Blok 04
Struktura podataka i algoritmi

prof. Dr Vidan Marković

Univerzitet Singidunum
2023/2024

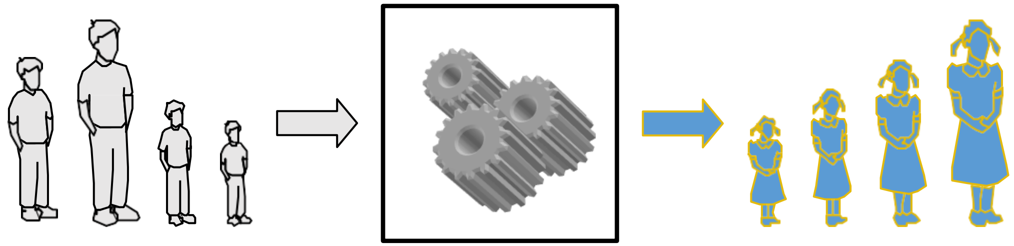
Sadržaj

1. Dizajn i analiza algoritama – deep dive

2. Izračunavanje efiksanost algoritama sa primerima

3. Diskusija

Dizajn i analiza algoritma



Input **Algorithm** **Output**

Algoritam je procedura "korak po korak" za rešavanje problema u određenom vremenskom periodu.

Algoritam

- Definicija: Algoritam je niz koraka koji se koriste za rešavanje problema u konačnom vremenskom periodu

- Svojstva
  - Ispravnost: mora da obezbedi tačan izlaz za svaki unos
  - Performanse: mereno u smislu korišćenih resursa (vreme i prostor)
  - Kraj: mora da se završi u određenom vremenskom periodu ≡ jeziku koji se ponavlja

Ispravnost:

- Algoritam je ispravan ako rešava dati kompjuterski problem.
- Za svaki unos (skup vrednosti) on daje željeni izlaz (drugi skup vrednosti)
- Prekida se u konačnom vremenu

Efikasnost

- Presudno: kako kvantifikovati efikasnost algoritma.
- Korišćeni resursi:
  - Vreme
  - Prostor
- Mere:

  Složenost (korišćenje standardne notacije: O, Ω, Θ)
  - Najgori slučaj
  - Prosečan slučaj
  - U najboljem slučaju

Problem vs algoritam

- Problemi:
  - Pronalaženje elementa u listi intedžera
  - Sortiranje liste brojeva brojeva
  - Najkraća staza u grafikonu
  - Množenje lanca matrice
  - Trup konveksa
  - Problem putujućeg prodavca (TSP)

- Algoritmi:
  - Sekvencijalna pretraga
  - Insertion sort, quicksort, meregsort
  - Dijkstra algoritam
  - Množenje lanca matrice (dinamičko programsko rešenje)
  - Graham scan
  - Backtracking za pronalaženje optimalne ture

Napomena: isti problem se može rešiti mnogim različitim algoritmima

Primer 1: Maksimum

Problem:

- Pronalaženje maksimuma na listi intedžera
- Formalnije:

  Za listu brojeva $a_1, a_2, \ldots, a_n$

  Pronaći broj a_i tako da $\forall j \neq i,\ a_i \geq a_j$

- Algoritam (jedan od mnogih):

```
Algorithm arrayMax(A, n)
   Input array A of n integers
   Output maximum element of A

   currentMax ← A[0]
   for i ← 1 to n – 1 do
      if A[i] > currentMax then
         currentMax ← A[i]
   return currentMax
```

Primer 2: Sortiranje

Problem:

- Sortiranje niza objekata po rastućem redosledu
- Formalnije:

  Za dati niz uporedivih stavki
  $s_1, s_2, \ldots, s_n$
  pronađi permutaciju $s_1', s_2', \ldots, s_n'$ od sekvence takve da $s_1' \leq s_2' \leq \ldots \leq s_n'$

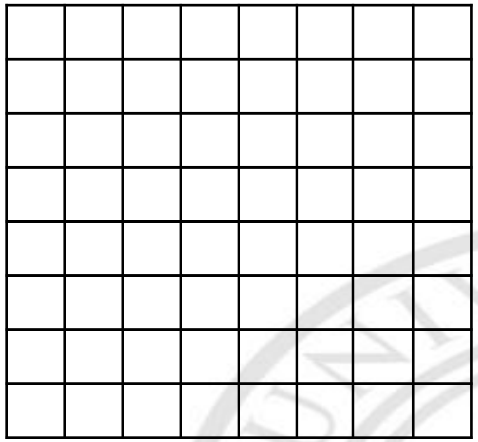
- Algoritam (jedan od mnogih):

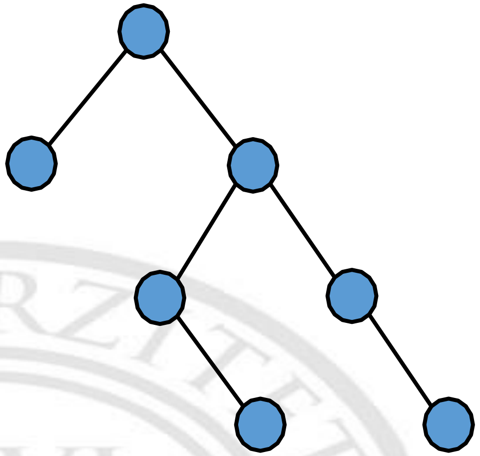
Algorithm *mergeSort*(S, C)
    Input sequence S with n
        elements, comparator C
    Output sequence S sorted
        according to C
    if $S.size() > 1$
        $(S_1, S_2) \leftarrow$ *partition*(S, $n/2$)
        mergeSort(S_1, C)
        mergeSort(S_2, C)
        $S \leftarrow$ *merge*(S_1, S_2)

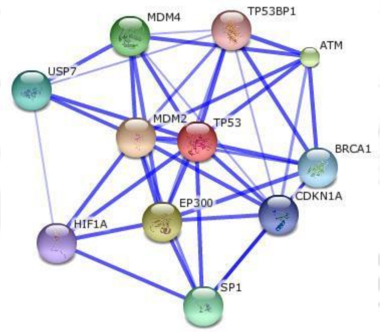
Veličina ulaza

A						
21	22	23	25	24	23	22
0	1	2	3	4	5	6

- Performanse algoritma mere se u smislu veličine inputa
- Primeri:
  - Broj elemenata u listi ili nizu A: *n*
  - Broj ćelija u mxn matrici: *m*, *n*
  - Broj bitova u celom broju: *n*
  - Broj čvorova na stablu: *n*
  - Broj temena i ivica u grafikonu: $|V|$, $|E|$

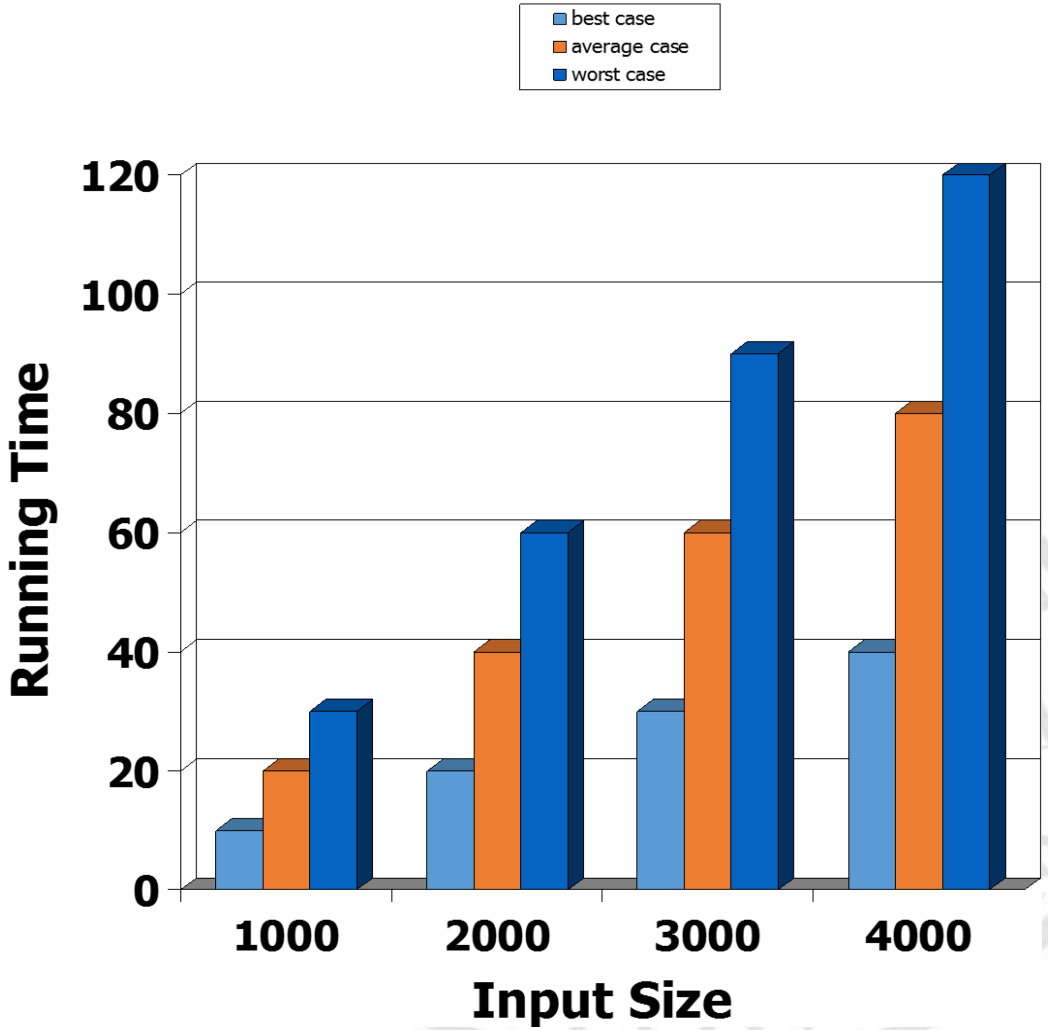




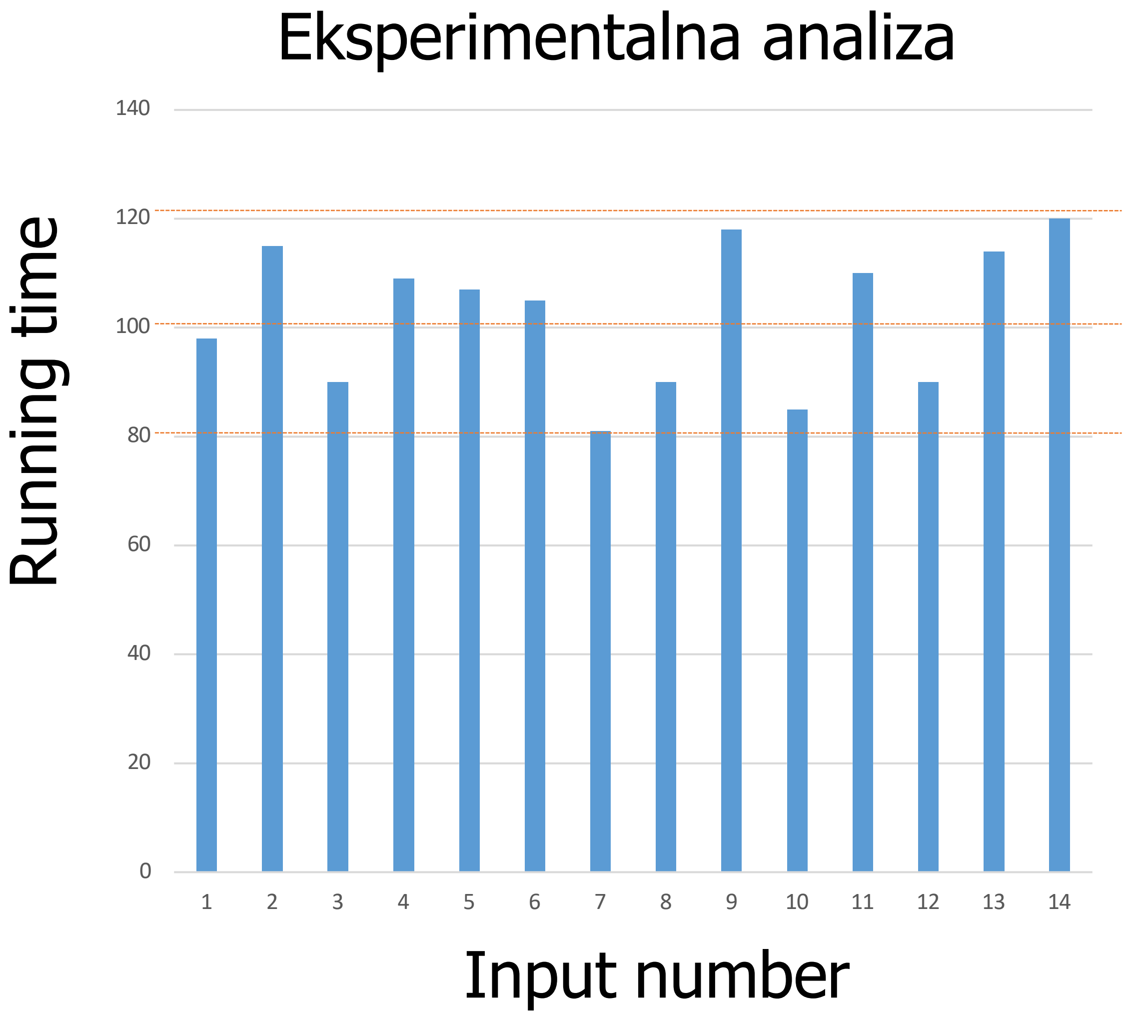


Running Time – vreme izvršavanja

- Većina algoritama transformiše ulazne objekte u izlazne objekte.
- Running time algoritma obično raste sa veličinom inputa.
- Prosečno vreme slučaja je često teško je utvrditi.
- Uglavnom se fokusiramo na najgore vreme izvršavanja.
  - Lakše analizirati
  - Ključno u aplikacijama kao što su socijalni inženjering, sajber bezbednost, igre, bioinformatika, robotika



Analiza slučaja



- Tri slučaja
- Najgori slučaj:
  - među svim mogućim inputima, onima koji
  - oduzima najveće vreme.
- Najbolji slučaj:
  - Unos za koji algoritam radi najbrže
- Prosečan slučaj:
  - Prosek je preko svih mogućih ulaza
  - Može se smatrati očekivanom vrednošću T(n), što je nasumična promenljiva
- Dobro za 14 ulaza
- Ali šta ako razmotrimo sve moguće inpute?
- e.g., niz veličine 10 sadrži 32-bitne intedžere
- Biće 2^{320} tih nizova!

