Formalne metode

u softverskom inženjerstvu

14 Složenost

ETFBL 24-25

Dunja Vrbaški

Halting problem

Da li postoji algoritam koji za proizvoljan program P i ulaz x može odgovoriti da li će izvršavanje zaustaviti ili ne?

Deluje da može - nekako analiziramo kod za dati ulaz .
Univerzalni algoritam ne postoji.
Halting problem je neodlučiv.

```
H(P,x)
  ako P(x) se zaustavlja
    return T
  ako P(x) se ne zaustavlja
    return F
```

```
D(Q)
  ako H(Q, Q) == T  // Q(Q) se zaustavlja
    loop
  ako H(Q, Q) == F
    halt
```

D(D)=?

Iz 1. Ako D(D) staje → H(D, D) = T
Iz 2. D(D) → loop

Iz 1. Ako D(D) ne staje → H(D, D) = F
Iz 2. D(D) → halt

Acceptance problem

Da li postoji algoritam koji za proizvoljan program P i ulaz x može odgovoriti da li će izvršavanje prihvatiti x?

Problem je neodlučiv.

```
H(P,x)
  ako P(x) prihvata
    return T
  ako P(x) ne prihvata
    return F
```

```
D(Q)
  ako H(Q, Q) == T
    odbija // loop ili halt bez prihvatanja
  ako H(Q, Q) == F
    prihvata
```

D(D)=?

Iz 1. Ako D(D) prihvata→ H(D, D) = T
Iz 2. D(D) → odbija

Iz 1. Ako D(D) ne prihvata → H(D, D) = F
Iz 2. D(D) → prihvata

	A_TM – Prihvata?	**HALT_TM – Staje?**
Pitanje	Da li se program zaustavlja i prihvata ulaz?	Da li se program uopšte zaustavlja (na bilo koji način)?
Odgovor DA	Ako se zaustavi i prihvati	Ako se zaustavi (bilo kako)
Odgovor NE	Ako odbije ili je u petlji	Ako je u petlji zauvek
Zašto NE ne možeš znati?	Jer ne znaš da li će *kasnije* možda prihvatiti	Jer ne znaš da li će *kasnije* možda stati

Acceptance problem — pripadanje jeziku tipa 0

Da li data reč w pripada jeziku L(G) koji generiše neka gramatika G tipa 0?
Jezike tipa0 prihvataju Tjuringove mašine.

Problem pripadanja jeziku - neodlučiv.

Rajsova teorema

Svaka netrivijalna semantička osobina programa je neodlučiva.

semantička → odnosi se na *ponašanje* programa, na jezik koji prihvata, a ne na sam kod.

sintaksička: program ima dva if iskaza; ima 10 linija koda; koristi promenljivu x
semantička: ispisuje 42; ima beskonačnu petlju.

netrivijalna → osobina nije tačna za sve Turing-mašine, niti netačna za sve Turing-mašine.

trivijalna: program je tjuringova mašina; program prihvata neki RE jezik;
netrivijalna: program prihvata samo 42; program prihvata sve reči koje počinju na slovo ‘a’;

Da li program ikad ispisuje 42?

semantička - ponašanje, ne oblik koda
netrivijalna - neki programi ispisuju, neki ne

Rajsova teoema - Ne postoji algoritam koji za proizvoljan program odlučuje da li će program ispisati 42 ili ne.

→ Ako znaš da je semantička i netrivijalna - znaš da je neodlučivo

Ne možemo napraviti alat koji će nam proveravati da li će se program izvršiti bez greške.
Ne postoji univerzalni algoritam koji će nam automatski verifikovati da ne postoje bagovi.

```
prints_42(code_str)
    output = …
    exec(code_str,  output)
    return ‘42’ in output


code1 = ‘print(42)’
code2 = ‘print(41)’
code3 = ‘while true’

prints_42(code1)
prints_42(code2)
prints_42(code3)
```

Za neku funkciju, ko kaže da ako ne ispiše odmah neće možda ispisati 42 za koji dan?

