STRUKTURE PODATAKA • PRIMERI U PYTHON-U

Polja

Vektori, matrice i višedimenzionalna polja, pristup elementima, traženje i sortiranje

PYTHON

```
a = [3, 5, 2, 7, 9]
print(a[3])
```

IZLAZ

```
7
```

DEFINICIJA

Polje je skup elemenata iste vrste sa direktnim pristupom preko indeksa

list

Ugrađena dinamička lista. Čuva reference, pa elementi mogu biti različitog tipa.

array.array

Polje jednog numeričkog tipa, kompaktno kao u C-u. Ugrađen: from array import array.

numpy.ndarray

Višedimenzionalno polje za numeričke proračune. Spoljna biblioteka: pip install numpy.

U ovom predavanju koristimo **list** kao polje: pristup elementu po indeksu traje isto, bez obzira na veličinu.

MODULI U PRIMERIMA

Šta je ugrađeno, a šta se mora instalirati

Modul	Odakle dolazi	Kako se uključuje	Instalacija
array	standardna biblioteka	`from array import array`	nije potrebna
abc	standardna biblioteka	`from abc import ABC, abstractmethod`	nije potrebna
numpy	spoljna biblioteka (PyPI)	`import numpy as np`	pip install numpy

import modul

Uvozi ceo modul; ime se piše sa prefiksom: **import array** pa **array.array('i', [1, 2])**

from modul import ime

Uvozi samo navedena imena: **from array import array** pa **array('i', [1, 2])**

TIPOVI POLJA

Polja se razlikuju po broju indeksa

1D

Vektori

Jednodimenzionalna polja, jedan indeks: a[i].

2D

Matrice

Dvodimenzionalna polja, vrsta i kolona: M[i][j].

ND

Višedimenzionalna

N indeksa: A[i][j][k]…

JEDNODIMENZIONALNO POLJE

Indeks određuje poziciju, vrednost je sadržaj ćelije

0	1	2	3	4	5	6	7	8	9
3	5	2	7	9	4	1	10	-3	0
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Gore: indeks od početka • Dole: negativni indeks od kraja

PYTHON

```
a = [3, 5, 2, 7, 9, 4, 1, 10, -3, 0]
print(a[8], a[-1], len(a))
```

IZLAZ

```
-3 0 10
```

PRISTUP ELEMENTIMA

Adresa elementa se računa, a ne traži

$$\text{adresa}(a[i]) = \alpha + \sigma \cdot (i - \delta)$$

$$\delta = 0 \Rightarrow \alpha + \sigma \cdot i$$

α	adresa prvog elementa
σ	veličina jednog elementa
δ	početna vrednost indeksa

Python indeksira od nule, kao C/C++. Lista čuva reference (σ je veličina pokazivača), a **array.array** same vrednosti.

Kako se C++ deklaracije prevode u Python

Operacija	C/C++	Python
Fiksna veličina	`int a[10];`	`a = [0] * 10`
Dinamička veličina	`int* a = new int[len];`	`a = [0] * n`
Dodela vrednosti	`int a[3] = {1,2,3};`	`a = [1, 2, 3]`
Dealokacija	`delete [] a;`	`del a (GC)`
Polje jednog tipa	`int a[10];`	`array('i', [0]*10)`

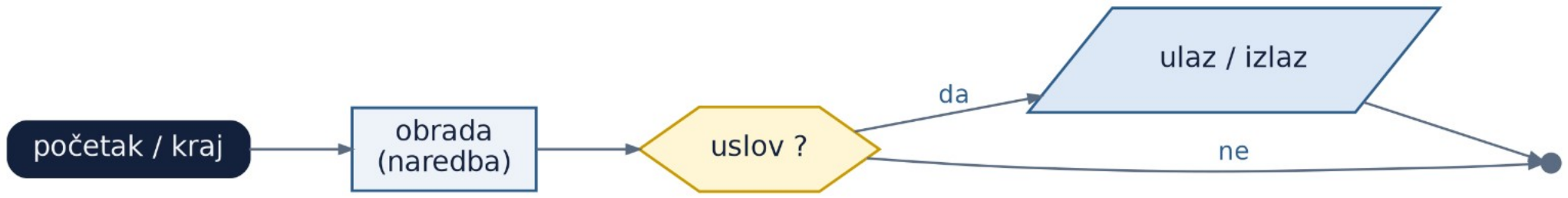
OPERACIJE NAD POLJEM

Pet osnovnih operacija nad poljem

Operacija	Opis	U Python-u
Obilazak	obrada svakog elementa	`for x in a: ...`
Traženje	pronalazak zadate vrednosti	`binary_search(a, x, 0, n - 1)`
Umetanje	novi element na zadatu poziciju	`a.insert(i, x)`
Brisanje	po vrednosti ili po poziciji	`a.remove(x), a.pop(i)`
Sortiranje	rastući ili opadajući redosled	`bubble_sort(a) … merge_sort(…)`

DIJAGRAMI TOKA

Kako se crta algoritam



Početak / kraj

Zaobljeni pravougaonik. Na početku je naziv funkcije sa parametrima.