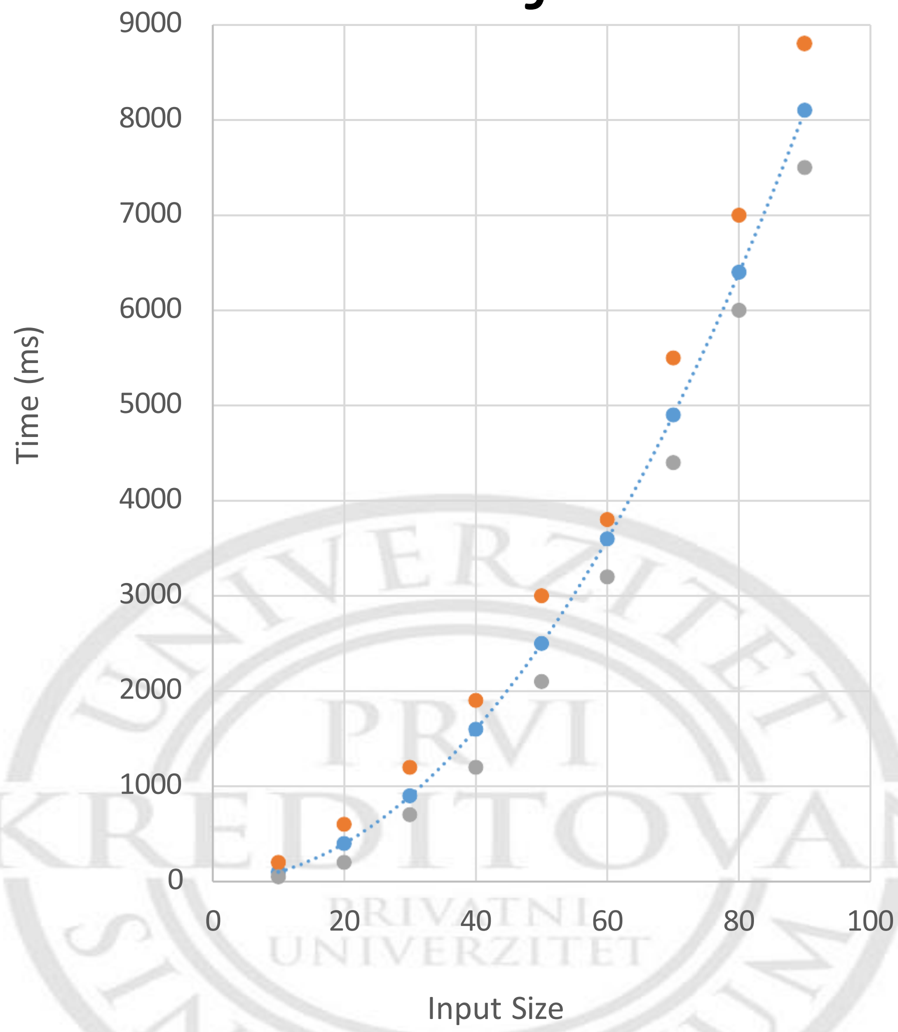
Eksperimentalna vs teorijska analiza

Eksperimentalna analiza:

- Napisati program koji primenjuje algoritam
- Pokretanje programa sa unosima različite veličine i sastava
- Praćenje CPU vremena koje program koristi za svaku veličinu unosa
- Prikazivanje rezultata na dvodimenzionalnom grafu

Ograničenja:

- Zavisi od hardvera i programskog jezika
- Potrebno je primeniti algoritam i otkloniti programske greške

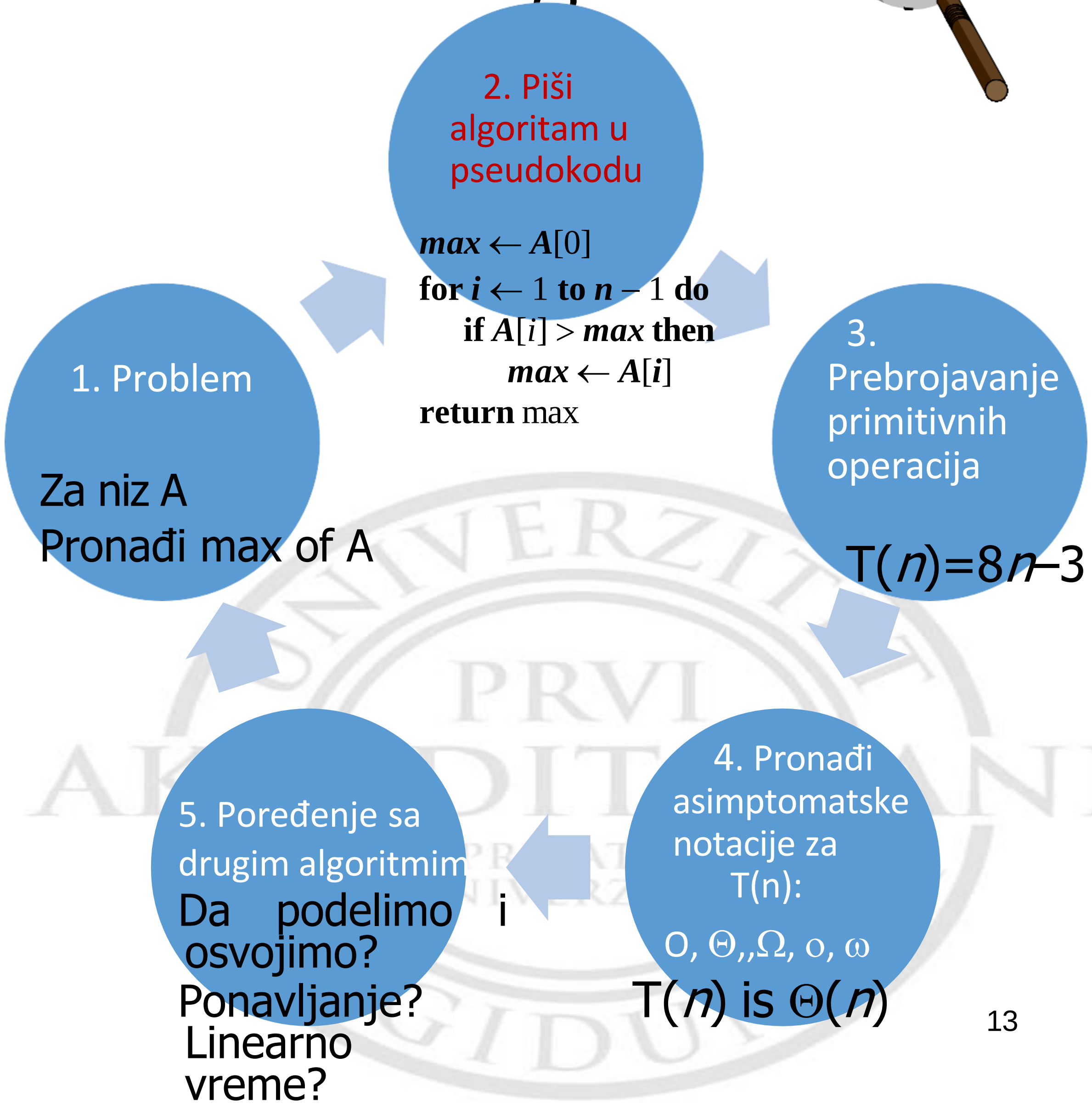


Teorijska analiza – glavni okvir

Prednosti:

- Koristi opis algoritma visokog nivoa umesto implementacije

- Opisuje vreme izvršavanja kao funkciju veličine ulaza, n.
- Uzima u obzir sve moguće ulaze
- Omogućava nam da procenimo brzinu algoritma nezavisno od hardversko/softverskog okruženja



Pseudocode

- Opis algoritma visokog nivoa
- Strukturisanija od engleske proze
- Manje detaljan od programa
- Poželjna napomena za opisivanje algoritama
- Sakrivanje problema sa dizajnom programa

A	3	2	4	8	5	2	6
	0	1	2	3	4	5	6

Primer: pronađi max element niza

Algorithm *arrayMax*(*A*, *n*)
  Input array *A* of *n* integers
  Output maximum element of *A*

  $currentMax \leftarrow A[0]$
  for $i \leftarrow 1$ **to** $n - 1$ **do**
    if $A[i] > currentMax$ **then**
      $currentMax \leftarrow A[i]$
  return *currentMax*

Pseudocode

Da li ćemo koristiti izgled pseudo koda koji ima stil Phytona, Jave, C-a, ili nečega četvrtog nije toliko važno.

```
a := 0
for x := 0 to 10:
    a = a + 1
    print(a)
```

```
a ← 0
for x ← 0 until x ≥ 10:
    a ← a + 1
    output(a)
```

```
a = 0
for(x = 0; x < 10; x++) {
    a = a + 1;
    print(a);
}
```

```
a = 0
For x = 0 to 10 Do
    a = a + 1
    print(a)
End
```

```
list := random list of numbers
even_counter := 0
five_counter := 0
for each number in list:
    if number is even:
        even_counter = evencounter + 1
    if number is divisible by five:
        five_counter = five_counter + 1
```

Pseudocode

- Tok kontrole
- **if** … **then** … [**else** …]
  - **while** … **do** …
  - **repeat** … **until** …
  - **for** … **do** …
  - Indentation replaces braces
- Deklaracija metoda
- **Algorithm** *method* (*arg* [, *arg*…])
  Input …
  Output …
  return …

- Poziv metode

  var.method (*arg* [, *arg*…])

- Povratna vrednost

  return *expression*

- Izrazi

  $\leftarrow$ Assignment (like $=$ in Java)

  $=$ Equality testing (like $==$ in Java)

  n^2 Eksponentni i drugo matematičko oblikovanje dozvoljeno

A	3	2	4	8	5	2	6
	0	1	2	3	4	5	6

Primer: pronalaženje maksimalnog elementa niza

Algorithm *arrayMax*(*A*, *n*)
  Input array *A* of *n* integers
  Output maximum element of *A*

  currentMax $\leftarrow$ *A*[0]
  for *i* $\leftarrow$ 1 **to** *n* − 1 **do**
    if *A*[*i*] > *currentMax* **then**
      currentMax $\leftarrow$ *A*[*i*]
  return *currentMax*

Pseudocode daje opis algoritma na visokom nivou i izbegava da prikaže detalje koji su nepotrebni za analizu.

