

Analiza algoritama


© Goodrich, Tamassia, Goldwasser

Katedra za informatiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

2021.

Analiza algoritama

- algoritam će od nekog ulaza proizvesti neki izlaz
- ULAZ → ALGORITAM → IZLAZ

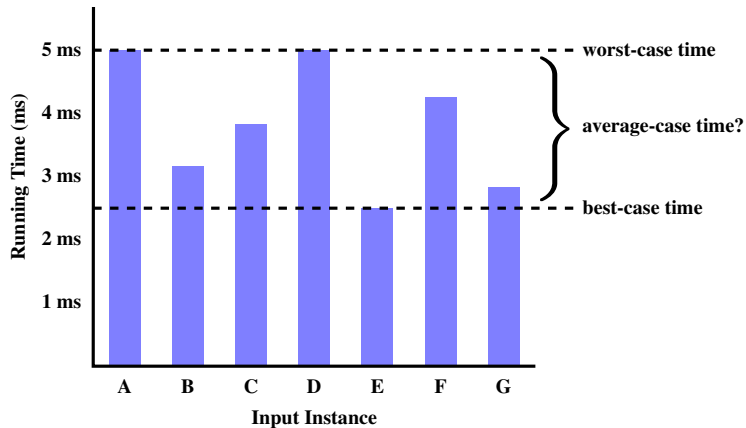


Da li je algoritam
funkcija?
Kako možemo uporediti ova
dva termina?

Vreme izvršavanja

- većina algoritama transformiše objekte na ulazu u objekte na izlazu
- **vreme izvršavanja** algoritma obično raste sa veličinom ulaza
- teško je izračunati prosečno vreme izvršavanja
- posmatraćemo najgori slučaj
 - jednostavnije za analizu
 - ključno za primene kao što su igre, finansije ili robotika

Vreme izvršavanja



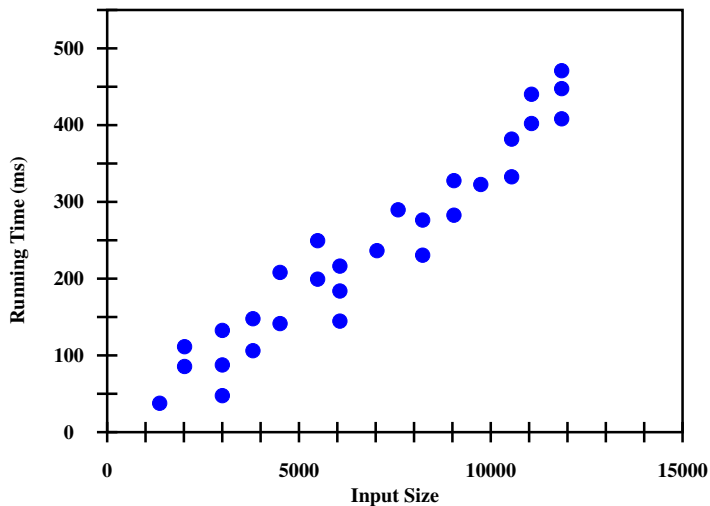
Eksperimentalno proučavanje algoritama

- napisati program koji implementira posmatrani algoritam
- pokrenuti program za različite veličine i strukturu ulaznih podataka
- meriti vreme izvršavanja
- analizirati rezultate

Merenje vremena pomoću sistemskog sata

```
from time import time
start_time() = time()
# ... run algorithm ...
end_time = time()
elapsed = end_time - start_time
```

Analiza rezultata merenja



Ograničenja eksperimentalnog pristupa

- potrebno je implementirati algoritam — može biti teško
- teško je predvideti rezultate za ulaze koji nisu obuhvaćeni eksperimentom
- za poređenje dva algoritma mora se koristiti identično hardversko i softversko okruženje

Teorijski pristup

- koristi se opis algoritma visokog nivoa umesto implementacije
- opisuje vreme izvršavanja kao funkciju veličine ulaza — n
- uzima u obzir sve moguće ulaze
- omogućava procenu brzine algoritma nezavisno od korišćenog hardvera ili softvera