→ Ako znaš da je semantička i netrivijalna - znaš da je neodlučivo

Ne možemo napraviti alat koji će nam proveravati da li će se program izvršiti bez greške.
Ne postoji univerzalni algoritam koji će nam automatski verifikovati da ne postoje bagovi.

Ako je problem odlučiv ne znači i da je “lak” za izračunavanje.

Uglavnom nas zanimaju praktično odlučivi problemi.

Mašina za rešavanje je posmatrana kao apstrakcija realnog sistema.

Klasifikacija složenosti rešenja se može posmatrati u odnosu na dostupne resurse:
vreme i memorija (vremenska i prostorna složenost).

Obično dužinu ili prostornu iskorišćenost rada limitiramo nekom funkcijom dužine ulaznih podataka.

Ostale odlučive probleme pokušavamo da aproksimiramo ili ogradimo.

NIje precizna granica, zavisi od nas, ali moramo poznavati algoritme da bismo mi mogli da odlučimo šta je nama smisleno.

Diskusije:

- Dovoljno je: raste brzina računara i dostupna memorija
- Nije važan algoritam ML/AI - važno je imati dovoljno veliku količinu podataka
- Raste i količina podataka
- Prostorna složenost je “manje važna”

Izučavanje složenosti - apstrakcija: zanemarujemo hardver i kompajler

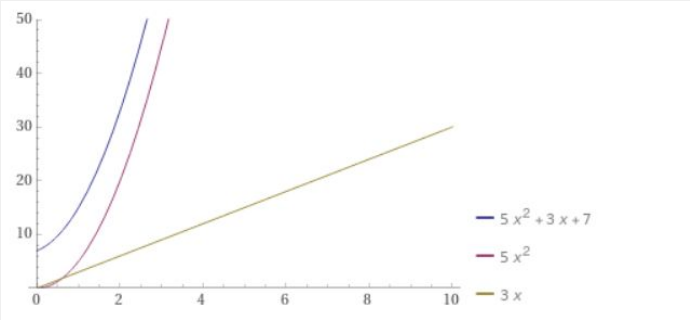
Kako je realizovan program?
Koliko koraka je potrebno za njegovo izvršavanje na osnovu veličine ulaza?

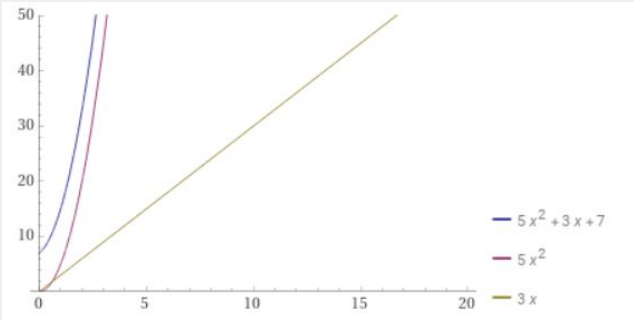
Što je ulaz veći verovatno treba više koraka (vremena).
Moguće da treba i više memorije.

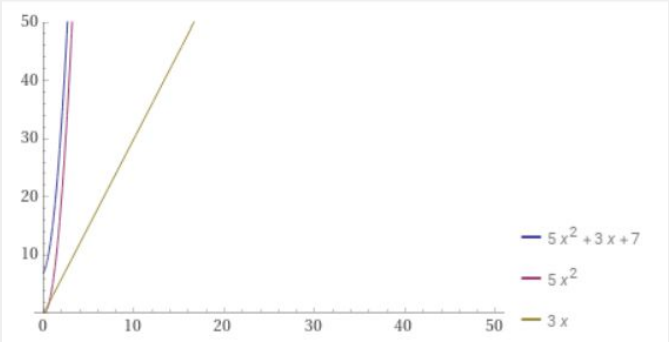
Nije svaki rast isti.
Razmatramo brzinu rasta.