Obrada

Pravougaonik: dodela vrednosti ili poziv funkcije.

Uslov

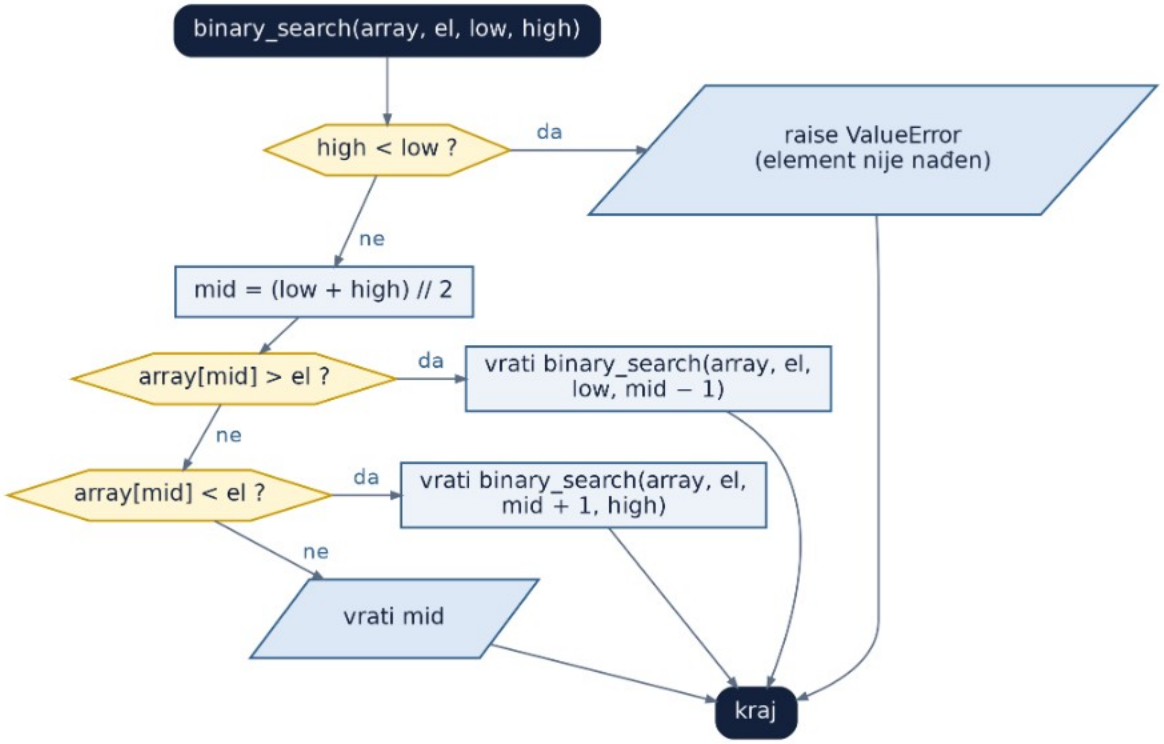
Šestougao sa dve izlazne grane: da i ne. Petlja je uslov sa granom nazad.

Ulaz / izlaz

Paralelogram: vraćanje rezultata ili podizanje izuzetka.

BINARNO TRAŽENJE

Dijagram toka



Ideja

Poredi traženu vrednost sa srednjim elementom i nastavi u polovini u kojoj ona može biti.

Složenost

Opseg se prepolovi u svakom koraku: O(log n). Polje mora biti sortirano.

BINARNO TRAŽENJE • SKICA RADA

Da li se 76 nalazi u sortiranom polju?

`min=0 mid=7 max=15`	2	9	11	15	28	33	40	47	51	64	76	78	82	85	91	99
`min=8 mid=11 max=15`	2	9	11	15	28	33	40	47	51	64	76	78	82	85	91	99
`min=8 mid=9 max=10`	2	9	11	15	28	33	40	47	51	64	76	78	82	85	91	99
`min=10 mid=10 max=10`	2	9	11	15	28	33	40	47	51	64	76	78	82	85	91	99

Svaki korak prepolovi opseg: 16 elemenata → najviše 4 poređenja, složenost **O(log n)**.