Pseudokôd

- opis algoritma visokog nivoa
- bolje strukturiran od prirodnog jezika
- manje detalja nego u stvarnom programu
- sakriva detalje vezane za dizajn programa
- poželjna notacija za opisivanje algoritama

Pseudokôd

- primer: pronalaženje najvećeg broja u nizu

arrayMax(A, n)

Input: A : niz celih brojeva

Input: n : dužina niza

$currentMax \leftarrow A[0]$

for $i \leftarrow 1$ **to** $n - 1$ **do**

if $A[i] > currentMax$ **then**

$currentMax \leftarrow A[i]$

return $currentMax$

Pseudokôd

- kontrola toka
 - **if ...then ...[else] end if**
 - **for ...do ...end for**
 - **while ...do ...end while**
 - **repeat ...until ...**
- izrazi
 - \leftarrow dodela vrednosti
 - $=$ poređenje vrednosti
 - n^2 matematička notacija je OK
- vraćanje rezultata
 - **return** vrednost

Sedam važnih funkcija ₁

- konstantna funkcija

$$f(n) = c$$

- osnovna funkcija $g(n) = 1$
- svaka druga može se prikazati kao $f(n) = c \cdot g(n)$
- može da opiše broj koraka potrebnih za neku od osnovnih operacija
- npr. sabiranje, dodela vrednosti, poređenje

Sedam važnih funkcija ₂

- logaritamska funkcija

$$f(n) = \log_b n$$

- za $b > 1$
- za osnovu logaritma se najčešće koristi 2
- 2 se podrazumeva, tj. $\log n = \log_2 n$
- podela problema na dva dela: čest princip koji se koristi u algoritmima

Sedam važnih funkcija ₃

- linearna funkcija

$$f(n) = n$$

- kada treba obaviti prostu operaciju nad svakim od n elemenata ulaza
- npr. poređenje broja sa svim elementima niza

Sedam važnih funkcija ₄

- n-log-n funkcija

$$f(n) = n \log n$$

- raste nešto brže od linearne funkcije
- i znatno sporije od kvadratne funkcije
- npr. najbrže sortiranje n brojeva zahteva $n \log n$ vreme

Sedam važnih funkcija 5

- kvadratna funkcija

$$f(n) = n^2$$

- npr. dve ugnježdene petlje
- gde unutrašnja obavlja linearan broj operacija nad elementima ulaza
- a spoljna se izvršava linearan broj puta
- su proporcionalne sa n^2

Sedam važnih funkcija ₆

- kubna funkcija

$$f(n) = n^3$$

- ređe se javlja od kvadratne, ali
- predstavlja jednu klasu **polinomijalnih** funkcija