Pseudocode vs Java Code

Algoritam HTMLTagMatch(X,n):

Input: Niz X of n tokena, svaki je ili HTML tag ili text karakter

Output: **true** if and only if all the HTML tags in X match

Let S be an empty stack

for i=0 to n-1 **do** {n is the number of tokens in X}

  if $X[i]$ is an opening HTML tag **then**

    S.push($X[i]$)

  else if $X[i]$ is a closing HTML tag **then**

    if S.isEmpty() **then**

      return false {nothing to match with}

    if S.pop() does not match the tag of $X[i]$ **then**

      return false {wrong tag}

if S.isEmpty() **then**

  return true {every tag matched}

else

  return false {some tags were never matched}

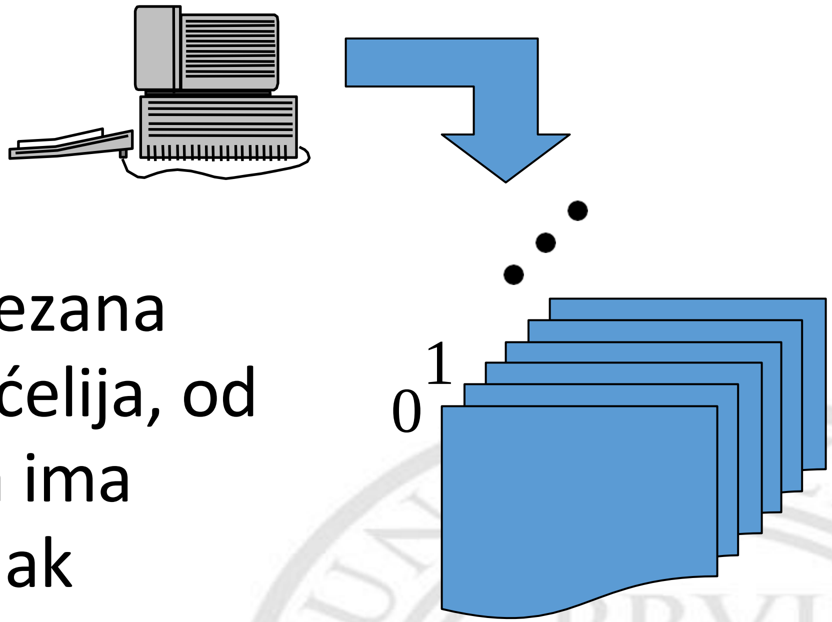
```java
import java.util.StringTokenizer;
import datastructures.Stack;
import datastructures.NodeStack;
import java.io.*;
/** Simplified test of matching tags in an HTML document. */
public class HTML { /** Nested class to store simple HTML tags */
        public static class Tag { String name; // The name of this tag
                boolean opening; // Is true i. this is an opening tag
                public Tag() { // Default constructor
                        name = "";
                        opening = false;
                }
                public Tag(String nm, boolean op) { // Preferred constructor
                name = nm;
                opening = op;
                }
                /** Is this an opening tag? */
                public boolean isOpening() { return opening; }
                /** Return the name of this tag */
                public String getName() {return name; }
        }
        /** Test if every opening tag has a matching closing tag. */
        public boolean isHTMLMatched(Tag[ ] tag) {
                Stack S = new NodeStack(); // Stack for matching tags
                for (int i=0; (i<tag.length) && (tag[i] != null); i++) {
                        if (tag[i].isOpening())
                        S.push(tag[i].getName()); // opening tag; push its name on the stack
                        else {
                            if (S.isEmpty()) // nothing to match
                                        return false;
                            if (!((String) S.pop()).equals(tag[i].getName())) // wrong match
                                        return false;
                        }
                }
                if (S.isEmpty())
                    return true; // we matched everything
                return false; // we have some tags that  were never matched
        }
```

Pseudocode vs Java Code

```
public final static int  CAPACITY = 1000;        // Tag array size upper bound
   /* Parse an HTML document into an array of html tags */
   public  Tag[ ]  parseHTML(BufferedReader r) throws  IOException {
                    String line;       // a line of text
                    boolean          inTag =  false   ;                       // true iff we are in a tag
                     Tag[ ]  tag  =  new  Tag[CAPACITY];  // our tag array (initially all null)
                    int  count   =  0                       ;                       // tag counter
                    while  ((line  =  r.readLine()) !=  null)  {
                // Create a string tokenizer for HTML tags (use  <  and  >  as delimiters)
                StringTokenizer st  =  new  StringTokenizer(line,"<> \t",true);
                while  (st.hasMoreTokens())  {
                        String token  = (String)  st.nextToken();
                        if  (token.equals("<"))  // opening a new HTML tag
                                inTag  =  true;
                        else if  (token.equals(">"))   // ending an HTML tag
                                inTag  =  false;
                        else if  (inTag)  { // we have a opening or closing HTML tag
                            if  ( (token.length() ==  0)  | | (token.charAt(0) !=  ’/’) )
                                tag[count++] =  new  Tag(token,  true);  // opening tag
                            else   // ending tag
                                tag[count++] =  new  Tag(token.substring(1),  false);  // skip the
                        } // Note: we ignore anything not in an HTML tag
                }
     }
     return  tag;  // our array of tags
   }
   /** Tester method */
   public static void  main(String[ ]  args)  throws  IOException  {
                    BufferedReader stdr;               // Standard Input Reader
                    stdr  =  new  BufferedReader(new  InputStreamReader(System.in));
                    HTML tagChecker  =  new  HTML();
     if  (tagChecker.isHTMLMatched(tagChecker.parseHTML(stdr)))
                    System.out.println("The input file is a matched HTML document.");
                    else
                    System.out.println("The input file is not a matched HTML document.");
     }
}
```

Random Access Machine (RAM) Model

- **CPU**

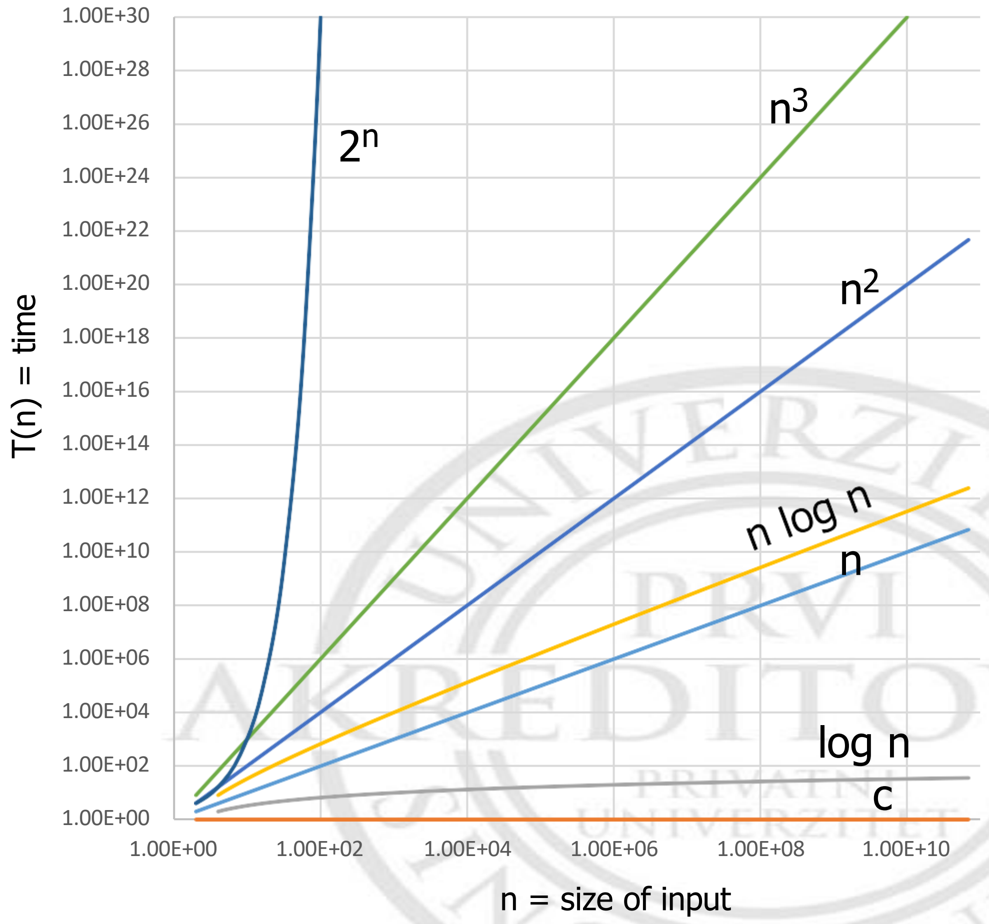


- Potencijalno nepovezana banka memorijskih ćelija, od kojih svaka može da ima proizvoljni broj ili znak

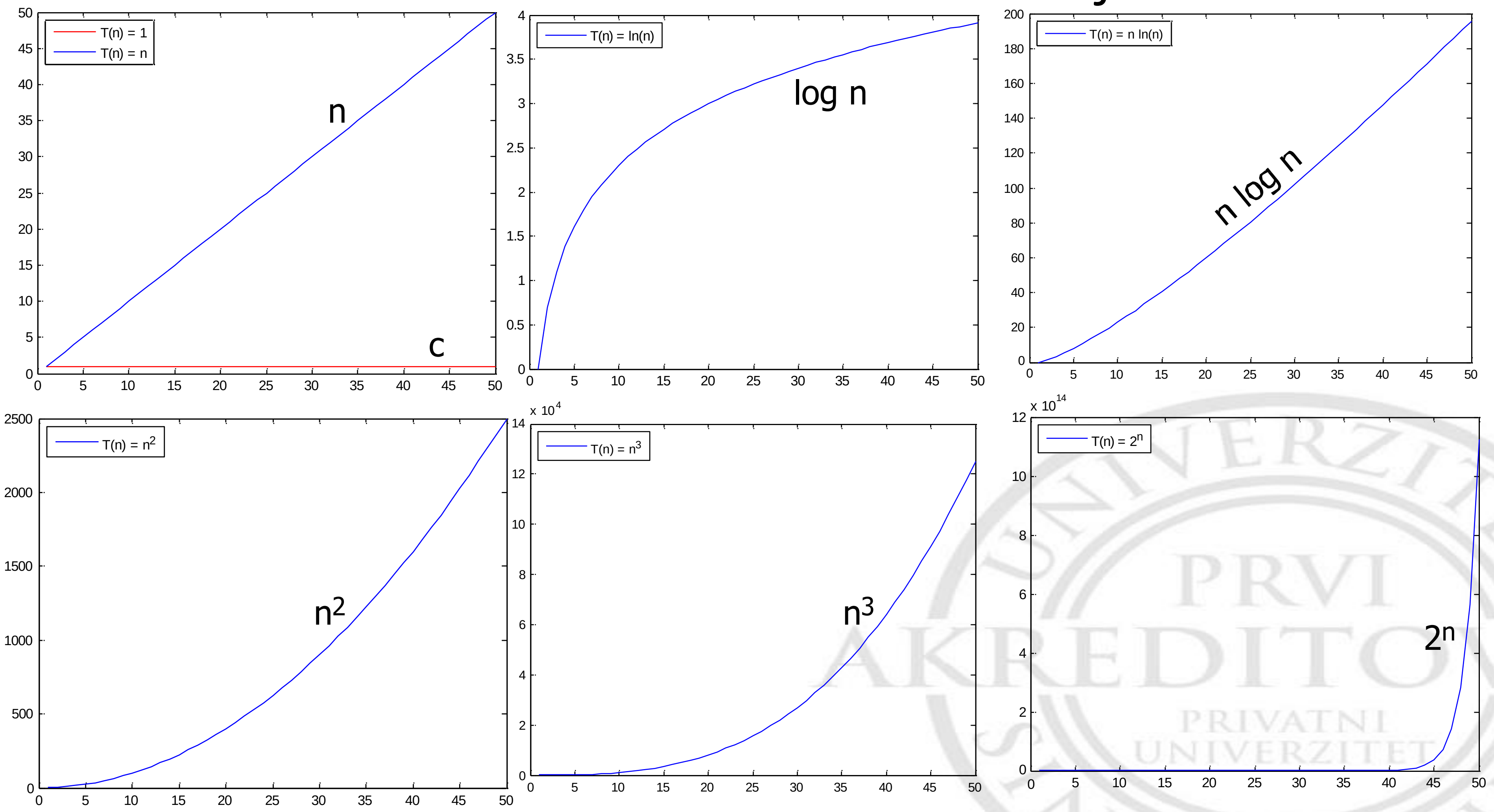
- Memorijske ćelije su numerisane
- Pristup bilo kojoj ćeliji u memoriji zahteva jednu jedinicu vremena

Sedam važnih funkcija

- Sedam funkcija koje se često pojavljuju u analizi algoritma:
  - Constant ≈ $\mathbf{\mathit{1}}$ (or $\mathbf{\mathit{c}}$)
  - Logarithmic ≈ log $\mathbf{\mathit{n}}$
  - Linear ≈ $\mathbf{\mathit{n}}$
  - N-Log-N ≈ $\mathbf{\mathit{n}}$ log $\mathbf{\mathit{n}}$
  - Quadratic ≈ $\mathbf{\mathit{n}}^2$
  - Cubic ≈ $\mathbf{\mathit{n}}^3$
  - Exponential ≈ $\mathbf{2}^n$
- U log-log chartu, nagib linije odgovara stopi rasta funkcije