BINARNO TRAŽENJE

Rekurzivna implementacija

PYTHON

```
def binary_search(array, el, low, high):
    if high < low:
        raise ValueError("Element nije nađen")
    mid = (low + high) // 2
    if array[mid] > el:
        return binary_search(array, el, low, mid - 1)
    elif array[mid] < el:
        return binary_search(array, el, mid + 1, high)
    return mid
```

POZIV

```
data = [2, 9, 11, 15, 28, 33,
        40, 47, 51, 64, 76, 78,
        82, 85, 91, 99]
print(binary_search(data, 76, 0, 15))
```

IZLAZ

```
10
```

Klasa Array sa proizvoljnom osnovom indeksa

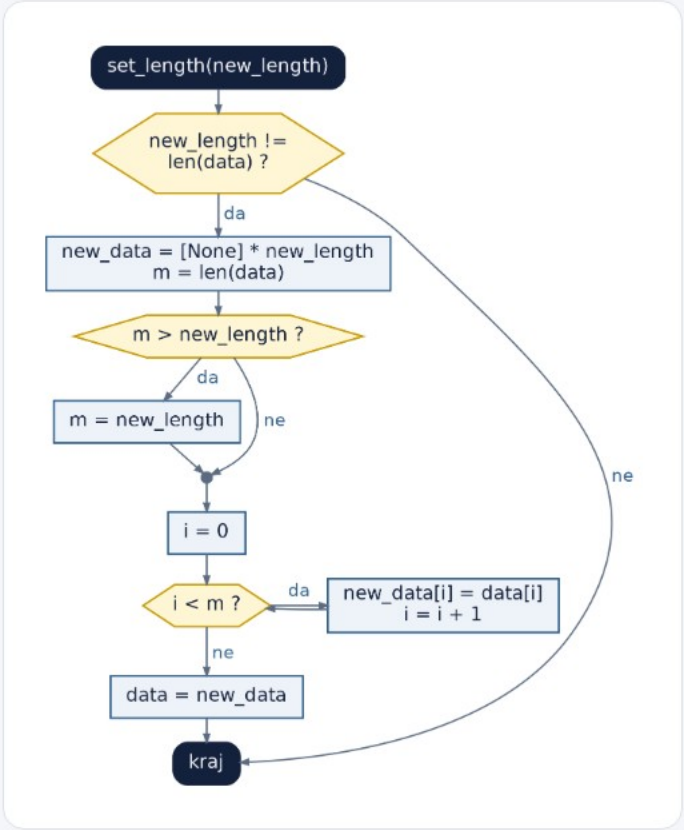
PYTHON

```
class Array:
    def __init__(self, length, base=0):
        self.base = base              # osnova
        self.data = [None] * length   # podaci

    def __len__(self):
        return len(self.data)

    def _offset(self, index):
        offset = index - self.base
        if not 0 ⩽ offset < len(self.data):
            raise IndexError("Indeks van granica!")
        return offset
```

`base`	prvi dozvoljeni indeks, npr. -1
`_offset`	indeks → pozicija u listi
`__len__`	omogućava len(p)

PROMENA VELIČINE POLJA

Dijagram toka i skica rada metode set_length



Povećanje sa 3 na 5 (m = 3)

staro	3	5	2		
novo	3	5	2	·	·

Smanjenje sa 5 na 2 (m = 2)

staro	3	5	2	·	·
novo	3	5			

kopira se kopirano prazno

Operator [] i promena veličine polja

PYTHON

```
    def __getitem__(self, index):
        return self.data[self._offset(index)]
    def __setitem__(self, index, value):
        self.data[self._offset(index)] = value
    def set_length(self, new_length):
        if new_length ≠ len(self.data):
            new_data = [None] * new_length
            m = len(self.data)
            if m > new_length:
                m = new_length
            for i in range(m):
                new_data[i] = self.data[i]
            self.data = new_data
```

`__getitem__`	čitanje: p[i]
`__setitem__`	upis: p[i] = x
`set_length`	novo polje + kopiranje

PRIMER KORIŠĆENJA

Polje sa osnovom –1 i provera granica

PYTHON

```
p = Array(3, base=-1)
p[-1] = 3
p[0] = 5
p[1] = 2
p.print()

p.set_length(5)
print(len(p), p.data)

try:
    p[10] = 1
except IndexError as e:
    print("Greška:", e)
```

IZLAZ

```
data[-1]=3
data[0]=5
data[1]=2
5 [3, 5, 2, None, None]
Greška: Indeks van granica!
```

