**



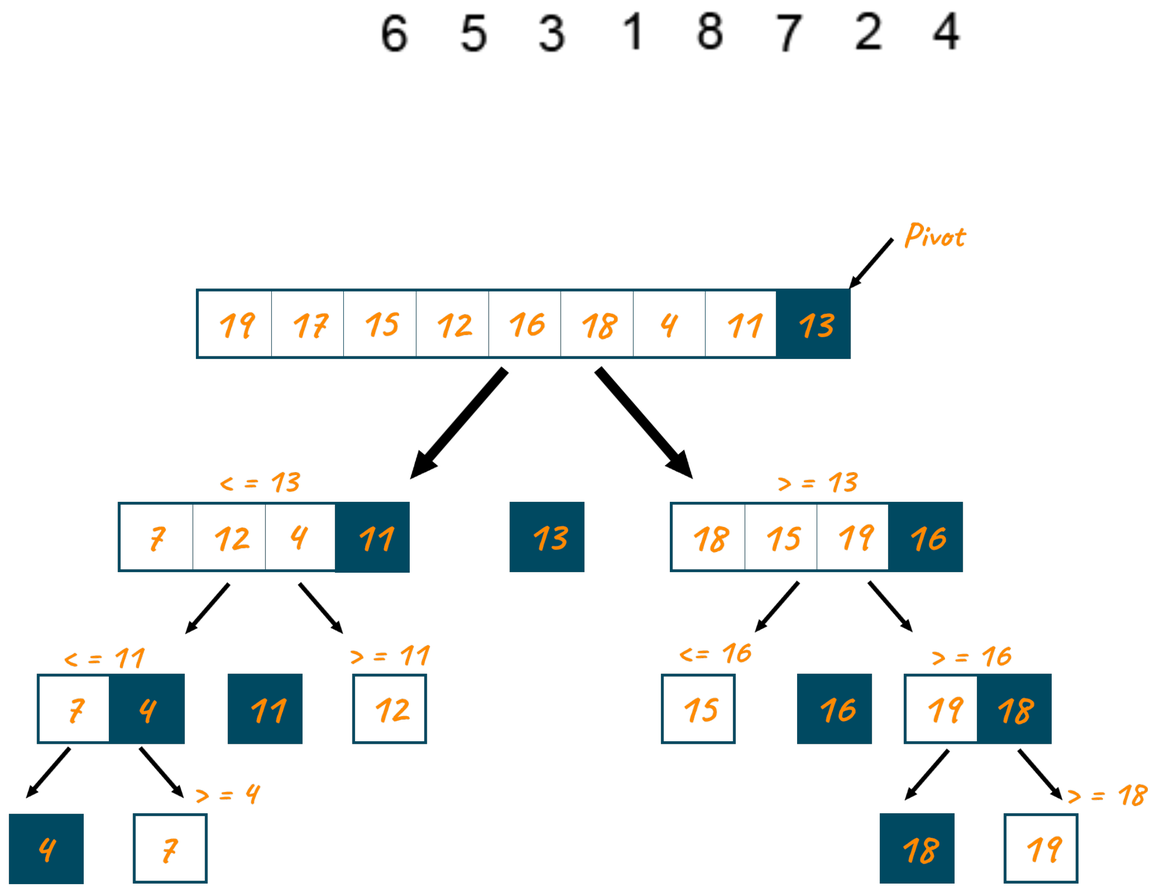
# Quick sort

## 📌 Karakteristike

- Deluje **rekurzivno** kao merge sort
- **Brži u praksi** ako su podaci već u memoriji
- **Manje memorijski zahtevan** od merge sorta (*sortira u mestu*)
- **Složeniji** za implementaciju

## 📉 Kompleksnost:

- **Najgori slučaj:** $O(n^2)$ (*niz je već sortiran sa lošim pivotom*)
- **Prosečan slučaj:** $O(n * log\ n)$



# Partition funckija

```python
def partition(niz, donjaGranica, gornjaGranica):
    pivot = niz[gornjaGranica]
    i = donjaGranica - 1

    for j in range(donjaGranica, gornjaGranica):
        if niz[j] ⩽ pivot:
            i += 1
            niz[i], niz[j] = niz[j], niz[i]

    niz[i + 1], niz[gornjaGranica] = niz[gornjaGranica], niz[i + 1]
    return i + 1
```

✏️ **Objašnjenje:**

- **Pivot** je poslednji element podniza
- **Elementi manji od pivota** se pomeraju ulevo
- Na kraju **pivot ide na svoje mesto** (tačka razdvajanja)

# Glavna funkcija

```
def quickSort(niz, donjaGranica, gornjaGranica):
    if donjaGranica < gornjaGranica:
        pi = partition(niz, donjaGranica, gornjaGranica)
        quickSort(niz, donjaGranica, pi - 1)
        quickSort(niz, pi + 1, gornjaGranica)
```

### ⚠️ Napomena:

- Prvi poziv: `donjaGranica = 0, gornjaGranica = len(niz)-1`
- **Svaki rekurzivni poziv** sortira podniz oko **novog pivota**
- Algoritam je **nestabilan**
- Sortira **in-place** (ne koristi dodatnu memeriju)

### ✅ Zašto je koristan:

- **Veoma brz u praksi** na velikim skupovima podataka
- **Jedan od najčešće korišćenih** sort algoritama

Softverski algoritmi u sistemima automatskog upravljanja

# Zadaci za vežbu

Danilo Kaćanski, Ivan Radman

# Da li niz sadrži duplikat? (*217.*)

1. Dat je niz celobrojnih vrednosti nums. Vrati `True` ako se bilo koji broj pojavljuje **barem dva puta**, a `False` ako su **svi brojevi različiti**.