Mera koja omogućava upoređivanje: Koji algoritam je “bolji”?

Uglavnom se analiziraju: najgori slučaj, najbolji slučaj, opšti slučaj ili prosečno.
Šta nas najviše zanima?


50
40
30
20
10
0
2
4
6
8
10
5 x^2 + 3 x + 7
5 x^2
3 x


50
40
30
20
10
0
5
10
15
20
5 x^2 + 3 x + 7
5 x^2
3 x


50
40
30
20
10
0
10
20
30
40
50
5 x^2 + 3 x + 7
5 x^2
3 x

O notacija (BigO)

Ako su f i g aritmetičke funkcije, onda je funkcija f u velikom O od g, u oznaci

$$f(x)=O(g(x))$$

ako postoje brojevi c i n takvi da za svaki x>n važi $f(x) \leq cg(x)$.

Može se reći da je f reda funkcije g ili da je **g asimptotska gornja granica f**.

Funkcije f i g rastu istom brzinom ako je $f(x)=O(g(x))$ i $g(x)=O(f(x))$.

Tada je:
$$f(x)=\Theta(g(x))$$

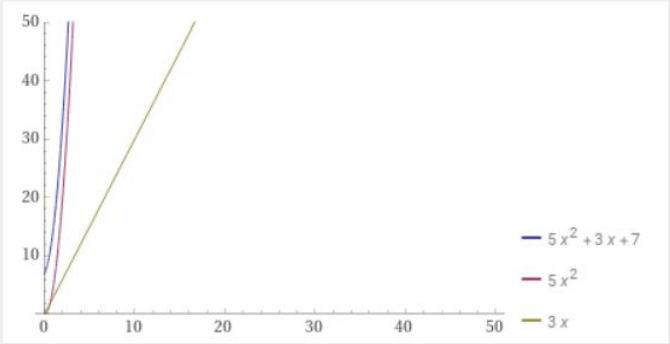
Ideja:

- Uz pomoć jednostavne funkcije opisujemo gornju granicu složenosti naše funkcije.
- Posmatramo asimptotsko ponašanje - šta se dešava kad veličina ulaza raste?

<u>postoje brojevi c i n</u> takvi da <u>za svaki x>n</u> važi ***f(x) ≤ cg(x)***

$3x^5 + 2x^4 - 7$

x^5



Ako je $P(n)=a_0+a_1n+...+a_rn^r$, $a_r \neq 0$ polinom stepena r sa celobrojnim koeficijentima onda za P(n) i n^m važi:

- ako je m=r, P(n) i n^m rastu istom brzinom
- ako je m<r, P(n) raste brže od n^m
- ako je m>r, n^m raste brže od P(n)

Funkcija k^n (k>1) raste brže od bilo kojeg polinoma sa celobrojnim koeficijentima.

Svaki polinom sa celobrojnim koeficijentima raste brže od bilo koje logaritamske funkcije.

Za bilo koje dvije realne konstante c>1 i d>1 važi $\log_c(x)=\Theta(\log_d(x))$, tj. baza logaritma ne utiče na brzinu rasta.

Kod računanja složenosti posmatra se dominantni član funkcije.

$e^X + x^{10} + x^3$

funkcije koje se najčešće javljaju u O-notaciji su:

- logaritamska funkcija $\log_2 n$ (ili češće samo logn),
- linearna funkcija kn,
- njihov proizvod nlogn,
- stepena funkcija n^k ,
- eksponencijalna funkcija k^n

Programiranje - algoritam - procena složenosti

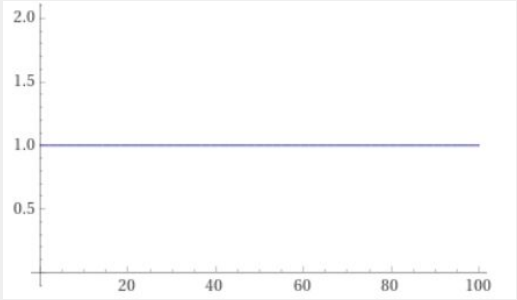
Posmatramo ulaz i tražimo funkciju u odnosu na veličinu ulaza.