Sedam funkcija: normalna skala



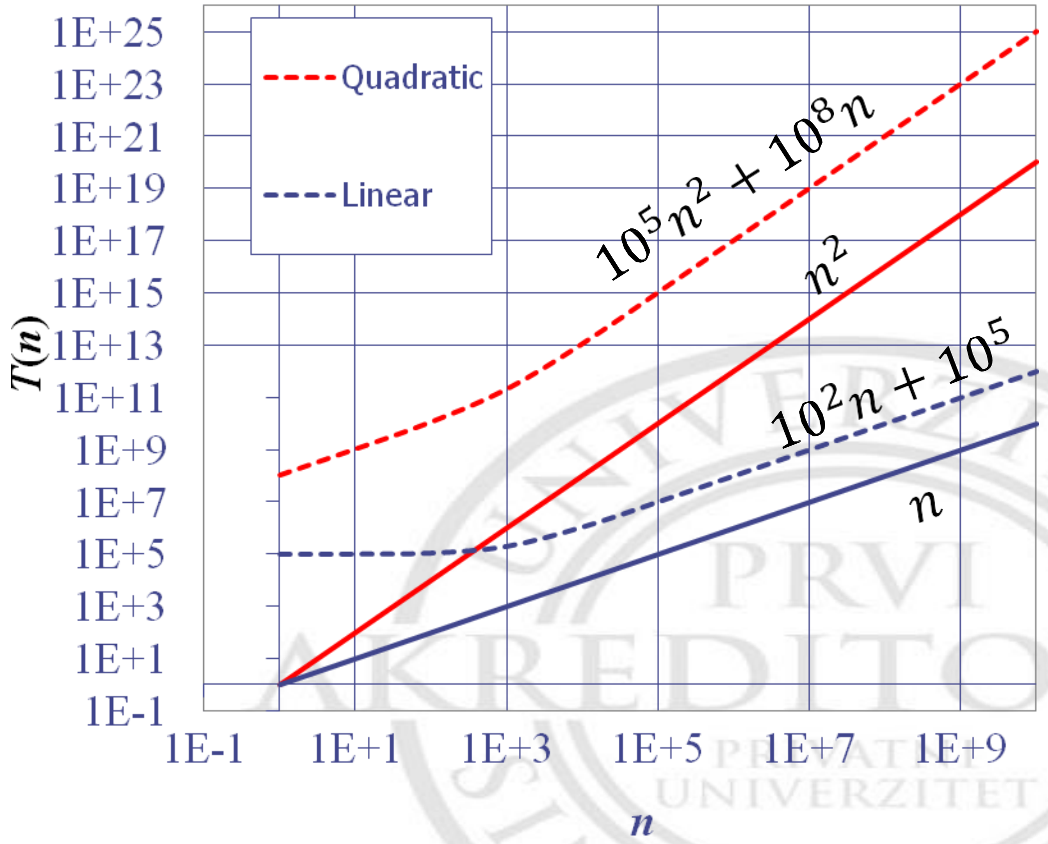
Veličina ulaza

Stopa rasta u brojevima

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84×10^{19}
128	7	128	896	16,384	2,097,152	3.40×10^{38}
256	8	256	2,048	65,536	16,777,216	1.15×10^{77}
512	9	512	4,608	262,144	134,217,728	1.34×10^{154}

Konstantni faktori

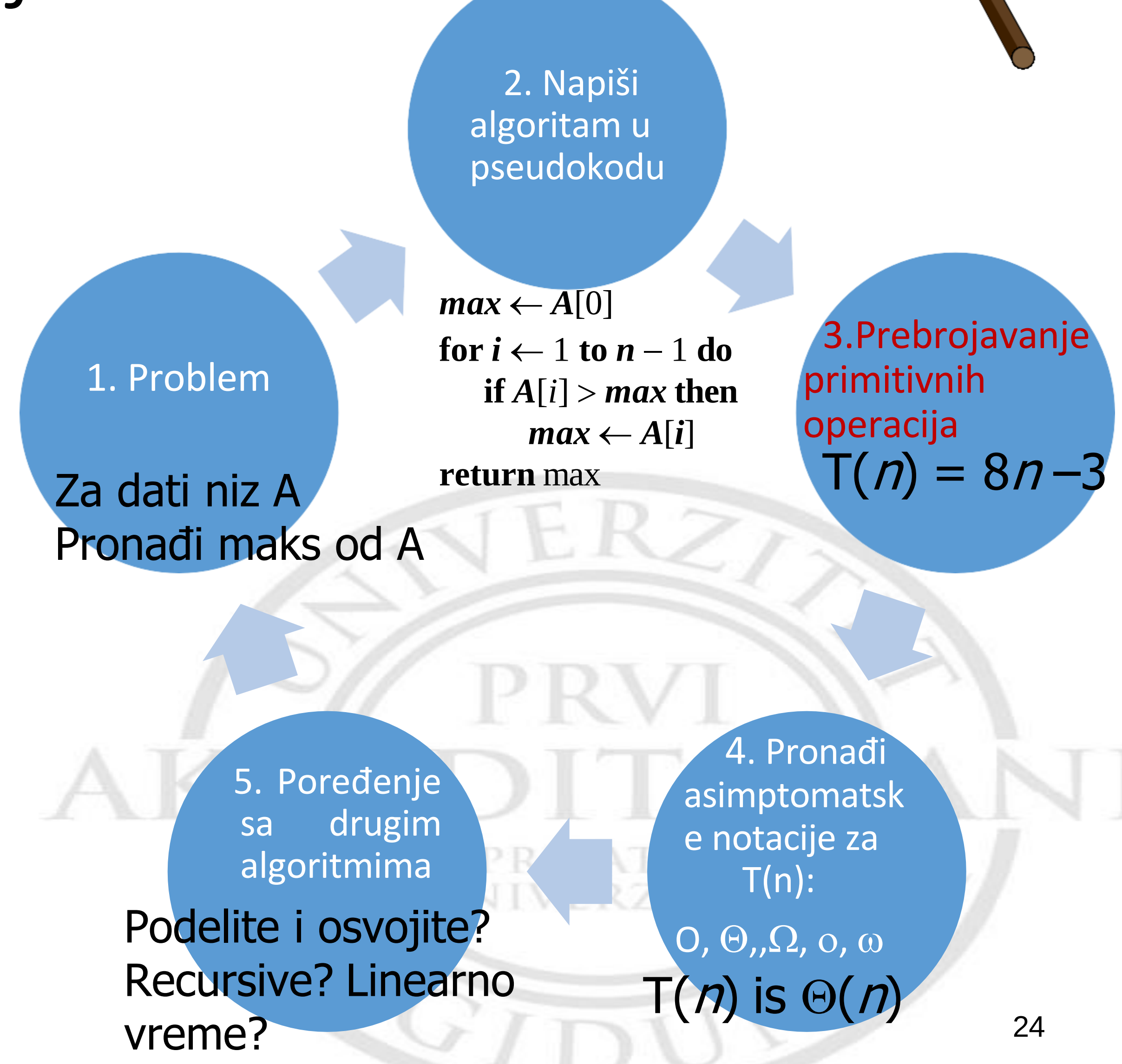
- Stopa rasta nije pogođena
  - Konstantama ili
  - uslovima nižeg reda
- Primeri
  - $10^2\boldsymbol{n} + 10^5$ je linerana funkcija
  - $10^5\boldsymbol{n}^2 + 10^8\boldsymbol{n}$ je kvadratna funkcija
- Pune linije: n i n^2



Teorijska analiza – glavni okvir

Prednosti:

- Koristi opis algoritma visokog nivoa umesto implementacije
- Opisuje vreme izvršavanja kao funkciju veličine unosa, *n*.
- Uzima u obzir sve moguće unose
- Omogućava nam da procenimo brzinu algoritma nezavisno od hardversko/softverskog okruženja



Primitivne operacije

- Izvršene osnovne računarske radnje po algoritmu
- Mogu se identifikovati u pseudokodu
- Uglavnom nezavisne od programski jezika
- Tačna definicija nije važna(videćemo zašto kasnije)

- Iskoristite konstantno vreme u RAM model (jedna jedinica vremena ili jedinica konstantnog vremena ili vremena)

- Primeri:
  - Procena izraza
    $a - 5 + c\sqrt{b}$
  - Dodeljivanje vrednosti promenljivoj
    $a \leftarrow 23$
  - Indeksiranje u nizu
    $A[i]$
  - Pozivanje metoda
    $v.method()$
  - Vraćanje sa metoda
    $return\, a$

Terminologija

- Sledeći uslovi će biti ocenjeni kao ekvivalentni prilikom merenja složenosti vremena ili performansi algoritma:
  - Jedinica vremena ili vremenska jedinica
  - Constant time: O(1) or Θ(1)
  - Broj koraka
  - Broj primitivnih operacija
  - Broj poređenja
  - Broj posećenih čvorova
  - Broj pristupa nizu
  - Broj poziva koji se ponavljaju
  - Broj posećenih vertikala/ivica

Prebrojavanje primitivnih operacija

- Proverom pseudokoda možemo odrediti maksimalan broj primitivnih operacija izvršenih algoritmom, kao funkciju veličine ulaza

Algorithm *arrayMax*(*A*, *n*)	# operations
currentMax ← *A*[0]	2
for *i* ← 1 **to** *n* – 1 **do**	$2n$
if *A*[*i*] > *currentMax* **then**	$2(n-1)$
currentMax ← *A*[*i*]	$2(n-1)$
{ increment counter *i* }	$2(n-1)$
return *currentMax*	1

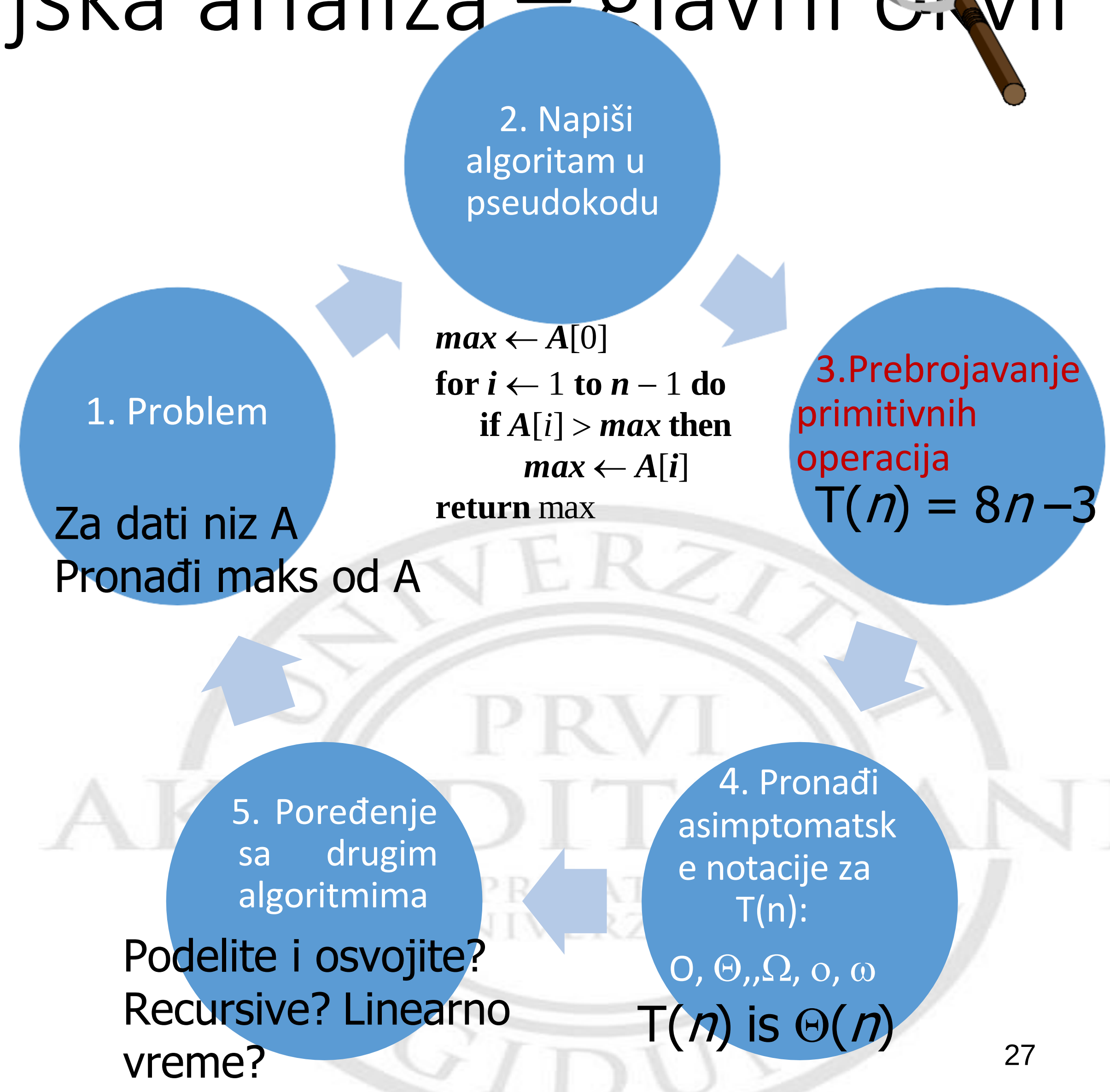
Total: $T(n) = 8n - 3$

konstante se mogu zanemariti (ili zameniti sa 1)