$$f(n) = a_0 + a_1n + a_2n^2 + a_3n^3 + \dots + a_dn^d$$

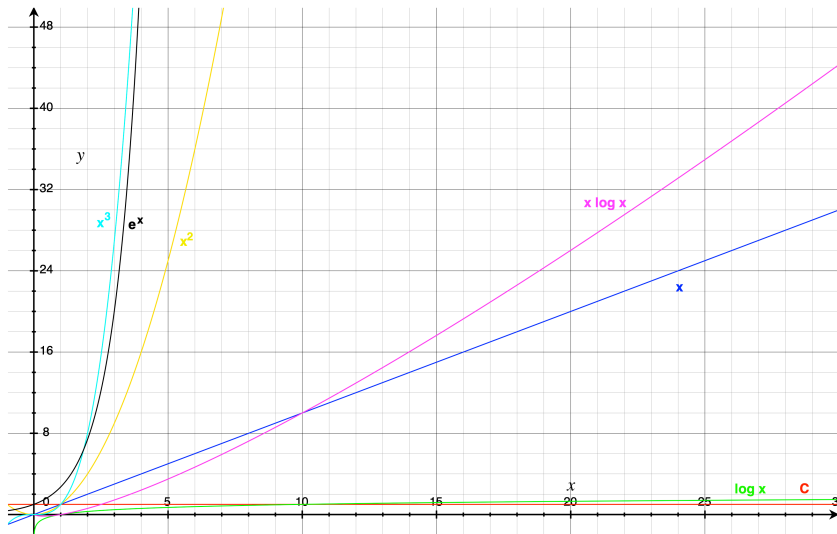
Sedam važnih funkcija 7

- eksponencijalna funkcija

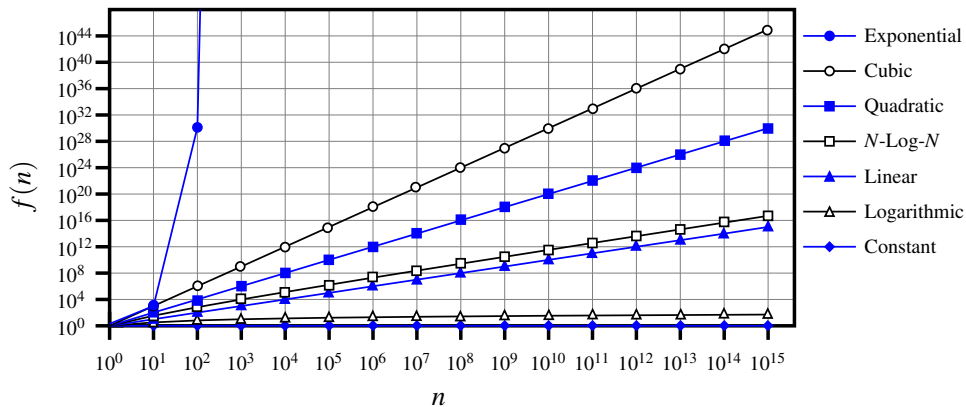
$$f(n) = b^n$$

- b je baza
- n je eksponent
- često je $b = 2$
- najsporija

Sedam važnih funkcija



Sedam važnih funkcija



Primitivne operacije

- osnovne operacije koje izvršava algoritam
- prikazane u pseudokodu
- nezavisne od programskog jezika
- troše konstantnu količinu vremena
- na primer:
 - izračunavanje izraza
 - dodela vrednosti promenljivoj
 - pristup elementu niza preko indeksa
 - poziv funkcije
 - vraćanje rezultata

Brojanje primitivnih operacija

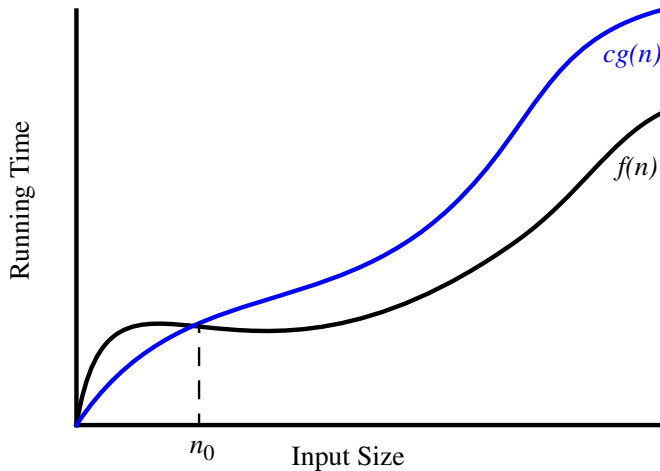
- analizom pseudokoda možemo odrediti maksimalan broj primitivnih operacija koje izvršava algoritam kao funkciju veličine ulaza

| Algoritam $arrayMax(A, n)$ | br. operacija |
|---|----------------------|
| $currentMax \leftarrow A[0]$ | 2 |
| for $i \leftarrow 1$ to $n - 1$ do | $2n$ |
| if $A[i] > currentMax$ then | $2(n - 1)$ |
| $currentMax \leftarrow A[i]$ | $2(n - 1)$ |
| increment i | $2(n - 1)$ |
| return $currentMax$ | 1 |
| ukupno | $8n - 2$ |