- **O(1)**

```
boolean daLiJePrviElementNull(String[] niz) {
   return niz[0]==null;
}
```

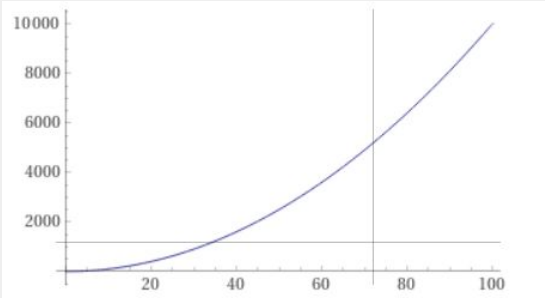
- **O(n)**

```
boolean sadrzi(String[] niz, String element) {
   for(int cnt=0;cnt<niz.length;cnt++) {
      if(niz[cnt].equals(element)) return true;
   }
   return false;
}
```

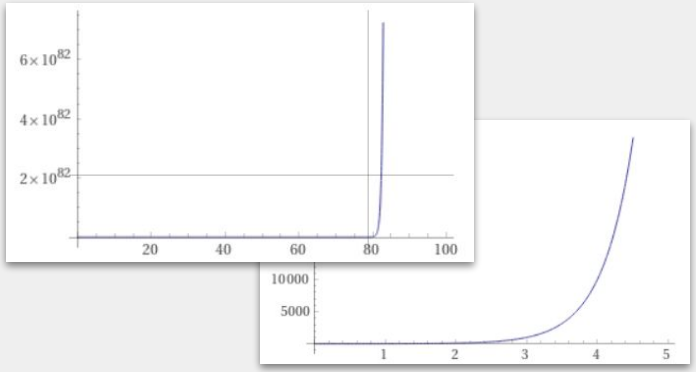


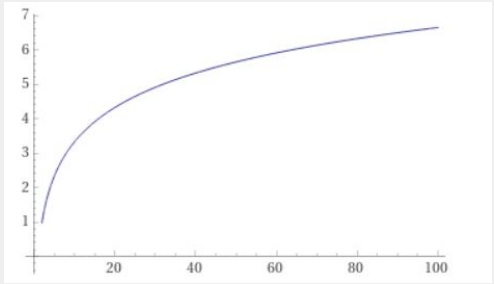
- **$O(n^2)$**

```
boolean sadrziDuplikate(String[] niz) {
   for(int i=0;i<niz.length;i++) {
      for(int j=0;j<niz.length;j++) {
         if(i==j) continue;
         if(niz[i].equals(niz[j])) return true;
      }
   }
   return true;
}
```



- **O(10^n)**
  - pogađanje lozinke dužine N sastavljene samo od cifara testiranjem svih mogućih vrijednosti,
- **O(logn)**
  - binarno pretraživanje sortiranog niza:
    - potrebno je pronaći element u sortiranom nizu,
    - algoritam pronalazi srednju vrijednost elemenata (tj. medijanu),
    - ako je to traženi broj, postupak se završava,
    - ako je traženi broj manji, onda se isti postupak ponavlja na "donjoj" polovini niza,
    - analogno važi ako je traženi broj veći,
    - algoritam se ne komplikuje mnogo sa porastom broja elemenata, jer se niz prepolovi u svakoj iteraciji





$O(1)$	bez petlji; provera
$O(\log n)$	binarno petraživanje
$O(n)$	prolazak kroz niz
$O(n \log n)$	merge sort; heapsort
$O(n^2)$	bubble sort; quicksort
$O(n^3)$	rešenje sistema sa tri nepoznate (matrice)
$O(2^n)$	podskupovi
$O(n!)$	permutacije

- Sudoku? Veći broj polja?
- Ostale mere za algoritme?
- Razmisliti za svaki - šta bi bio najgori i najbolji slučaj