Teorijska analiza – glavni okvir

Prednosti:

- Koristi opis algoritma visokog nivoa umesto implementacije
- Opisuje vreme izvršavanja kao funkciju veličine unosa, *n*.
- Uzima u obzir sve moguće unose
- Omogućava nam da procenimo brzinu algoritma nezavisno od hardversko/softverskog okruženja



Asimptomatska notacija

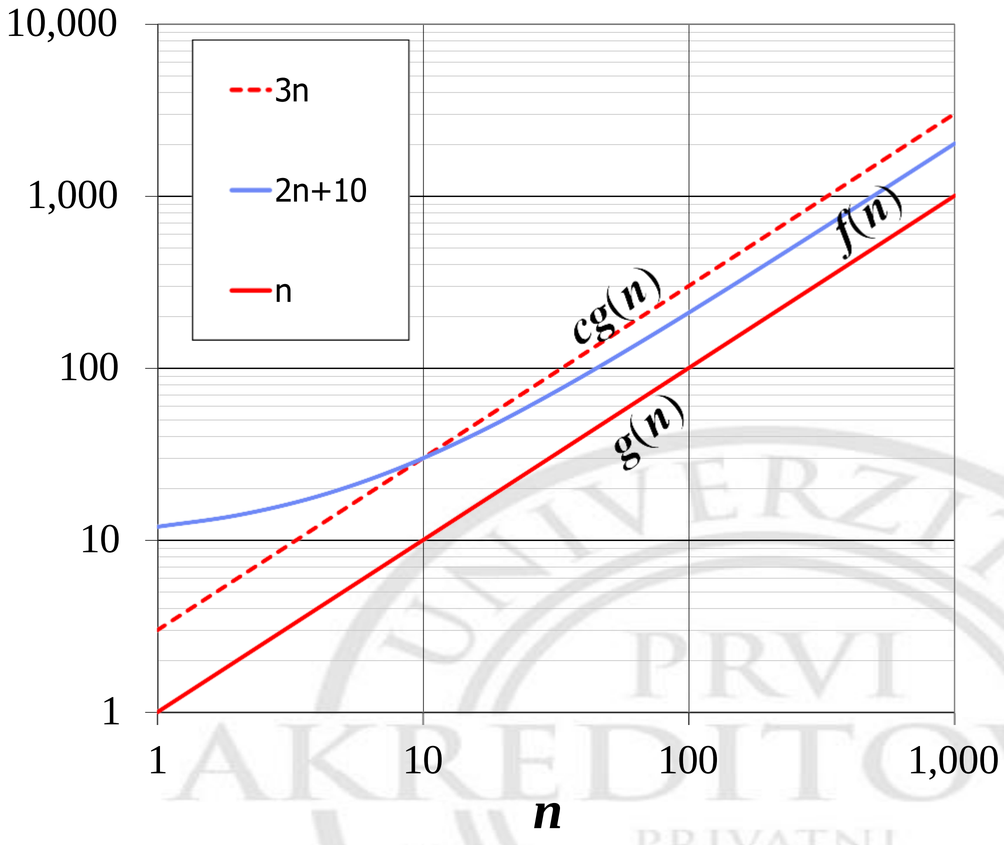
Ime	Notation /korišćena	Neforma Ino Ime	Povezana	Beleške
Big-Oh	$O(n)$	order of	Upper bound – tight	Najčešće korišćena notacija za procenu složenosti algoritma
Big-Theta	$\Theta(n)$		Upper and lower bound – tight	Najtačnija asimptomatska notacija
Big-Omega	$\Omega(n)$		Lower bound – tight	Uglavnom se koristi za određivanje nižih granica problema, a ne algoritama (npr. sortiranje)
Little-Oh	$o(n)$		Upper bound – loose	Koristi se kada je teško nabaviti tesna gornja granica
Little-Omega	$\omega(n)$		Lower bound – loose	Koristi se kada je teško dobiti tesnu donju granicu

Veliko-oh notacija

- Date funkcije $f(n)$ i $g(n)$, mi kažemo $f(n)$ is $O(g(n))$ ako postoje pozitivne konstante
  c i n_0 takve da

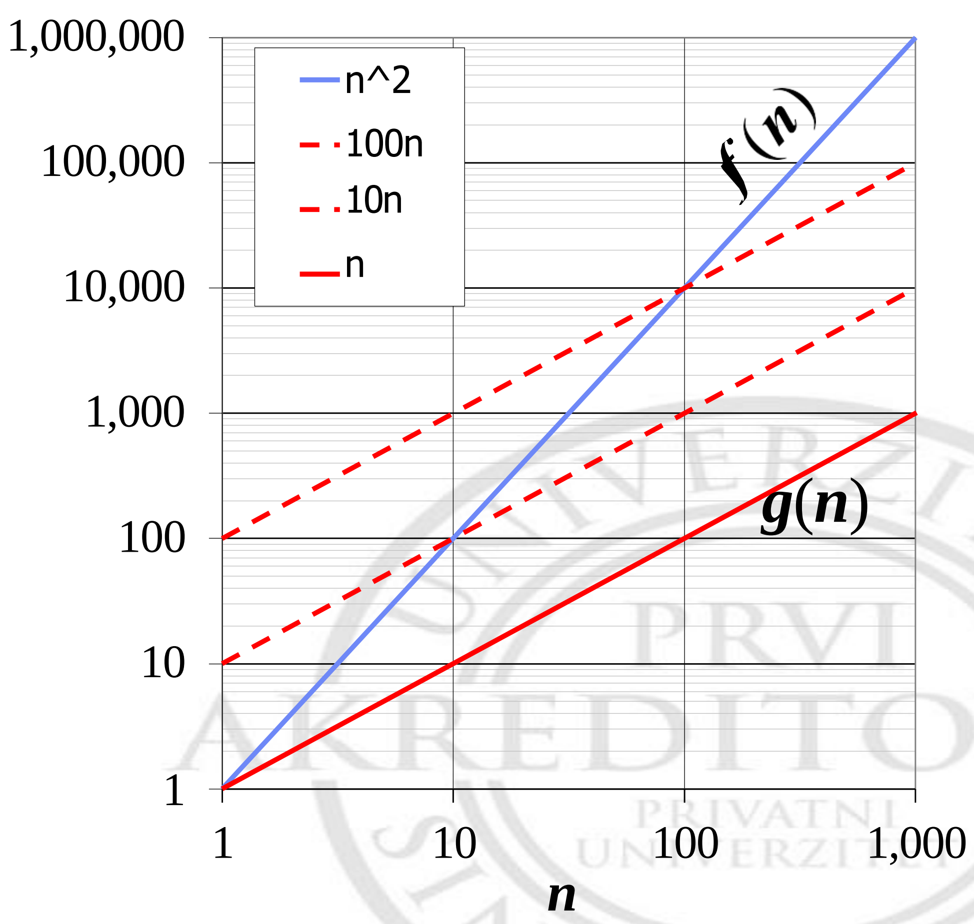
  $$f(n) \leq cg(n) \quad \text{for } n \geq n_0$$

- Primer: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Izaberi $c = 3$ and $n_0 = 10$



Big-Oh primer

- Primer: funkcija n^2 nije $O(n)$
  - $n^2 \leq cn$
  - $n \leq c$
  - Navedena nejednakost se ne može zadovoljiti s obzirom da c mora biti konstanta



Big-Oh i stopa rasta

- Veliko-Oh notacija daje gornju granicu stope rasta funkcije
- Izjava "$f(n)$ is $O(g(n))$" znači da stopa rasta od$f(n)$ nije ništa više od stope rasta $g(n)$
- Možemo da iskoristimo big-Oh napomena o rangiranju funkcija na osnovu njihove stope rasta

	$f(n)$ je $O(g(n))$	$g(n)$ je $O(f(n))$
$g(n)$ raste više	Da	Ne
$f(n)$ raste više	Ne	Da
Isti rast	Da	da

Big-Oh pravila

Ako $\boldsymbol{f(n)}$ je polinom nivoa $\boldsymbol{d}$, onda
$\boldsymbol{f(n)}$ je $\boldsymbol{O(n^d)}$, i.e.,

1. Izbaciti uslove nižeg reda
2. izbaciti konstante

$f(n) = a_0 + a_1 n + a_2 n^2 + a_3 n^3 + \ldots + a_d n^d, \quad a_d > 0$

$\Rightarrow f(n) \leq cn^d$
$\Rightarrow f(n)$ is $O(n^d)$

If $\boldsymbol{d = 0}$, then $\boldsymbol{f(n)}$ is $\boldsymbol{O(1)}$

Big-Oh pravila

- Korišćenje najmanje moguće klase funkcija

  Reći “2n is O(n)” umesto “2n is O(n^2)”

Primer:

$2n + 3 \le cn$ → “2n + 3 je O(n)”

$2n + 3 \le cn^2$ → “2n + 3 je O(n^2)”

$2n + 3 \le cn^3$

$2n + 3 \le cn^{20}$ → “2n + 3 je O(n^{20})”

$2n + 3 \le c\, n \log n$ → “2n + 3 je O(n log n)”

Big-Oh pravila

- Koristite najjednostavniji izraz za klasu

  reći "3n + 5 is O(n)" umesto "3n + 5 is O(3n)"

2n + 3 is O(n) → "2n + 3 is O(n)" (najjed.)

2n + 3 is O(2n + 3)
2n + 3 is O(3n – 2)
2n + 3 is O(100n – 1000000)