Procena vremena izvršavanja

- algoritam **arrayMax** izvršava $8n - 2$ primitivnih operacija u najgorem slučaju
- neka je
 - a : vreme izvršavanja **najbrže** primitivne operacije
 - b : vreme izvršavanja **najsporije** primitivne operacije
 - $T(n)$ vreme u najgorem slučaju
- tada je
 - $a(8n - 2) \leq T(n) \leq b(8n - 2)$
 - $\Rightarrow T(n)$ je ograničena sa dve linearne funkcije!

Primer ograničene funkcije



Porast vremena izvršavanja

- izmena u hardverskom ili softverskom okruženju
 - menja $T(n)$ za konstantan faktor
 - ali ne menja brzinu rasta $T(n)$
- linearni porast vremena izvršavanja $T(n)$ je suštinska osobina algoritma **arrayMax**

Zašto je porast vremena bitan

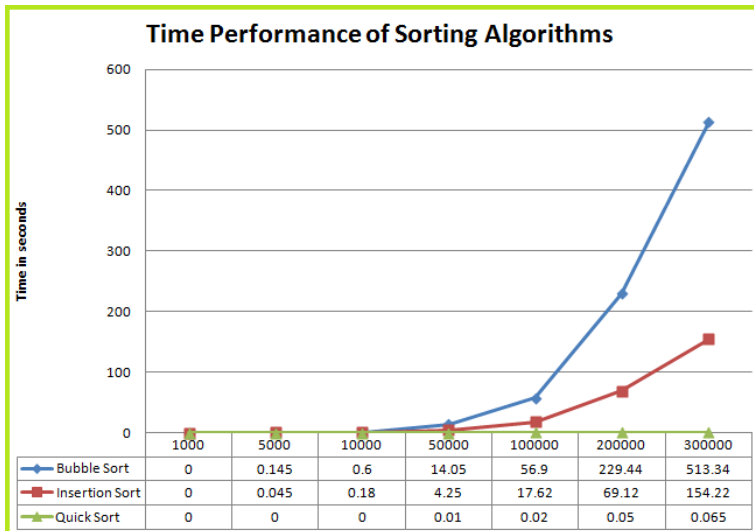
| $T(n)$ | vreme za $n+1$ | vreme za $2n$ | vreme za $4n$ |
|-------------|-----------------------|--------------------|--------------------|
| $c \log n$ | $c \log(n+1)$ | $c(\log n + 1)$ | $c(\log n + 2)$ |
| cn | $c(n+1)$ | $2cn$ | $4cn$ |
| $cn \log n$ | $\sim cn \log n + cn$ | $2cn \log n + 2cn$ | $4cn \log n + 4cn$ |
| cn^2 | $\sim cn^2 + 2cn$ | $4cn^2$ | $16cn^2$ |
| cn^3 | $\sim cn^3 + 3cn$ | $8cn^3$ | $64cn^3$ |
| $c2^n$ | $c2^{n+1}$ | $c2^{2n}$ | $c2^{4n}$ |

* za dvostruki ulaz četverostruko vreme

Primer poređenja dva algoritma

- insertion sort je $n^2/4$
- merge sort je $2n \log n$
- sortiramo milion elemenata
 - insertion sort ~ 70 sati
 - merge sort ~ 40 sekundi
- na 100x bržoj mašini to bi bilo
 - insertion sort ~ 40 minuta
 - merge sort ~ 0.5 sekundi

Primer poređenja tri algoritma



Konstantni činioci

- porast vremena izvršavanja ne zavisi od
 - konstantnih činilaca
 - izraza nižeg reda
- na primer
 - $10^2n + 10^5$ je linearna funkcija
 - $10^5n^2 + 10^8n$ je kvadratna funkcija