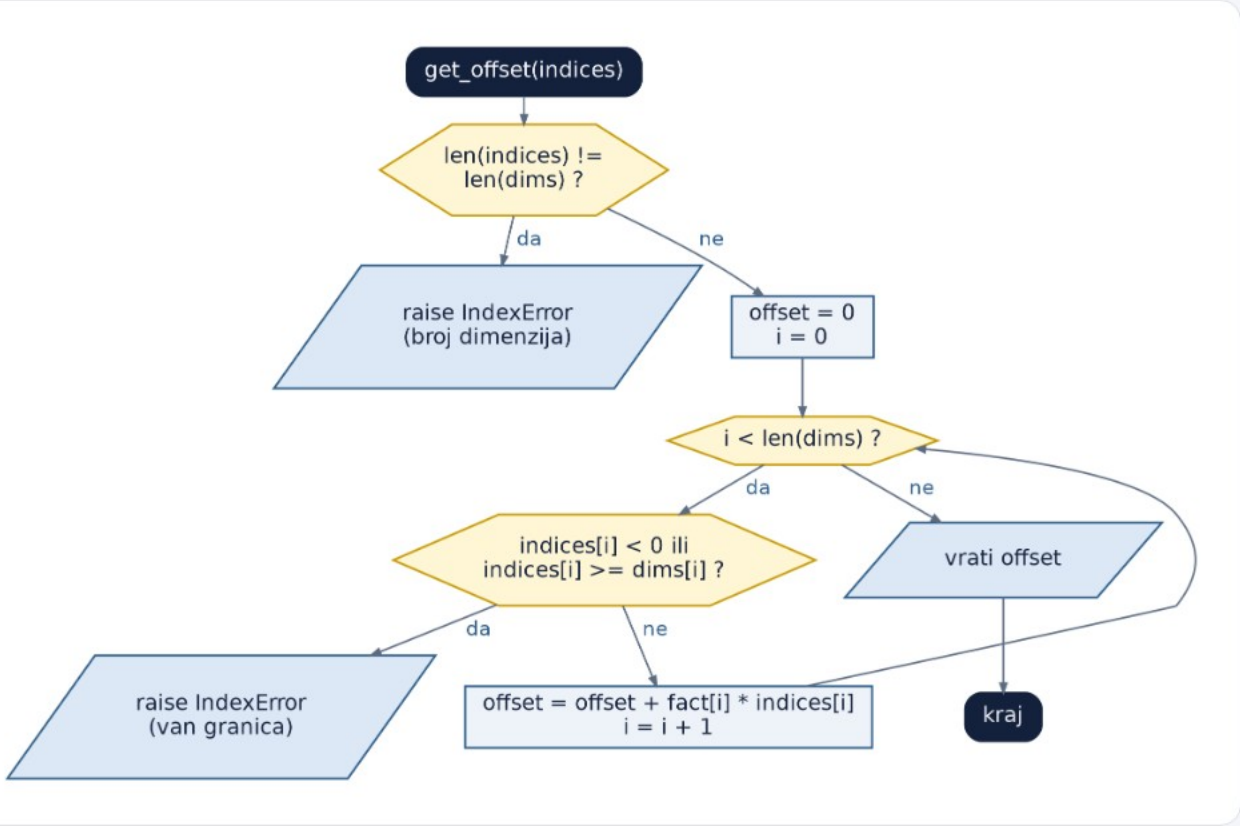
N indeksa se svodi na jedan pomeraj

$$\text{adresa}(a[i_0, \ldots, i_{n-1}]) = \alpha + \sigma \cdot \Sigma\ \text{faktori}[k] \cdot i_k$$

```
faktori[k] = 1, za k = N - 1
faktori[k] = dim[k+1] · … · dim[N-1], za k < N - 1
```

Za polje 2 × 3 × 4: faktori su [12, 4, 1], pa je element [1, 2, 3] na poziciji 12 + 8 + 3 = **23**.

Dijagram toka i skica za polje 2 × 3 × 4, indeksi [1, 2, 3]



Faktori (konstruktor)

i	fact[i]	product
2	1	4
1	4	12
0	12	24

Pomeraj za [1, 2, 3]

k	fact	idx	fact·idx	offset
0	12	1	12	12
1	4	2	8	20
2	1	3	3	**23**

KLASA VIŠEDIMENZIONALNOG POLJA

Faktori se računaju jednom, u konstruktoru

KONSTRUKTOR

```
class MultiDimensionalArray:
    def __init__(self, *dims):
        self.dims = dims
        self.fact = [0] * len(dims)
        product = 1
        for i in range(len(dims) -
 1, -1, -1):
            self.fact[i] = product
            product *= dims[i]
        self.data = [None] * product
```

POMERAJ

```
    def get_offset(self, indices):
        if len(indices) ≠ len(self.dims):
            raise IndexError("Broj dimenzija!")
        offset = 0
        for i in range(len(self.dims)):
            if indices[i] < 0 or indices[i] ⩾ self.dims[i]:
                raise IndexError("Van granica!")
            offset += self.fact[i] * indices[i]
        return offset
```

MATRICE U PYTHON-U

Lista listi i jedna česta zamka

ISPRAVNO

```
M = [[0] * 4 for _ in range(3)]
M[1][2] = 7
print(M)
```

IZLAZ

```
[[0, 0, 0, 0], [0, 0, 7, 0], [0, 0, 0, 0]]
```

POGREŠNO

```
bad = [[0] * 3] * 2
bad[0][0] = 1
print(bad)
```

IZLAZ

```
[[1, 0, 0], [1, 0, 0]]
```

Množenje liste kopira **reference**: obe vrste su isti objekat. Svaku vrstu pravite posebno.