Programski kod:

- sekvenca: $O(f_1(n)) + O(f_2(n))$
- grananje: Gleda se onaj koji je značajniji: $max\{O(f_1(n)), O(f_2(n))\}$
- petlja: $O(h(n)P(n))$

P_1 P_2	$O(P) = O(P_1) + O(P_2)$
if (uslov) then: P_1 else: P_2	$O(P) = \max \{O(P_1), O(P_2)\}$
for i = 1 to m: P_1	$O(P) = m * O(P_1)$
while uslov: P_1	$O(P) = m * O(P_1)$

Kako izračunati za neki algoritam?

- poznavati funkcije
- prepoznavati konstrukte i kojim funkcijama odgovara njihov rast
- prepoznavati koji članovi najbrže rastu i kako koji članovi utiču jedni na druge

Petlje i rekurzivni pozivi najviše utiču.
Traga se za takvim članovima.
Sabiranje i množenje konstantama ne utiče.

> Pokušava se smanjivanje složenosti petlji na svim nivoima

U čemu je razlika kad imamo dve for petlje:

- jedna ispod druge
- ugnježdene

Za ideju, ako nismo sigurni ili nas zanima: simulacija, dobro osmišljena, plot

napomena, terminologija

Šta je vreme?

- potrebno vreme za izvršavanje algoritma
- koje su jedinice?
- npr, prosečno vreme trajanja jedne instrukcije na računaru; ciklus na računaru?
- potrebno vreme → zavisi od jedinice
- upoređivanje algoritama → ako promenimo jedinicu: *const

Aritmetička složenost

- potreban broj osnovnih aritmetičkih operacija
- ne zavisi od arhitekture
- paralelizacija: mogućnost, analiza, smanjivanje

Tjuringova mašina - model za analizu izvršavanja

Ako je x ulazni podatak, onda se njegova dužina označava sa |x|

Vrijeme izvršavanja Tjuringove mašine M sa ulaznim podatkom x je jednako dužini niza konfiguracija koje predstavljaju to izračunavanje.

Drugim rečima, Tjuringova mašina radi u vremenu f(n) ako je za ulaz x dužina izvršavanja najviše f(|x|).
Funkcija f je u tom slučaju vremenska granica složenosti za M.

Prostorna granica složenosti se definiše nešto drugačije, jer se ne računa prostor u kojem su ulazni ili izlazni podaci (nego samo ono što se koristi u radu, međurezultati).

Prostor izvršavanja Tjuringove mašine M sa ulaznim podatkom x je jednak broju različitih ćelija traka (osim ulazne i izlazne) iznad kojih se u toku izvršavanja nađu glave.

Drugim rečima, Tjuringova mašina radi u prostoru f(n) ako je za ulaz x broj korištenih ćelija traka najviše f(|x|).

Funkcija f je u tom slučaju prostorna granica složenosti za M.

Zadatak:

Koja je vremenska i prostorna složenost za tjuringove mašine koje proveravaju palindrome

- mašina sa jednom trakom
- mašina sa dve trake

Još notacija...

$$f(x) = O(g(x)) \text{ ako } \exists c,\ n \text{ tako da } f(x) \leqslant cg(x),\ x > n$$

f nije većeg reda od g
f ne raste brže od g
f ograničeno odgore sa g

$$f(x) = o(g(x)) \text{ ako } \forall c,\ \exists n \text{ tako da } f(x) < cg(x)$$

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$$

f raste sporije
f je manjeg reda

$f(x) = \Omega(g(x))$ *ako nije* $f(x) = o(g(x))$
$\exists c, \exists n, f(x) \geqslant cg(x), x > n$
$\exists \epsilon$, *niz* x_n, $x_n \to \infty$ *tako da* $f(x) > \epsilon|g(x_n)|$

f raste bar jednako brzo
f nije manjeg reda

$f(x) = \omega(g(x))$ *ako nije* $O(g(x))$
$\forall c, \exists n, f(x) > cg(x), x > n$
$\exists \epsilon_n, x_n, \epsilon_n \to \infty, x_n \to \infty$ *tako da* $f(x_n) \geqslant \epsilon_n g(x), \forall n$