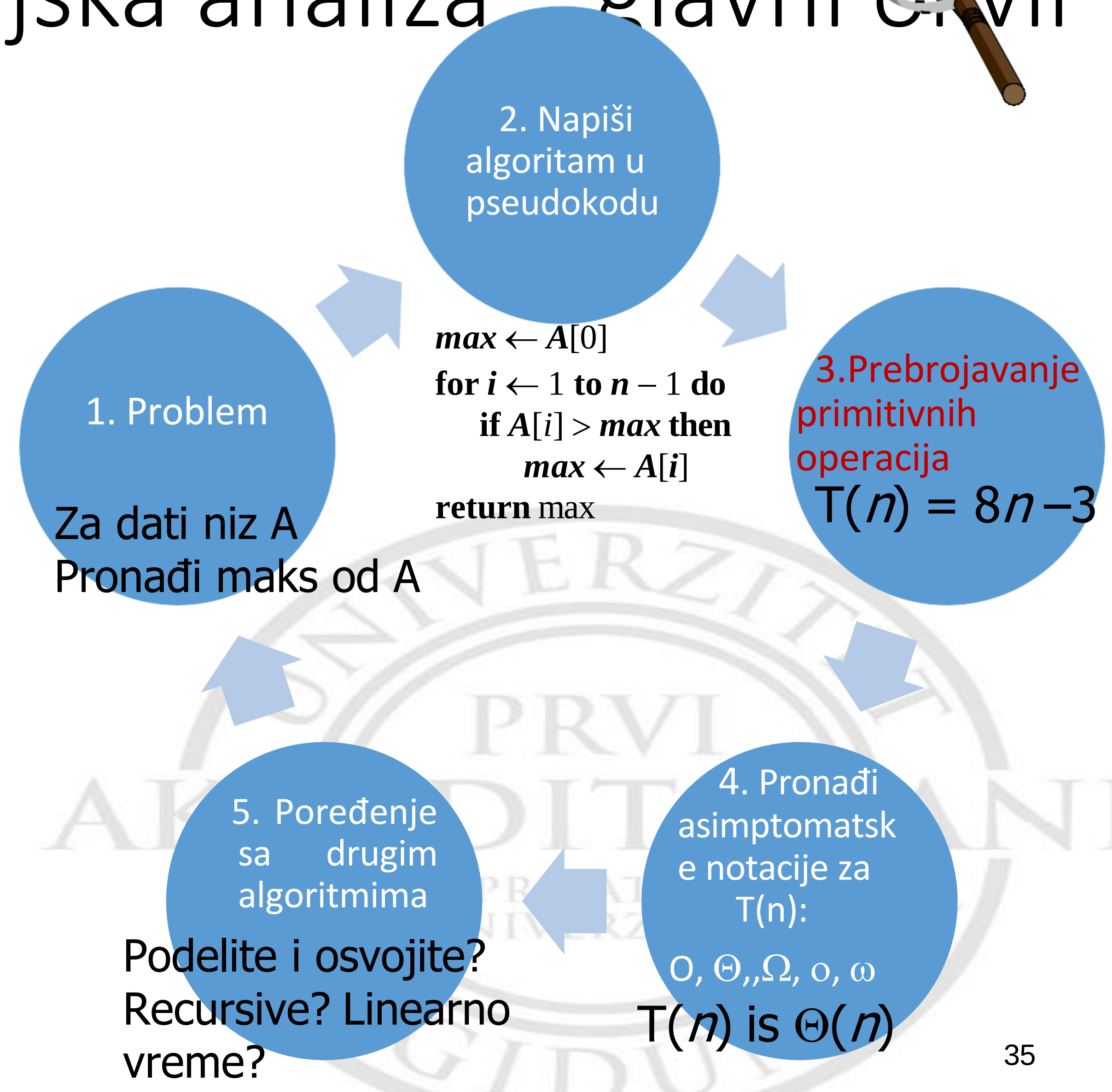
} ok ali nije najednostavnije

3 is O(1) najjednostavnije
3 is O(3) ok ali nije najednostavnije

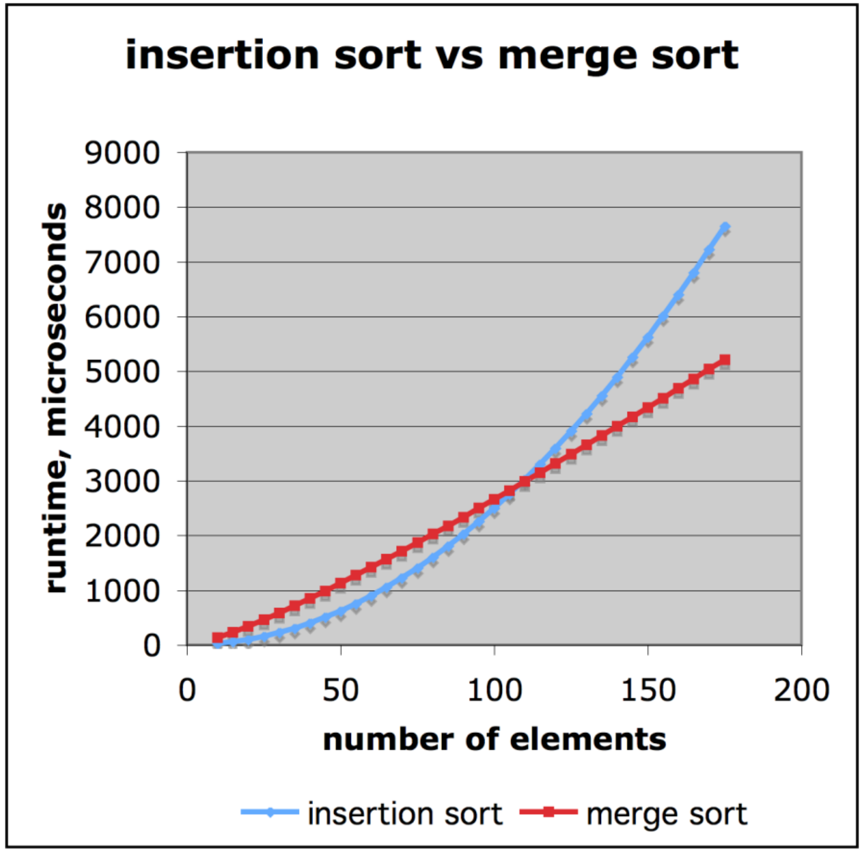
Teorijska analiza – glavni okvir

Prednosti:

- Koristi opis algoritma visokog nivoa umesto implementacije
- Opisuje vreme izvršavanja kao funkciju veličine unosa, *n*.
- Uzima u obzir sve moguće unose
- Omogućava nam da procenimo brzinu algoritma nezavisno od hardversko/softverskog okruženja



Poređenje dva algoritma



Pretpostavimo da je vreme izvršenja:

insertion sort je $n^2 / 4$

merge sort je $2 n \log n$

sort milion stavki?

Za insertion sort je potrebno approx. 70 sati
Dok merge sort uzme approx. 40 sekundi

Ovo je spora mašina, ali ako je 100x brža onda je 40 sekundi vs manje od 0.5 sekundi

Studija slučaja 1: Pretraga u mapi (sortirana lista)

- Problem: Za dati sortirani niz S intedžera (mapa), pronađi ključ k na toj mapi.
- Jedan od najvažnijih problema u rač. nauci
- Rešenje 1: Linearna pretraga
  - Skeniranje elemenata na listi jedan po jedan
  - Dok se ne nađe ključ k
- Primer:

S[*i*]	**8**	**12**	**19**	**22**	**23**	**34**	**41**	**48**
i	0	1	2	3	4	5	6	7

- Linearna pretraga se pokreće u linearno vreme

Algorithm linearSearch(S, *k, n*):
Input: Sorted array S of size *n*, and key *k*
Output: Null ili pronađeni element

	# op. (more accurate)	*# op.* (simplified)
$i \leftarrow 0$	1	1
while $i < n$ **and** $S[i] != k$	3n	2n
$\quad i \leftarrow i + 1$	2(n-1)	n
if $i = n$ **then**	1	1
$\quad$ **return null**	1	1
else		
$\quad e \leftarrow S[i]$	2	1
$\quad$ **return** *e*	1	1
Total:		T(n) = 3n + 4

Biće izvršena jedna od dve grane - računati max

Najgori slučaj running time: T(n) = 3n + 4 ➔ T(n) is O(n)

Studija slučaja 1: Pretraga u nizu (sortirana lista)

- Problem: Za dati sortirani niz S intedžera, pronađi indeks za vrednost k u nizu.

- Rešenje 1: Linearna pretraga
  - Skeniranje elemenata na listi jedan po jedan
  - Dok se ne nađe indeks za k

- Primer:

S[*i*]	8	12	19	22	23	34	41	48
i	0	1	2	3	4	5	6	7

- Linearna pretraga se pokreće u linearno vreme

Algorithm linearSearch(S, *k, n*):

Input: Sorted array S of size *n*, and value *k*

Output: Null ili pronađeni index

	# op. (more accurate)	*# op.* (simplified)
$i \leftarrow 0$	1	1
while $i < n$ **and** $S[i] != k$	3n	2n
$\quad i \leftarrow i + 1$	2(n-1)	n
if $i = n$ **then**	1	1
$\quad$ **return null**	1	1
else		
$\quad e \leftarrow i$	2	1
$\quad$ **return** *e*	1	1
Total:		T(n) = 3n + 4

Biće izvršena jedna od dve grane - računati max

Najgori slučaj running time: T(n) = 3n + 4 ➔ T(n) is O(n)

Studija slučaja 1: Pretraga u nizu (sortirana lista)

- Problem: Za dati sortirani niz intedžera, pronađi index za ključ k
- Rešenje 2: Binarna pretraga
- Binarna pretraga se pokreće u logaritamsko vreme
- Isti problem:
  - Dva algoritma imaju različito vreme izvršavanja

S[*i*]	8	12	19	22	23	34	41	48
i	0	1	2	3	4	5	6	7

Algorithm binarySearch(S, *k*):

Input: Vrednost *k*

Output: Null ili pronađeni ključ

```
def binary_search(arr, k):
    low, high = 0, len (arr) - 1

    while low <= high:
        mid = (low + high) / 2
        if arr[mid] == k:
            return mid  // Found the key, return its index
        elif arr[mid] < k:
            low = mid + 1  // Search in the right half
        else:
            high = mid - 1  // Search in the left half

    return -1  // Key not found

Primer korišćenja
sorted_array = [8, 12, 19, 22, 23, 34 41, 48]
key = 22
index = binary_search(sorted_array, key)
print("Index:", index)  // Output: Index: 3
```


Najgori slučaj vreme izvršavanja: $T(n) = T(n/2) + 1$ ➔ $T(n)$ je $O(\log n)$

Studija slučaja 1: Pretraga u mapi (sortirana lista)

- Problem: Za dati sortirani niz intedžera (mapa), pronađi ključ k u toj mapi
- Rešenje 2: Binarna pretraga
- Binarna pretraga se pokreće u logaritamsko vreme
- Isti problem:
  - Dva algoritma imaju različito vreme izvršavanja

S[*i*]	8	12	19	22	23	34	41	48
i	0	1	2	3	4	5	6	7

Algorithm binarySearch(S, *k*, low, high):
Input: A key *k*
Output: Null ili pronađeni elementl
if low > high **then**
        return null
else
    mid ← ⌊(low + high) / 2⌋
    e ← S[mid]
    if *k* = *e*.getKey() **then**
        return *e*
    else if *k* < *e*.getKey() **then**
            return binarySearch(S, *k*, low, mid-1)
      else
            return binarySearch(S, *k*, mid+1, high)

Najgori slučaj vreme izvršavanja: T(n) = T(n/2) + 1 ➔ T(n) je O(log n)

Studija slučaja 2: Prefiks prosečnosti

- ***i***-ti prefix proseka niza ***S*** je prosek prvih $(\boldsymbol{i} + 1)$ elemenata od ***S***:

$$\boldsymbol{A}[\boldsymbol{i}] = (\boldsymbol{S}[0] + \boldsymbol{S}[1] + \ldots + \boldsymbol{S}[\boldsymbol{i}])/(\boldsymbol{i}+1)$$

- Problem: Izračunaj niz ***A*** proseka prefiksa drugog niza ***S***
- Ima aplikacija u finansijskoj analizi
- Rešenje 1: Algoritam kvadratičkog vremena: quadPrefixAve
- Primer:

S

21	23	25	31	20	18	16
0	1	2	3	4	5	6

A

21	22	23	25	24	23	22
0	1	2	3	4	5	6

Algorithm quadPrefixAve(*S*, *n*)

Input: niz *S* od *n* intedžera

Output: niz *A* od prefiks proseka od *S*

	#operacija
A ← new array of *n* integers	n
for *i* ← 0 **to** *n* - 1 **do**	n
s ← *S*[0]	n – 1
for *j* ← 1 **to** *i* do	1 + 2 + …+ (n - 1)
s ← *s* + *S*[*j*]	1 + 2 + …+ (n - 1)
A[*i*] ← *s* / (*i* + 1)	n - 1
return *A*	1

$$\frac{n(n-1)}{2}$$

$T_2(n) = 2n + 2(n\text{-}1) + 2n(n\text{-}1)/2 + 1$ is $O(n^2)$

Studija slučaja 2: Prefiks prosečnosti

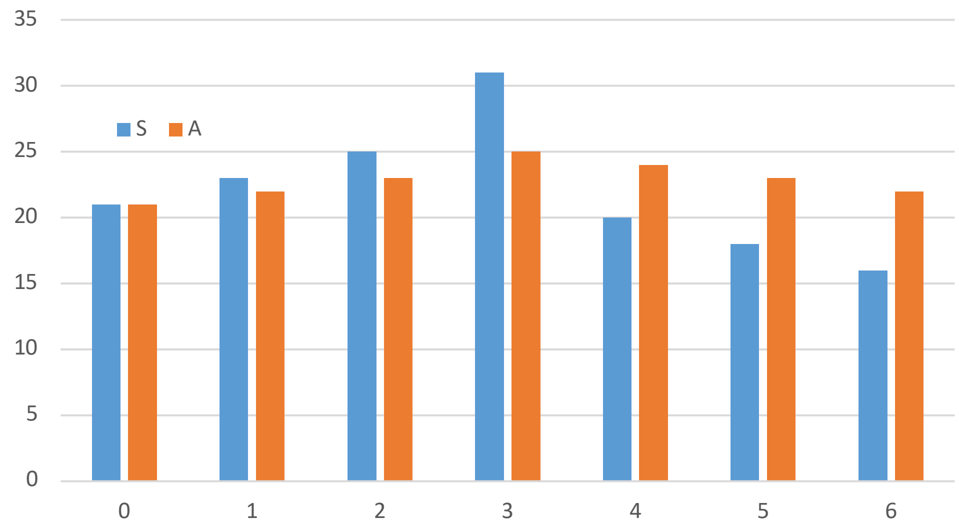
- Rešenje 2: linearan vremenski algoritam: linearPrefixAve
- Za svaki element koji se skenira zadržati tekuću sumu

S

21	23	25	31	20	18	16
0	1	2	3	4	5	6

A

21	22	23	25	24	23	22
0	1	2	3	4	5	6



Algorithm linearPrefixAve(*S*, *n*)

Input: niz *S* od *n* intedžera

Output: niz *A* proseka prefiksa od *S*

	#operacije
$A \leftarrow$ new array of n integers	n
$s \leftarrow 0$	1
for $i \leftarrow 0$ **to** $n - 1$ **do**	n
$\quad s \leftarrow s + S[i]$	n – 1
$\quad A[i] \leftarrow s / (i + 1)$	n – 1
return A	1

$T_2(n) = 4n$ is $O(n)$

Studija slučaja 3: Maximum susednih podsekv. suma (MCSS)

- Problem:
  - Dato: niz celih brojeva (moguće negativnih) $A = A_1, A_2, \dots, A_n$
  - Pronaći: max vrednost od $\sum_{k=i}^{j} A_k$
  - Ako su svi celi brojevi negativni MCSS je 0
- Primer:
  - Za A = -3, 10, -2, 11, -5, -2, 3 MCSS is 19
  - Za A = -7, -10, -1, -3 MCSS is 0
  - Za A = 12, -5, -6, -4, 3 MCSS is 12
- Razni algoritmi rešavaju isti problem
  - Cubic time
  - Quadratic time
  - Divide and conquer
  - Linear time

MCSS: Cubic vs quadratic time algoritmi

Algorithm cubicMCSS(*A*,*n*)

Input: niz celih brojeva *A* dužine *n*

Output: vrednost MCSS

maxS ← 0

for *i* ← 0 **to** *n* – 1 **do for**

j ← *i* **to** *n* – 1 **do**

curS ← 0

for *k* ← *i* **to** *j* **do**

curS ← curS +*A*[*k*]

if *curS* > *maxS*

maxS ← *curS*

return *maxS*

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=i}^{j} 1 = \frac{n^3 + 3n^2 + 2n}{6}$$

T(n) = $\frac{n^3+3n^2+2n}{6}$ + *c* *je* $O(n^3)$

Algorithm quadraticMCSS(*A*,*n*)

Input: niz celih brojeva *A* dužine *n*

Output: vrednost MCSS

maxS ← 0

for *i* ← 0 **to** *n* – 1 **do**

for *j* ← *i* **to** *n* – 1 **do**

curS ← *curS* + *A*[*j*]

if *curS* > *maxS*

maxS ← *curS*

$$\propto \frac{n(n-1)}{2}$$

return *maxS*

Dvostruka suma će dati $O(n^2)$

Primer: Za A = -3, 10, -2, 11, -5, -2, 3 MCSS is 19

MCSS: Podeli i osvoji

- Glavne karakteristike:
  - Popriličnu duga
  - Razdvajanje niza na dva dela

Prva polov. Druga polov.