MATRICE

Dvodimenzionalna polja: prvi indeks je vrsta, drugi kolona

Kvadratne

Isti broj vrsta i kolona (square matrix).

Pravougaone

Različit broj vrsta i kolona (rectangular matrix).

Retko posednute

Većina elemenata je nula (sparse matrix).

Guste

Većina elemenata je popunjena (dense matrix).

SPECIJALNE MATRICE

Trougaona matrica čuva samo n(n+1)/2 elemenata



Gornje trougaona



Donje trougaona

DONJE TROUGAONA

```
def lower_index(i, j):   # j ⩽ i
    return i * (i + 1) // 2 + j
```

Simetrična matrica (a[i][j] = a[j][i]) takođe se čuva kao donje ili gornje trougaona.

APSTRAKTNA KLASA MATRICE

Modul abc zamenjuje virtuelne funkcije iz C++

MATRIX · ABC JE UGRAĐEN

```
from abc import ABC, abstractmethod

class Matrix(ABC):
    @abstractmethod
    def get(self, i, j): ...

    @abstractmethod
    def put(self, i, j, value): ...
```

DENSEMATRIX

```
class DenseMatrix(Matrix):
    def __init__(self, rows, cols):
        self.rows, self.cols = rows, cols
        self.data = [0.0] * (rows * cols)

    def get(self, i, j):
        return self.data[i * self.cols + j]

    def put(self, i, j, value):
        self.data[i * self.cols + j] = value
```

PRIMENA POLJA

Četiri klasična algoritma sortiranja

Bubble sort

Metod mehurova: susedni elementi se porede i menjaju mesta.

Selection sort

Metod selekcije: najmanji element ide na sledeće mesto.

Insertion sort

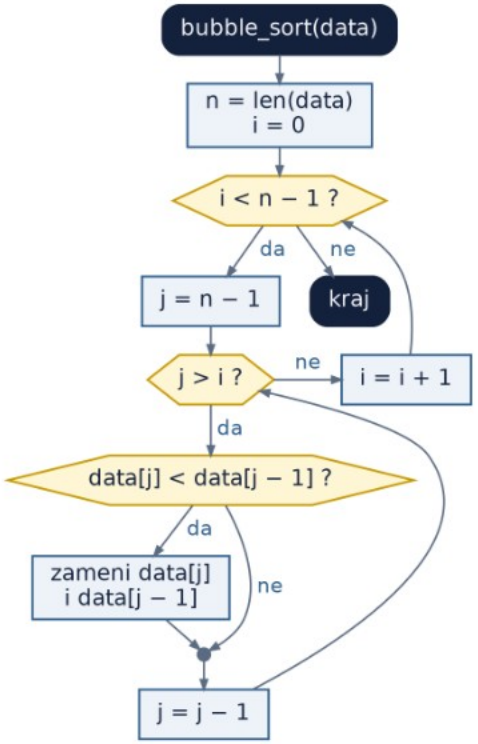
Metod umetanja: element se umeće u već sortirani deo.

Merge sort

Metod spajanja: deli polje na pola, sortira i spaja.

BUBBLE SORT

Dijagram toka



Ideja

Dve ugnježđene petlje. Unutrašnja ide sa kraja ka početku i menja susedne elemente koji nisu u redu.

Složenost

Uvek $n(n-1)/2$ poređenja: $O(n^2)$.