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = \infty$$

f je većeg reda od g
f raste brže od g

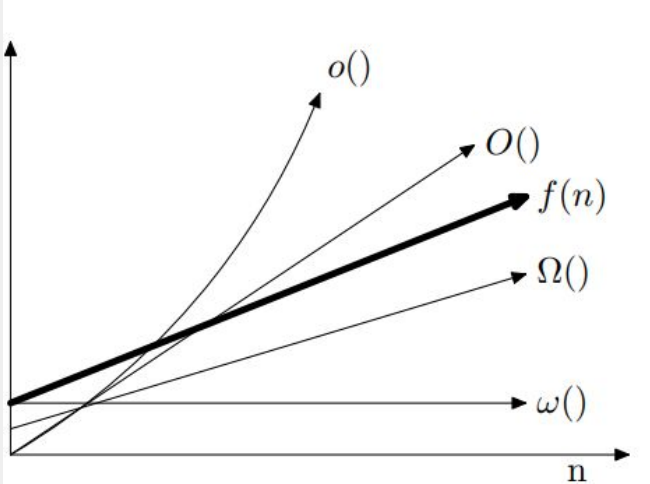
$f(x) = \Theta(g(x))$ *ako* $\exists c_1, c_2, n$ *tako da* $c_1 g(x) < f(x) < c_2 g(x)$

f je reda g*
f raste istom brzinom kao g

$f(x) = O(g(x))$ *i* $f(x) = \Omega(g(x))$

$f(x) \sim g(x)$ *ako* $\lim_{x \to \infty} \frac{f(x)}{g(x)} = 1$

f i g su asimptotski jednake



o: $\forall c,\ \exists n,\ f(x) < cg(x)$
O: $\exists c,\ \exists n,\ f(x) \leqslant cg(x)$
Ω: $\exists c,\ \exists n,\ f(x) \geqslant cg(x)$
w: $\forall c,\ \exists n,\ f(x) > cg(x)$

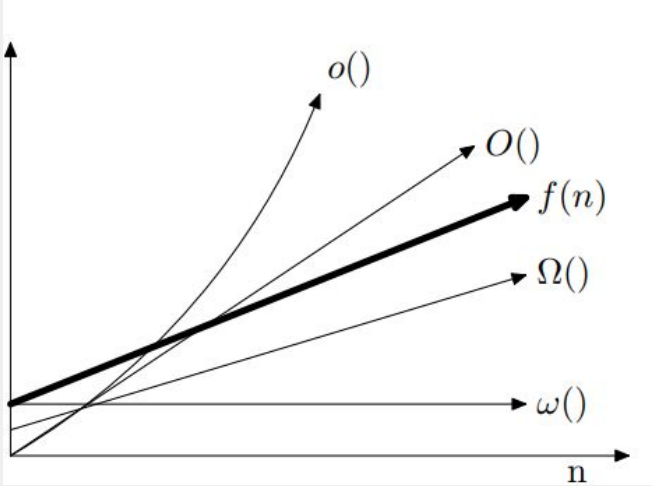
$$f(x) = o(g(x)) \text{ ako } \lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$$

$$f(x) = O(g(x)) \text{ ako } \exists c,\ n \text{ tako da } f(x) \leqslant cg(x),\ x > n$$

$$f(x) = \Theta(g(x)) \text{ ako } \exists c_1,\ c_2,\ n \text{ tako da } c_1 g(x) < f(x) < c_2 g(x)$$

$$f(x) = \Omega(g(x)) \text{ ako nije } f(x) = o(g(x))$$
$$\exists \epsilon,\ \text{niz } x_n,\ x_n \to \infty \text{ tako da } f(x) > \epsilon |g(x_n)|$$

$$f(x) = \omega(g(x)) \text{ ako nije } O(g(x))$$
$$\exists \epsilon_n,\ x_n,\ \epsilon_n \to \infty,\ x_n \to \infty \text{ tako da } f(x_n) \geqslant \epsilon_n g(x)$$



$$f(x) = o(g(x)) \text{ ako } \lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$$