A = -3, 10, -2, 11, -1, 2, -3

MCSS se prostire na obe polovine

- Algoritam:
  - Podeliti podsekvenciju na dve polovine
  - Pronaći maksimalni levi zbir ivice (strelica nalevo)
  - Pronaći maksimalni zbir desne ivice (strelica nadesno)
  - Vratitu zbir oba maksimuma kao maksimalne sume
  - Uradite ovo ponavljajući za svaku polovinu
  - Data kompleksnost T(n) = 2T(n/2) + n, where T(1) = 1
  - Izvršava se u O(n log n)

Linearni vremenski algoritam

- Nezgodni delovi ovog algoritma su:
- Ni jedan MCSS neće **početi** ili se **završiti** sa negativnim brojem
  - Nalazimo samo vrednost MCSS-a
  - Ali ako nam treba prava podsekvencija, moraćemo bar da pribegnemo podeli i osvoji

Algorithm linearMCSS(*A*,*n*)

Input: niz intedžera *A* dužine *n*

Output: vrednost MCSS

$maxS \leftarrow 0$; $curS \leftarrow 0$

for $j \leftarrow 0$ **to** $n - 1$ **do**

  $curS \leftarrow curS + A[j]$

  if $curS > maxS$

    $maxS \leftarrow curS$

  else

    if $curS < 0$

      $curS \leftarrow 0$

return *maxS*

Jedna for petlja daje $O(n)$

Primer: Za A = -3, 10, -2, 11, -5, -2, 3 MCSS je 19

Big-Oh rođaci

- **big-Oh**

  f(n) je O(g(n)) ako postoji constant $c > 0$ i integer constant $n_0 \geq 1$ tako da

  $$f(n) \leq c\, g(n) \text{ for } n \geq n_0$$

- **big-Omega**
  - f(n) is $\Omega(g(n))$ ako postoji constant $c > 0$ i integer constant $n_0 \geq 1$ tako da

    $$f(n) \geq c\, g(n) \text{ for } n \geq n_0$$

- **big-Theta**
  - f(n) is $\Theta(g(n))$ ako postoje constante $c' > 0$ i $c'' > 0$ i integer constant $n_0 \geq 1$ tako da

    $$c'\, g(n) \leq f(n) \leq c''\, g(n) \text{ for } n \geq n_0$$

Važna teorema:

$$f(n) \text{ je } O(g(n)) \quad \text{i} \quad \Omega(g(n)) \Leftrightarrow f(n) \text{ je } \Theta(g(n))$$

Intuicija za asimptotsku notaciju

Big-Oh

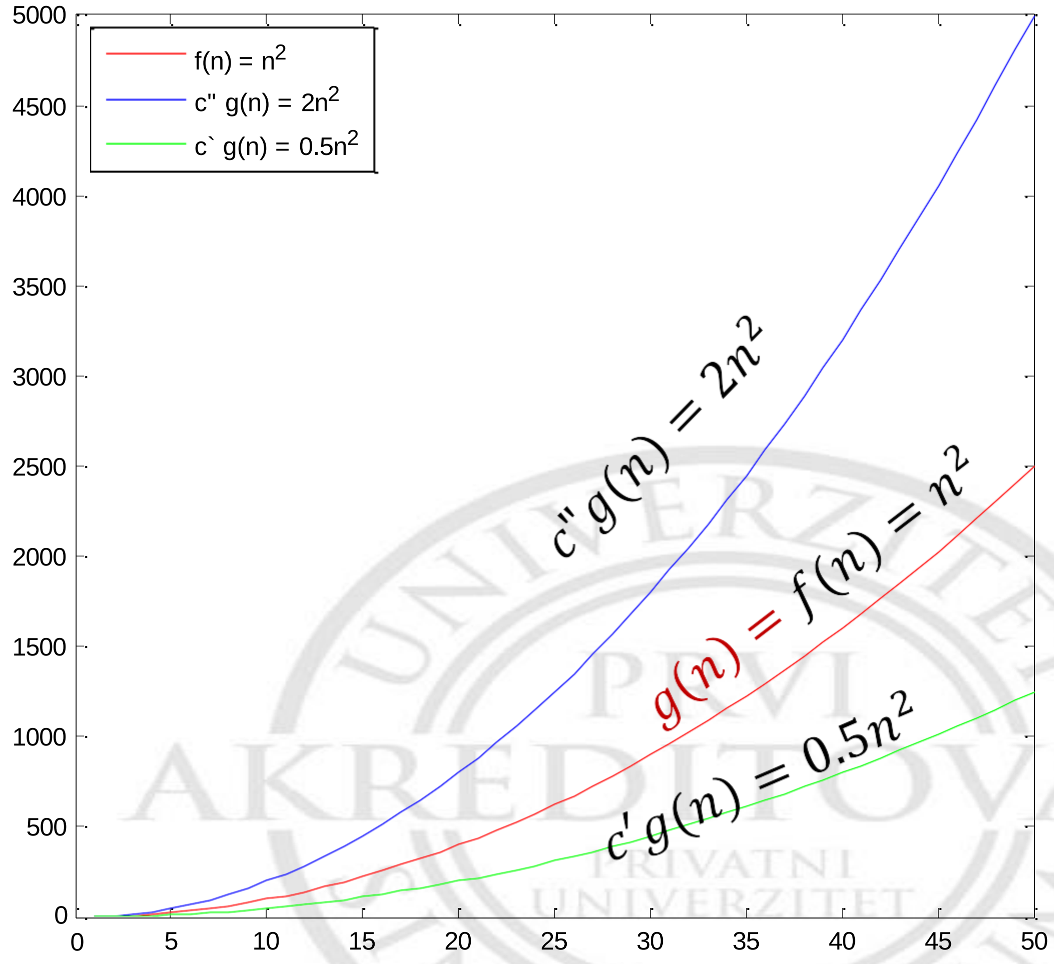
- f(n) je O(g(n)) ako f(n) je asymptotically **manje od ili jednako** g(n)

big-Omega

- f(n) je Ω(g(n)) ako f(n) je asymptotically **veći od ili jednak** g(n)

big-Theta

- f(n) je Θ(g(n)) ako f(n) je asymptotically **jednako** g(n)



Primer: korišćenje Big-Oh rođaka

- **5n^2 je $\Omega(n^2)$**

  $f(n)$ je $\Omega(g(n))$ ako postoji konstanta $c > 0$ i intedžer konstanta $n_0 \geq 1$ tako da $f(n) \geq c\, g(n)$ for $n \geq n_0$

  Neka je $c = 5$ i $n_0 = 1$. Onda,

  $$5n^2 \geq cn^2 \;\; \forall n \geq 1, \qquad c = 5$$

- **5n^2 je $\Omega(n)$**

  $f(n)$ is $\Omega(g(n))$ ako postoji konstanta $c >$ i intedžer konstanta $n_0 \geq 1$ tako da $\mathrm{f}(n) \geq c\, g(n)$ for $n \geq n_0$

  Neka je $c = 1$ i $n_0 = 1$. Onda,

  $$5n^2 \geq cn \;\; \forall n \geq 1, \qquad c = 1$$

Oba su tačna, ali mi više volimo da koristimo **5n^2 je $\Omega(n^2)$**

Primer: korišćenje Big-Oh rođaka

- **$5n^2$ je $\Theta(n^2)$**

  $f(n)$ je $\Theta(g(n))$ ako je $\Omega(n^2)$ i $O(n^2)$. Već smo ga našli za $\Omega(n^2)$.

  Za $O(n^2)$, setiti se da $f(n)$ je $O(g(n))$ ako postoji konstanta $c > 0$ i intedžer konstanta $n_0 \geq 1$ tako da $f(n) \leq c\, g(n)$ za $n \geq n_0$

  Neka je $c = 5$ i $n_0 = 1$

$\Omega(n^2)$

$$f(n) = 5n^2 \geq cn^2, \qquad \forall n \geq 1 \text{ i } c=5$$

$O(n^2)$

$$f(n) = 5n^2 \leq cn^2, \qquad \forall n \geq 1 \text{ i } c=5$$

f(n) je $\Omega(n^2)$ **i** f(n) je $O(n^2)$ $\Leftrightarrow$ f(n) je $\Theta(n^2)$

Matematika koju treba da pregledate

- Serije
  - Aritmetika/geometrija
- Algebra
- Logaritami i eksponenti
  - Logaritmi će biti baze 2, osim ako nije drugačije navedeno
  - Zašto?

$$\log_c n = \frac{\log_2 n}{\log_2 c} = \frac{1}{\log_2 c}\log_2 n$$

  - *c* je konstanta, kao i $\frac{1}{\log_2 c}$
- Tehnike dokazivanja
- Osnovna verovatnoća

- **svojstva logaritama:**

  $\log_b(xy) = \log_b x + \log_b y$

  $\log_b (x/y) = \log_b x - \log_b y$

  $\log_b x^a = a \log_b x$

  $\log_b a = \log_x a/\log_x b$

- **svojstva eksponencijalnih:**

  $a^{(b+c)} = a^b a^c$

  $a^{bc} = (a^b)^c$

  $a^b / a^c = a^{(b-c)}$

  $b = a^{\log_a b}$

  $b^c = a^{c \log_a b}$

Pod i plafon

Pod:

$\lfloor x \rfloor$ = najveći intedžer i $\leq x$

Plafon:

$\lceil x \rceil$ = najmanji intedžer i $\geq x$

Primeri:

$\lfloor 3.4 \rfloor = 3$ $\lceil 3.4 \rceil = 4$

$\lfloor 3.0 \rfloor = 3$ $\lceil 3.0 \rceil = 3$

$\lfloor 3.99999 \rfloor = 3$ $\lceil 3.99999 \rceil = 4$

For Loop:

For $i \leftarrow 0$ to $n-1$	$n+1$	
$Q \leftarrow Q + S$	n	$O(n)$
For $i \leftarrow n$ down to 1	$n+1$	
$Q \leftarrow Q^3$	n	$O(n)$
For $i \leftarrow 1$ to n step 2	$\frac{n}{2}+1$	
$Q \leftarrow Q^2$	$\frac{n}{2}$	$O(n)$
For $i \leftarrow 1$ to n	$\frac{n}{2}+1$	
$i \leftarrow i + 1$	$\frac{n}{2}$	$O(n)$

For Loop:

$$\begin{array}{lll} \text{For } i \leftarrow 1 \text{ to } \frac{n}{2} & \frac{n}{2}+1 & \\ \quad Q \leftarrow Q+1 & \frac{n}{2} & O(n) \end{array}$$

$$\begin{array}{lll} \text{For } i \leftarrow 1 \text{ to } \frac{n}{4} & \frac{n}{4}+1 & \\ \quad Q \leftarrow Q-1 & \frac{n}{4} & O(n) \end{array}$$

$$\begin{array}{lll} \text{For } i \leftarrow 1 \text{ to } \frac{n}{100} & \frac{n}{100}+1 & \\ \quad Q \leftarrow Q+1 & \frac{n}{100} & O(n) \end{array}$$

$$\begin{array}{lll} \text{For } i \leftarrow 1 \text{ to } 4n & 4n+1 & \\ \quad Q \leftarrow Q+1 & 4n & O(n) \end{array}$$

For Loop:

For $i \leftarrow 1$ to $\log n$
$Q \leftarrow Q + 1$

$\log n + 1$
$\log n$
$O(\log n)$

For $i \leftarrow 1$ to $n \log n$
$Q \leftarrow Q - 1$

$(n \log n) + 1$
$n \log n$
$O(n \log n)$

For $i \leftarrow 1$ to n^2
$Q \leftarrow Q - 1$

$n^2 + 1$
n^2
$O(n^2)$

For $i \leftarrow 1$ to n^3
$Q \leftarrow Q - 1$

$n^3 + 1$
n^3
$O(n^3)$

Primeri: Petlje

For Loop:

$$
\begin{array}{lll}
\text{For } i \leftarrow 1 \text{ to } n & n+1 & \\
\quad \text{for } j \leftarrow 1 \text{ to } n & n(n+1) & O(n^2) \\
\quad\quad Q \leftarrow Q+1 & n^2 &
\end{array}
$$

$$
\left.
\begin{array}{ll}
\text{For } i \leftarrow 1 \text{ to } n & n+1 \\
\quad \text{For } j \leftarrow 1 \text{ to } n & n(n+1) \\
\quad\quad \text{For } k \leftarrow 1 \text{ to } n & n^2(n+1) \\
\quad\quad\quad Q \leftarrow Q+1 & n^3
\end{array}
\right\} O(n^3)
$$

Primeri: Petlje

For Loop:

$\text{For } i \leftarrow 1 \text{ to } n$

$\quad \text{For } j \leftarrow i \text{ to } n$

$\quad Q \leftarrow Q + 1$

n

$$1 + 2 + \cdots + n + (n+1) = \frac{(n+1)(n+2)}{2}$$

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

$$O(n^2)$$

$\text{For } i \leftarrow 1 \text{ to } n$

$\quad \text{For } j \leftarrow i \text{ to } n$

$\quad\quad \text{For } k \leftarrow j \text{ to } n$

$\quad\quad Q \leftarrow Q + 1$

n

$$1 + 2 + \cdots + n$$

$$1 + (1+2) + (1+2+3) + \cdots + (1+2+\cdots+n)$$

$$\sum_{i=1}^{n} \sum_{j=i}^{n} j$$

$$O(n^3)$$

$n \quad n$

Primeri: While Petlje

$$\left.\begin{array}{ll} i \leftarrow 1 & \\ \text{while} i \leq n \text{do} & n+1 \\ i \leftarrow i+1 & n \end{array}\right\} O(n)$$

$$\left.\begin{array}{ll} i \leftarrow 1 & \\ \text{while} i \leq \frac{n}{2} \text{do} & \frac{n}{2}+1 \\ i \leftarrow i+1 & \frac{n}{2} \end{array}\right\} O(n)$$

$$\left.\begin{array}{ll} i \leftarrow 1 & \\ \text{while} i \leq n \text{do} & (\log n)+1 \\ i \leftarrow i * 2 & \frac{\log n}{2} \end{array}\right\} O(\log n)$$

Primeri: While Petlje

For while: koristimo “n” and “log n” umesto “n+1” ili “(log n) +1”

$$\left.\begin{array}{cc} i \leftarrow 1 & 1 \\ \text{while} i < n \text{do} & \log n \\ i \leftarrow i * 2 & \log n \\ \text{for} j \leftarrow 1 \text{to } n & n \log n \\ Q \leftarrow Q + 1 & n \log n \end{array}\right\} O(n \log n)$$

$$\left.\begin{array}{cc} \text{for} i \leftarrow 1 \text{to} n & n \\ j \leftarrow 1 & n \\ \text{while} j < i \text{do} & \sum_{i=1}^{n} \log i \\ j \leftarrow j * 2 & \sum_{i=1}^{n} \log i \end{array}\right\} O(n \log n)$$

$$\log n! \leq \log n^n = O(n \log n)$$

Examples: Worst vs Best case

If rečenice:

Najgori slučaj: Uzmi MAX od dve grane

$$\begin{array}{ll} \text{If } a > 0 & 1 \\ \quad a \leftarrow 2 & 1 \text{ ova grana} = 2\text{op.} \\ \text{else} & \\ b \leftarrow Q & \left.\begin{array}{l}1\\1\end{array}\right\} \text{ova grana} = 3\text{op.} \\ Q \leftarrow 3 & \\ & \text{Max} = 3 = O(1) \end{array}$$

Najgori = Najbolji slučaj: O(1)

$$\begin{array}{ll} \text{if} a > 0 & 1 \\ \text{for} i \leftarrow 1 \text{to} n & \left.\begin{array}{l}n\\n\end{array}\right\} \text{this branch} = 2n + 1\text{op.} \\ Q \leftarrow Q + 1 & \\ \text{else} & \\ Q \leftarrow 3 & 1\}\text{this branch} = 2\text{op.} \\ & \text{Max} = 2n + 1 = O(n) \end{array}$$

Najgori slučaj: uzeti MAX: O(n)
Najbolji slučaj: uzeti MIN: O(1)

Najgori vs najbolji slučaj

Loops:

Najgori slučaj: uzeti MAX koraka od while petlje
Najbolji slučaj: uzeti MIN

	najgori	najbolji
i ← 0	1	1
while **i < n** and **A[i] != 7**	n	1
i ← i + 1	n	0
	O(n)	**O(1)**

Najgori:

3	1	4	2	3	2	1	8	
0	1	2	3	4	5	6	7	**n**

Najbolji:

7	1	5	4	8	2	1	9	
0	1	2	3	4	5	6	7	**n**

Big-Oh primeri

- $7n - 2$
  - $7n - 2$ je $O(n)$
  - Treba $c > 0$ i $n_0 \geq 1$ tako da $7n-2 \leq cn$ za $n \geq n_0$
- Evaluation for $c = 7$, $n_0 = 1$:

$$7n - 2 \leq cn$$
$$7n - 2 \leq 7n, \forall n \geq 1$$

- $3n^3 + 20n^2 + 5$
  - $3n^3 + 20n^2 + 5$ is $O(n^3)$
  - Treba $c > 0$ i $n_0 \geq 1$ tako da $3n^3 + 20n^2 + 5 \leq cn^3$ za $n \geq n_0$
- Eavluacija za for $c = 4$ i $n_0 = 21$:

$21n^2 \leq n^3$ za sve $n \geq 21$

$$3n^3 + 20n^2 + 5 \leq 4n^3$$
$$20n^2 + 5 \leq n^3$$
$$20n^2 + 5 \boxed{\leq 20n^2 + n^2 \leq n^3, \forall n} \geq 21$$

Big-Oh Primeri

- 3 log n + 5
  - 3 log n + 5 is O(log n)
  - Treba c > 0 i $n_0 \geq 1$ tako da 3 log n + 5 ≤ c log n za $n \geq n_0$
- Evaluacija za c = 8, n_0 = 2:

$$3\log n + 5 \leq c\log n$$
$$5 \leq c\log n - 3\log n$$
$$5 \leq (c-3)\log n$$
$$5 \leq 5\log n\,,\ \text{when}\ c\ =\ 8, n_0$$
$$= 2$$

Recap - asimptotska notacija

Name	Notation /use	Informal name	Bound	Notes
Big-Oh	$O(n)$	order of	Upper bound – tight	Najčešće korišćena notacija za procenu složenosti algoritma
Big-Theta	$\Theta(n)$		Upper and lower bound – tight	Najtačnija asimptomatska notacija
Big-Omega	$\Omega(n)$		Lower bound – tight	Uglavnom se koristi za određivanje nižih granica problema, a ne algoritama (npr. sortiranje)
Little-Oh	$o(n)$		Upper bound – loose	Koristi se kada je teško nabaviti tesna gornja granica
Little-Omega	$\omega(n)$		Lower bound – loose	Koristi se kada je teško dobiti tesnu donju granicu

Dokazi – Opravdanja

Tipovi dokaza:

- Direktno
- Indirektni
- Kontraeksample
- Kontradikcija (kontrapozitivna)
- Indukcija
- Loop invariant

Opravdanja za O, Ω, Θ

- Obično direktan dokaz
- Kontra primer koji se koristi za opovrgnuti
- Koristite nejednakosti i prolaznost
- Različiti načini:
  - Find a constant c and then n_0
  - Fix n_0, say $n_0 = 1$, and then find c
- Za zbirove(serije uopšte):
  - Upotreba indukcije

Direktan dokaz:

- $f(n) = 5n^4 + 3n^3 + 2n^2 + 4n + 1$

Note: $5n^4 \le 5n^4$
$3n^3 \le 3n^4$
$2n^2 \le 2n^4$
$4n \le 4n^4$
$1 \le 1n^4$
$\forall n \geq 1$

Tako da:

$$f(n) = 5n^4 + 3n^3 + 2n^2 + 4n + 1$$

$$\leq 5n^4 + 3n^4 + 2n^4 + 4n^4 + 1n^4$$
$$\leq (5 + 3 + 2 + 4 + 1)n^4$$

c=15

$f(n) \le 15n^4 \Rightarrow$ f(n) is $O(n^4)$

Primeri

- $f(n) = a_d n^d + a_{d-1} n^{d-1} + \ldots + a_2 n^2 + a_1 n + a_0$

  $\leq (a_d + |a_{d-1}| + \ldots + |a_2| + |a_1| + |a_0|)n^d$

  $\leq cn^d,\ n_0 = 1$

  $\Rightarrow$ $f(n)$ je $O(n^d)$, $a_d > 0$

- $f(n) = 3n^2 - 2n + 4$

  $\leq 3n^2 + 4$, za sve $n \geq 1$

  $\leq (3 + 4)n^2$

  $\Rightarrow$ $f(n)$ je $O(n^2)$, izabrati $c = 7$

Primeri

- $f(n) = 2n^3 + 10\ n \log n + 5$

  $\leq (2 + 10 + 5)n^3$

  ⇨ $f(n)$ je $O(n^3)$, $n_0 = 1$ i $c = 17$

- $f(n) = 3n \log n + 2n$

  $\geq 3n \log n$

  ⇨ $f(n)$ je $\Omega(n \log n)$, $n_0 = 1$ i $c = 3$

Primer: Indukcija

- Dokaz indukcijom -- dokaži da: $f(n) = \sum_{i=1}^{n} i = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$

Prvi korak:

n = 1

$$\sum_{i=1}^{1} i = \frac{1(1+1)}{2} = 1$$

Inductive hipoteza:

pretpostavimo istinu za n = k

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

Note: u *jakoj* indukciji, induktivna hipoteza pretpostavlja da je P(k) istina za sve n ≤ k

Primer: Indukcija

Induktivni korak:

n = k + 1

$$\sum_{i=1}^{k+1} i = \frac{(k+1)[(k+1)+1]}{2}$$

$$\sum_{i=1}^{k+1} i = \left[\sum_{i=1}^{k} i\right] + (k+1)$$ (Def. of ∑)

$$= \frac{k(k+1)}{2} + (k+1)$$ (Induktivna hipoteza)

$$= \frac{k(k+1)}{2} + \frac{2(k+1)}{2}$$ (množ. sa 2/2)

$$= \frac{(k+1)(k+2)}{2}$$ (izolacija (k+1))

$$= \frac{(k+1)[(k+1)+1]}{2}$$ (k+2=(k+1)+1 asocijativnost +)

Reference

1. Algorithm Design and Applications by M. Goodrich and R. Tamassia, Wiley, 2015.
2. Data Structures and Algorithms in Java, 6th Edition, by M. Goodrich and R. Tamassia, Wiley, 2014.
3. Data Structures and Algorithm Analysis in Java, 3rd Edition, by M. Weiss, Addison-Wesley, 2012.
4. Algorithm Design by J. Kleinberg and E. Tardos, Addison-Wesley, 2006.
5. Introduction to Algorithms, 2nd Edition, by T. Cormen et al., McGraw-Hill, 2001.

Pitanja

1. Koje su glavne prednosti teorijske analize?
2. Sortirajte sledeće funkcije po povećanju redosleda rasta: $2n^2$, $6 \log n$, $0.6 n^3$, $2n \log n$, 34
3. Prebrojavanje operacija za algoritam linearSearch i naćiT(n).
4. Opišite glavni pristup teorijske analize.
5. Diskutujte o razlikama između različitih tipova asimpomatske notacije.
6. Zašto je binarySearch's vreme izvršavanja $O(\log n)$?
7. Pokaži da $T(n) = 3n^2 – 4n + 32$ je $O(n^2)$, $\Omega(n^2)$ $\Theta(n^2)$? Da li je $O(n^3)$? Ili $\Omega(n \log n)$? A $\Omega(1)$?
8. Uraditi isto za $0.2 \log n – 8 n \log n + 4 n^3$
9. Koje su glavne metode za dokaze/opravdanja za asimptotičko zapazanje? Dati primer svakog metoda.
10. Razmotrite problem. Napišite dva algoritma za koja se razlikuju najbolja i najgora vremena izvršavanja.