Veliko O notacija

- „Big-Oh“
- opisuje granično ponašanje funkcije kada argument raste
- za date $f(n)$ i $g(n)$ kažemo da $f(n)$ je $O(g(n))$ ako postoje pozitivne konstante c i n_0 takve da

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

Veliko O notacija

- primer: $2n + 10$ je $O(n)$
 - $2n + 10 \leq cn$
 - $(c - 2)n \geq 10$
 - $n \geq 10/(c - 2)$
 - izaberemo $c = 3$ i $n_0 = 10$

Veliko O notacija

- primer: n^2 nije $O(n)$
 - $n^2 \leq cn$
 - $n \leq c$
 - ova nejednakost ne može biti zadovoljena jer je c konstanta

Još primera

- $7n - 2$ je $O(n)$
 - tražimo $c > 0$ i $n_0 \geq 1$ takve da $7n - 2 \leq cn$ za $n \geq n_0$
 - ovo je zadovoljeno za $c = 7$ i $n_0 = 1$
- $3n^3 + 20n^2 + 5$ je $O(n^3)$
 - tražimo $c > 0$ i $n_0 \geq 1$ takve da $3n^3 + 20n^2 + 5 \leq cn^3$ za $n \geq n_0$
 - ovo je zadovoljeno za $c = 4$ i $n_0 = 21$
- $3 \log n + 5$ je $O(\log n)$
 - tražimo $c > 0$ i $n_0 \geq 1$ takve da $3 \log n + 5 \leq c \log n$ za $n \geq n_0$
 - ovo je zadovoljeno za $c = 8$ i $n_0 = 2$

Veliko O i porast vremena

- veliko O definiše gornju granicu na rast funkcije
- tvrdnja $f(n)$ je $O(g(n))$ znači da $f(n)$ ne raste brže od $g(n)$
- možemo da koristimo veliko O da rangiramo funkcije po brzini rasta

| | $f(n)$ je $O(g(n))$ | $g(n)$ je $O(f(n))$ |
|--------------------|---------------------|---------------------|
| $g(n)$ raste brže | da | ne |
| $f(n)$ raste brže | ne | da |
| jednako brzo rastu | da | da |

Veliko O: još neka pravila

- ako je $f(n)$ polinom stepena d tada $f(n)$ je $O(n^d)$, tj.
 - možemo zanemariti niže stepene polinoma
 - možemo zanemariti konstantne koeficijente
- koristimo najsporiju moguću klasu funkcija
 - kažemo „ $2n$ je $O(n)$ “ umesto „ $2n$ je $O(n^2)$ “
- koristimo najjednostavniji izraz koji predstavlja klasu
 - kažemo „ $3n + 5$ je $O(n)$ “ umesto „ $3n + 5$ je $O(3n)$ “

Asimptotska analiza algoritama

- asimptotska analiza algoritama određuje vreme izvršavanja u „veliko O“ notaciji
- kako obaviti asimptotsku analizu
 - odredimo broj primitivnih operacija u najgorem slučaju kao funkciju veličine ulaza
 - izrazimo funkciju u „veliko O“ notaciji
- primer:
 - ustanovimo da **arrayMax** izvršava najviše $8n - 2$ primitivnih operacija
 - kažemo da je složenost **arrayMax** algoritma $O(n)$
- konstantne činioce i izraze nižeg stepena svakako ne iskazujemo na kraju, pa ih možemo zanemariti kada brojimo primitivne operacije