BUBBLE SORT

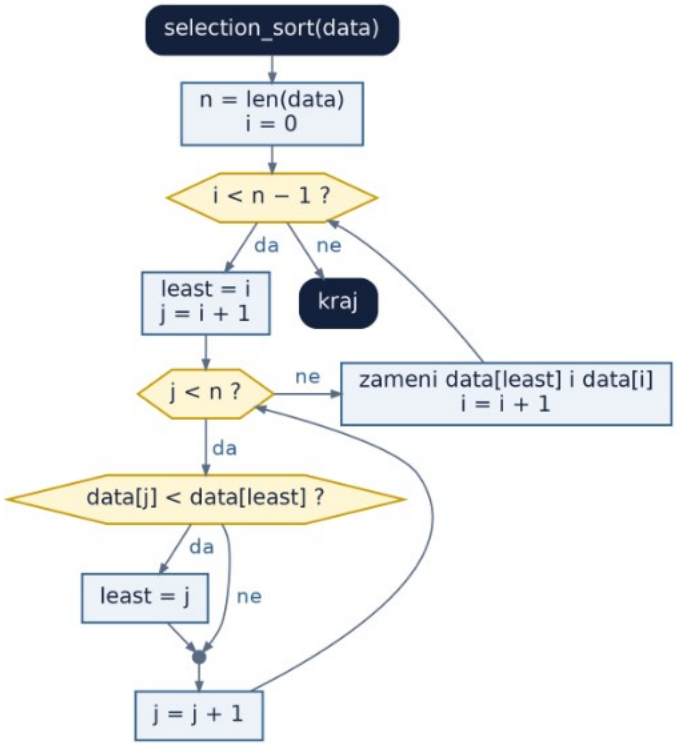
Algoritam i skica rada na polju [5, 2, 4, 1, 3]

1. Za i od 0 do n – 2:
2. idi sa kraja polja (j = n – 1) unazad do i + 1,
3. ako je **data[j] < data[j – 1]**, zameni ih.
4. Posle prolaza i, najmanji od preostalih je na poziciji i.



SELECTION SORT

Dijagram toka



Ideja

Unutrašnja petlja pamti indeks najmanjeg elementa u ostatku, a spoljašnja ga zamenjuje sa i-tim.

Složenost

n(n − 1)/2 poređenja, najviše n − 1 zamena: $O(n^2)$.

SELECTION SORT

Algoritam i skica rada na polju [5, 2, 4, 1, 3]

1. Za i od 0 do n – 2:
2. nađi indeks **least** najmanjeg elementa u data[i:],
3. zameni data[least] i data[i].
4. Levo od i je uvek sortiran deo.



Dva jednostavna algoritma složenosti O(n²)

BUBBLE SORT

```
def bubble_sort(data):
    n = len(data)
    for i in range(n - 1):
        for j in range(n - 1, i, -1):
            if data[j] < data[j - 1]:
                data[j], data[j - 1] = \
                    data[j - 1], data[j]
```

Najmanji element „isplivava“ na početak u svakom prolazu.

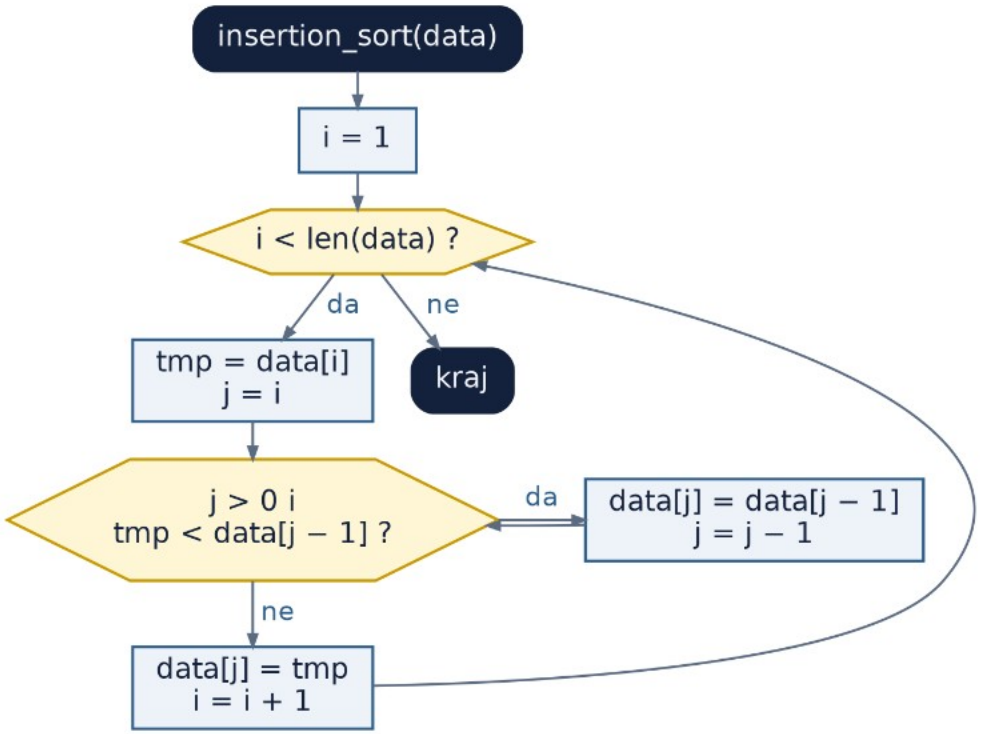
SELECTION SORT

```
def selection_sort(data):
    n = len(data)
    for i in range(n - 1):
        least = i
        for j in range(i + 1, n):
            if data[j] < data[least]:
                least = j
        data[least], data[i] = data[i], data[least]
```

Traži se najmanji u ostatku i menja mesto sa i-tim.