**Example 1:**

> **Input:** `nums = [1,2,3,1]`
>
> **Output:** `true`
>
> **Explanation:**
>
> The element 1 occurs at the indices 0 and 3.

**Example 2:**

> **Input:** `nums = [1,2,3,4]`
>
> **Output:** `false`
>
> **Explanation:**
>
> All elements are distinct.

**Example 3:**

> **Input:** `nums = [1,1,1,3,3,4,3,2,4,2]`
>
> **Output:** `true`

# Da li je reč anagram? (*242.*)

2. Data su dva stringa `s` i `t`. Vrati `True` ako je `t` **anagram** stringa `s`, a `False` u suprotnom. **Anagram** je reč ili fraza koja nastaje **premeštanjem slova** iz druge reči, koristeći **sva slova tačno jednom**.

**Example 1:**

> **Input:** `s = "anagram", t = "nagaram"`
>
> **Output:** `true`

**Example 2:**

> **Input:** `s = "rat", t = "car"`
>
> **Output:** `false`

# Sortiraj parne i neparne indekse (*2164.*)

3. Dat je niz celih brojeva `nums`, indeksiran od nule (`0`-indeksiran). Preuredi elemente niza prema sledećim pravilima:
    1. **Elementi na neparnim indeksima** (1, 3, 5, ...) treba da budu sortirani opadajuće (*od većeg ka manjem*)
    2. **Elementi na parnim indeksima** (0, 2, 4, ...) treba da budu sortirani rastuće (*od manjeg ka većem*)

   Ova dva dela sortiranja se rade nezavisno.

**Example 1:**

> **Input:** nums = [4,1,2,3]
> **Output:** [2,3,4,1]
> **Explanation:**
> First, we sort the values present at odd indices (1 and 3) in non-increasing order.
> So, nums changes from [4,**1**,2,**3**] to [4,**3**,2,**1**].
> Next, we sort the values present at even indices (0 and 2) in non-decreasing order.
> So, nums changes from [**4**,1,**2**,3] to [**2**,3,**4**,1].
> Thus, the array formed after rearranging the values is [2,3,4,1].

**Example 2:**

> **Input:** nums = [2,1]
> **Output:** [2,1]
> **Explanation:**
> Since there is exactly one odd index and one even index, no rearrangement of values takes place.
> The resultant array formed is [2,1], which is the same as the initial array.

# Sortiraj niz po parnosti (*905.*)

4. Dat je niz celih brojeva `nums`. Potrebno je da svi parni brojevi budu na početku niza, a neparni nakon njih.

   Nije bitan redosled unutar parnih i neparnih brojeva — **bilo koji rezultat** koji zadovoljava uslov je prihvatljiv.

**Example 1:**

> **Input:** nums = [3,1,2,4]
> **Output:** [2,4,3,1]
> **Explanation:** The outputs [4,2,3,1], [2,4,1,3], and [4,2,1,3] would also be accepted.

**Example 2:**

> **Input:** nums = [0]
> **Output:** [0]

# Relativni plasmani takičara (*508.*)

5. Dat je niz `score` dužine `n`, gde `score[i]` predstavlja rezultat `i`-tog takmičara. Svi rezultati su **jedinstveni**. Potrebno je odrediti **plasman** svakog takmičara, tako da:
   - Takmičar sa **najvećim rezultatom** dobija `"Gold Medal"`,
   - Takmičar sa **drugim najvećim rezultatom** dobija `"Silver Medal"`,
   - Takmičar sa **trećim najvećim rezultatom** dobija `"Bronze Medal"`,
   - Ostali dobijaju plasman kao **tekstualni broj** (`"4"`, `"5"`, itd.).

   Vrati niz `answer` iste dužine, gde `answer[i]` predstavlja plasman takmičara sa rezultatom `score[i]`.

**Example 1:**

> **Input:** score = [5,4,3,2,1]
> **Output:** ["Gold Medal","Silver Medal","Bronze Medal","4","5"]
> **Explanation:** The placements are [1st, 2nd, 3rd, 4th, 5th].

**Example 2:**

> **Input:** score = [10,3,8,9,4]
> **Output:** ["Gold Medal","5","Bronze Medal","Silver Medal","4"]
> **Explanation:** The placements are [1st, 5th, 3rd, 2nd, 4th].