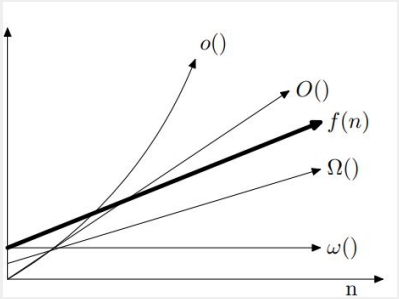
$$f(x) = O(g(x)) \text{ ako } \exists c,\ n \text{ tako da } f(x) \leqslant cg(x),\ x > n$$

$$\begin{aligned} o: &\quad \forall c,\ \exists n,\ f(x) < cg(x) \\ O: &\quad \exists c,\ \exists n,\ f(x) \leqslant cg(x) \\ \Omega: &\quad \exists c,\ \exists n,\ f(x) \geqslant cg(x) \\ w: &\quad \forall c,\ \exists n,\ f(x) > cg(x) \end{aligned}$$

o - za svaku konstantu će, u nekom momentu, g postati veće od f

O - dovoljno da postoji konstanta za koju će g biti bar f

$$\begin{aligned} f(x) &= 3x + 5 \\ f(x) &= O(x),\ f(x) \neq o(x) \\ f(x) &= O\left(x^2\right),\ f(x) = o\left(x^2\right) \end{aligned}$$



$$f(x) = O(x^a),\ \forall a > 0$$

blagi rast

$$\exists a_1,\ a_2 > 0\ tako\ da\ f(x) = \Omega(x^{a1})\ i\ f(x) = O(x^{a2})$$

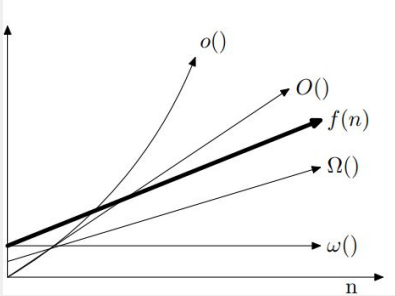
polinomni rast / polinomijalni

$$\exists a_1,\ a_2 > 1\ tako\ da\ f(x) = \Omega(a_1{}^x)\ i\ f(x) = O(a_2{}^x)$$

eksponencijalni rast

$$f(x) = \Omega(a^x),\ \forall a > 0$$

nadeksponencijalni rast



napomena, terminologija

$$\exists a_1,\ a_2 > 0 \text{ tako da } f(x) = \Omega\left(x^{a1}\right) \text{ i } f(x) = O\left(x^{a2}\right)$$

polinomni rast / polinomijalni

Ako ima polinomni rast ne znači da moraju biti istog reda.
Može važiti:

$$f(x) = O\left(x^n\right) \text{ i } f(x) \neq \Theta\left(x^n\right)$$

$f(x) = O(x^a), \forall a > 0$

blagi rast

$f(x) = log(x)$

$\exists a_1, a_2 > 0$ *tako da* $f(x) = \Omega(x^{a1})$ *i* $f(x) = O(x^{a2})$

polinomni rast

$f(x) = x\,log(x)$

$\exists a_1, a_2 > 1$ *tako da* $f(x) = \Omega(a_1{}^x)$ *i* $f(x) = O(a_2{}^x)$

eksponencijalni rast

$f(x) = x\,2^x$

$f(x) = \Omega(a^x), \forall a > 0$

nadeksponencijalni rast

$f(x) = x^x$

- Laki i teški problemi
- Jako laki, laki, teški i jako teški problemi

- Klase složenosti
- Da li ima poklapanja
- Da li ima svođenja

- Heuristike