INSERTION SORT

Dijagram toka



Ideja

tmp čuva element koji se umeće; unutrašnja petlja pomera veće elemente udesno.

Složenost

O(n²) u najgorem slučaju, O(n) za već sortirano polje.

INSERTION SORT

Algoritam i skica rada na polju [5, 2, 4, 1, 3]

1. Za i od 1 do n − 1: **tmp = data[i]**.
2. Dok je levi sused veći od tmp, pomeri ga za jedno mesto udesno.
3. Upiši tmp na oslobođeno mesto.
4. Levo od i + 1 je sortiran deo.



INSERTION SORT

Element se umeće na svoje mesto u sortiranom delu

PYTHON

```
def insertion_sort(data):
    for i in range(1, len(data)):
        tmp = data[i]
        j = i
        while j > 0 and tmp < data[j - 1]:
            data[j] = data[j - 1]
            j -= 1
        data[j] = tmp
```

POZIV

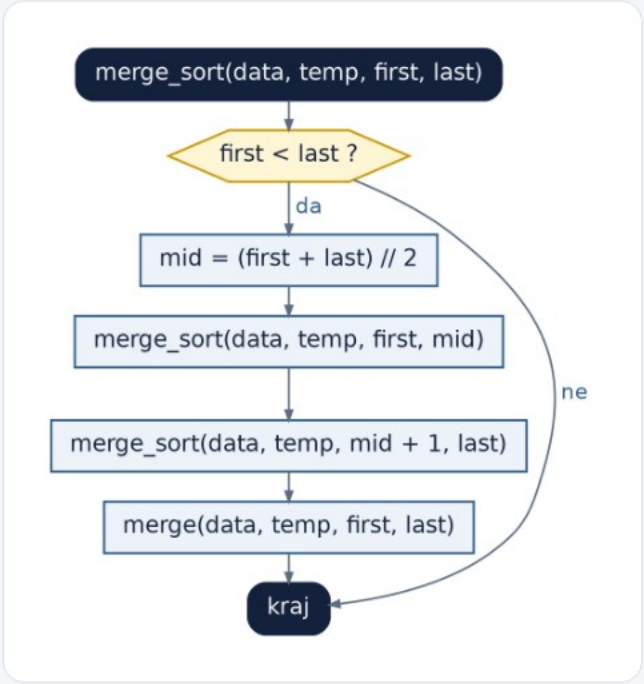
```
x = [3, 5, 2, 7, 9, 4, 1, 10, -3, 0]
insertion_sort(x)
print(x)
```