Računanje proseka prefiksa

- i -ti prosek prefiksa niza X je prosek vrednosti prvih $i + 1$ elemenata X

$$A[i] = (X[0] + X[1] + \dots + X[i]) / (i + 1)$$

- niz A se koristi u finansijskoj analizi

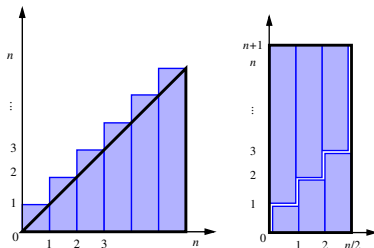
Računanje proseka prefiksa

- računanje proseka prefiksa prema definiciji: $O(n^2)$

| Algoritam <i>prefixAverages1</i> (X, n) | br. operacija |
|---|---------------------------|
| $A \leftarrow$ novi niz od n integera | n |
| for $i \leftarrow 0$ to $n - 1$ do | n |
| $s \leftarrow X[0]$ | n |
| for $j \leftarrow 1$ to i do | $1 + 2 + \dots + (n - 1)$ |
| $s \leftarrow s + X[j]$ | $1 + 2 + \dots + (n - 1)$ |
| $A[i] \leftarrow s / (i + 1)$ | n |
| return A | 1 |

Računanje proseka prefiksa

- složenost **prefixAverages1** je $O(1 + 2 + \dots + n)$
- suma prvih n prirodnih brojeva je $n(n + 1)/2$
- \Rightarrow algoritam radi za $O(n^2)$ vreme



Računanje proseka prefiksa ₂

- računanje proseka prefiksa u linearnom vremenu pomoću tekuće sume

| Algoritam <i>prefixAverages2</i> (X, n) | br. operacija |
|---|----------------------|
| $A \leftarrow$ novi niz od n integera | n |
| $s \leftarrow 0$ | 1 |
| for $i \leftarrow 0$ to $n - 1$ do | n |
| $s \leftarrow s + X[i]$ | n |
| $A[i] \leftarrow s / (i + 1)$ | n |
| return A | 1 |

Šta nam treba od matematike

- tehnike izvođenja dokaza
- verovatnoća
- redovi
- logaritmi
 - $\log_b(xy) = \log_b x + \log_b y$
 - $\log_b(x/y) = \log_b x - \log_b y$
 - $\log_b(xa) = a \log_b x$
 - $\log_b a = \log_x a / \log_x b$
- eksponencijalne funkcije
 - $a^{b+c} = a^b a^c$
 - $a^{bc} = (a^b)^c$
 - $a^b / a^c = a^{b-c}$
 - $b = a \log_a b$
 - $b^c = a^{c \log_a b}$

Rodaci velikog O

- Ω : veliko Omega
 - $f(n)$ je $\Omega(g(n))$ ako postoje $c > 0$ i $n_0 \geq 1$ takvi da $f(n) \geq cg(n)$ za $n \geq n_0$
- Θ : veliko Teta
 - $f(n)$ je $\Theta(g(n))$ ako postoje $c_1 > 0$ $c_2 > 0$ i $n_0 \geq 1$ takvi da $c_1g(n) \leq f(n) \leq c_2g(n)$ za $n \geq n_0$

Rođaci velikog O

- veliko O
 - $f(n)$ je $O(g(n))$ ako je $f(n)$ asimptotski **manje ili jednako** $g(n)$
- veliko Ω
 - $f(n)$ je $\Omega(g(n))$ ako je $f(n)$ asimptotski **veće ili jednako** $g(n)$
- veliko Θ
 - $f(n)$ je $\Theta(g(n))$ ako je $f(n)$ asimptotski **jednako** $g(n)$

Primeri sa O , Ω , Θ

- $5n^2$ je $\Omega(n^2)$
 - $f(n)$ je $\Omega(g(n))$ ako postoje $c > 0$ i $n_0 \geq 1$ takvi da $f(n) \geq cg(n)$ za $n \geq n_0$
 - rešenje: $c = 5$ i $n_0 = 1$
- $5n^2$ je $\Omega(n)$
 - $f(n)$ je $\Omega(g(n))$ ako postoje $c > 0$ i $n_0 \geq 1$ takvi da $f(n) \geq cg(n)$ za $n \geq n_0$
 - rešenje: $c = 1$ i $n_0 = 1$
- $5n^2$ je $\Theta(n^2)$
 - $f(n)$ je $\Theta(g(n))$ ako je $\Omega(n^2)$ i $O(n^2)$; prvi uslov smo proverili a drugi je ispunjen za
 - $c = 5$ i $n_0 = 1$