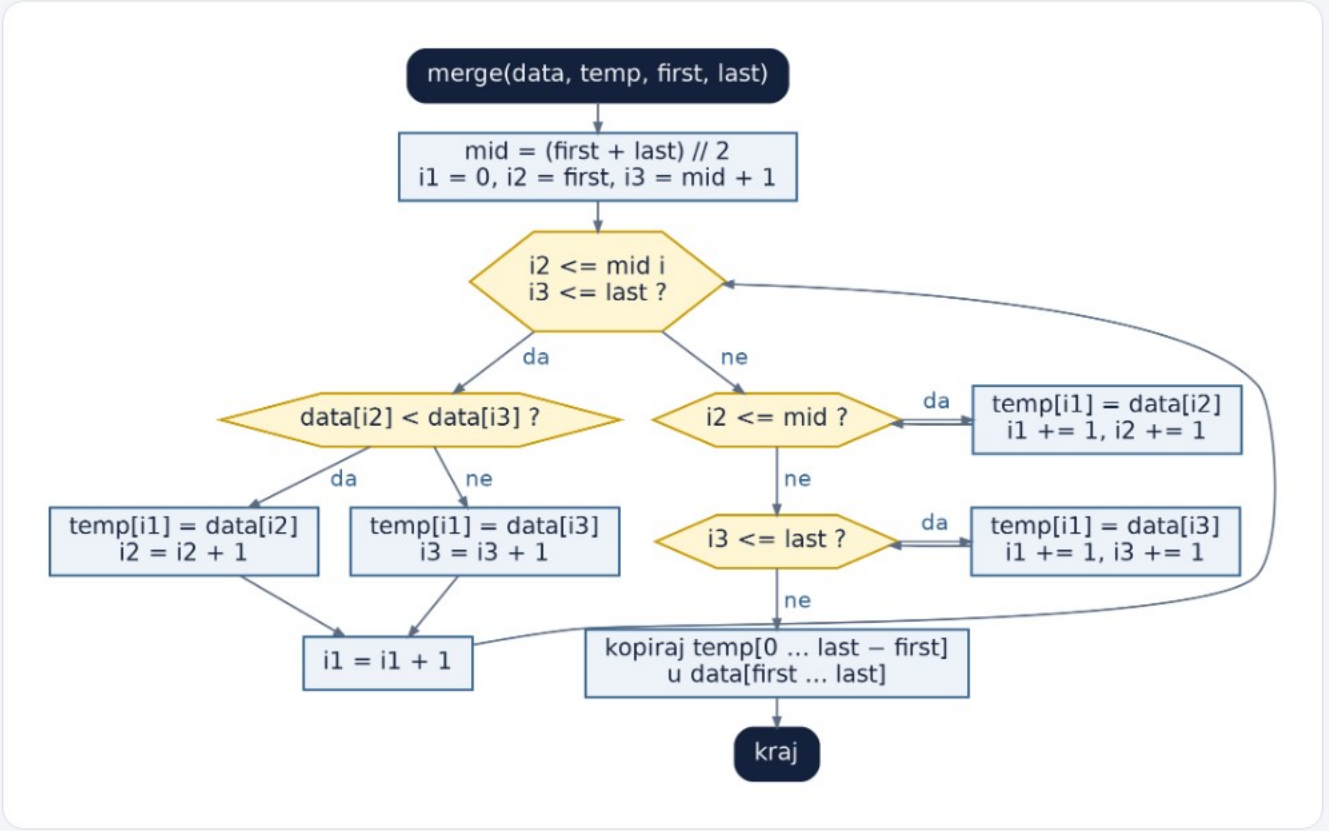
IZLAZ

```
[-3, 0, 1, 2, 3, 4, 5, 7, 9, 10]
```

MERGE SORT

Dijagram toka rekurzije i spajanja





MERGE SORT

Algoritam i skica rada na polju [5, 2, 4, 1, 3]

1. Ako polje ima 0 ili 1 element, već je sortirano.
2. Podeli polje na dve polovine.
3. Rekurzivno sortiraj obe polovine.
4. **merge**: poredi prve elemente polovina i manji upiši u temp.
5. Prepiši ostatak polovine, pa temp vrati u data.



MERGE SORT

Rekurzivna funkcija i poziv

MERGE_SORT

```
def merge_sort(data, temp, first, last):
    if first < last:
        mid = (first + last) // 2
        merge_sort(data, temp, first, mid)
        merge_sort(data, temp, mid + 1, last)
        merge(data, temp, first, last)
```

POZIV

```
x = [3, 5, 2, 7, 9, 4, 1, 10, -3, 0]
temp = [0] * len(x)
merge_sort(x, temp, 0, len(x) - 1)
print(x)
```

IZLAZ

```
[-3, 0, 1, 2, 3, 4, 5, 7, 9, 10]
```

Složenost **O(n log n)**; pomoćno polje **temp** je iste veličine kao polje koje se uređuje.

MERGE SORT

Spajanje dve sortirane polovine

```
def merge(data, temp, first, last):
    mid = (first + last) // 2
    i1, i2, i3 = 0, first, mid + 1
    while i2 ⩽ mid and i3 ⩽ last:
        if data[i2] < data[i3]:
            temp[i1] = data[i2]; i2 += 1
        else:
            temp[i1] = data[i3]; i3 += 1
        i1 += 1
    while i2 ⩽ mid:
        temp[i1] = data[i2]; i1 += 1; i2 += 1
    while i3 ⩽ last:
        temp[i1] = data[i3]; i1 += 1; i3 += 1
    for k in range(last - first + 1):
        data[first + k] = temp[k]
```

i2	leva polovina
i3	desna polovina
i1	pozicija u temp

POREĐENJE EFIKASNOSTI

Broj poređenja po algoritmu i slučaju

Algoritam	Najbolji	Srednji	Najgori
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection sort	$O(n)$*	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

* odnosi se na broj zamena (n – 1); broj poređenja je uvek n(n – 1)/2.

ZADACI ZA PROVERU

Proverite razumevanje gradiva

1

Izračunajte koliko množenja i sabiranja štede unapred izračunati faktori kod M-dimenzionalnog polja.

2

Izvedite izraze za pristup elementima gornje i donje trougaone matrice.

3

Napišite klasu za donje trougaonu matricu sa metodama get(i, j) i put(i, j, value).

4

Izračunajte uštedu memorije u odnosu na gustu matricu i složenost pristupa.