RI301

Strukture podataka

dr.sc. Edin Pjanić

Pregled predavanja

- Operacije nad strukturama podataka
- Pojam složenosti (kompleksnosti) algoritma

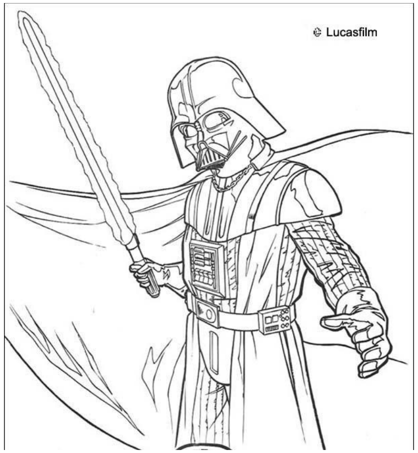
Operacije nad strukturama podataka

- Pri implementaciji struktura podataka često se susrećemo sa operacijama kao što su:
    - Kopiranje većeg broja elemenata
    - Dodavanje jednog ili više elemenata
    - Uklanjanje jednog ili više elemenata
    - Određivanje veličine strukture
    - Pristup određenom elementu strukture
- Performanse strukture podataka zavise od organizacije same strukture podataka i načina implementacije operacija nad strukturom podataka.

Složenost (kompleksnost) algoritma

- Kada osmislimo neki algoritam, potrebno je odlučiti na koji način ga implementirati.
- Koje vrijeme izvođenja očekujemo?
- Koliko memorije takav algoritam zahtijeva?
- Kako će se ponašati algoritam u slučaju povećanja broja ulaznih podataka, tj. koliki će biti porast zahtjeva za vremenom i/ili memorijom?
- Npr. razvili smo neki program (algoritam) za obradu podataka o nekim entitetima i testirali funkcionalnost i brzinu sa 20 entiteta: odziv je bio vrlo brz i u zadanim granicama.
    - Da li će (i kako) taj algoritam raditi sa 10,000 entiteta, a sa 100,000, a da li sa 5,000,000?

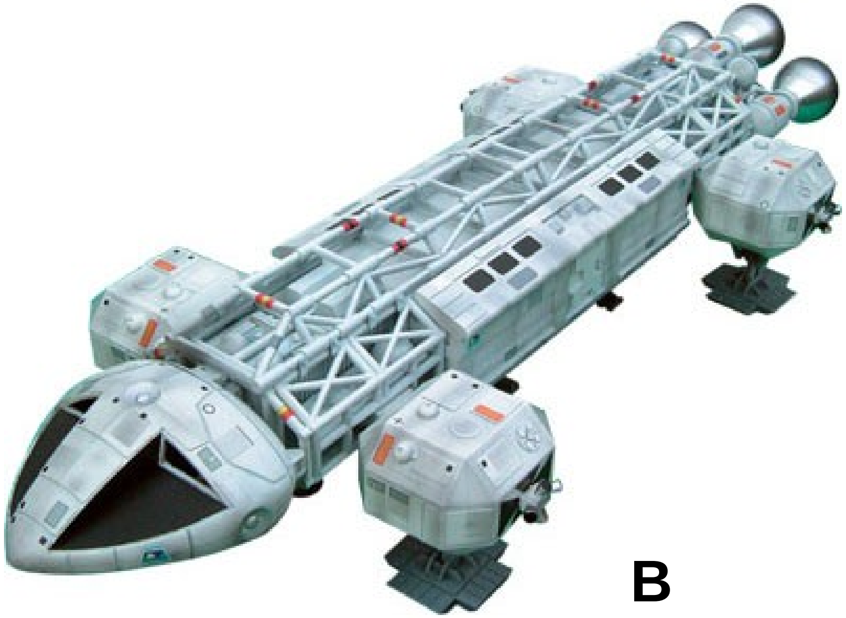
Primjer 1



- Firma *Darth Wader Transport* treba da raznese 100 kontejnera Hipermaterije na 100 lokacija (svemirskih stanica, planeta i asteroida) oko Zvijezde Smrti (Death Star). Na raspolaganju ima dvije letjelice.

A

Nosivost: 1 kontejner (jedva)
Potrošnja goriva: 2 t na 100 AJ

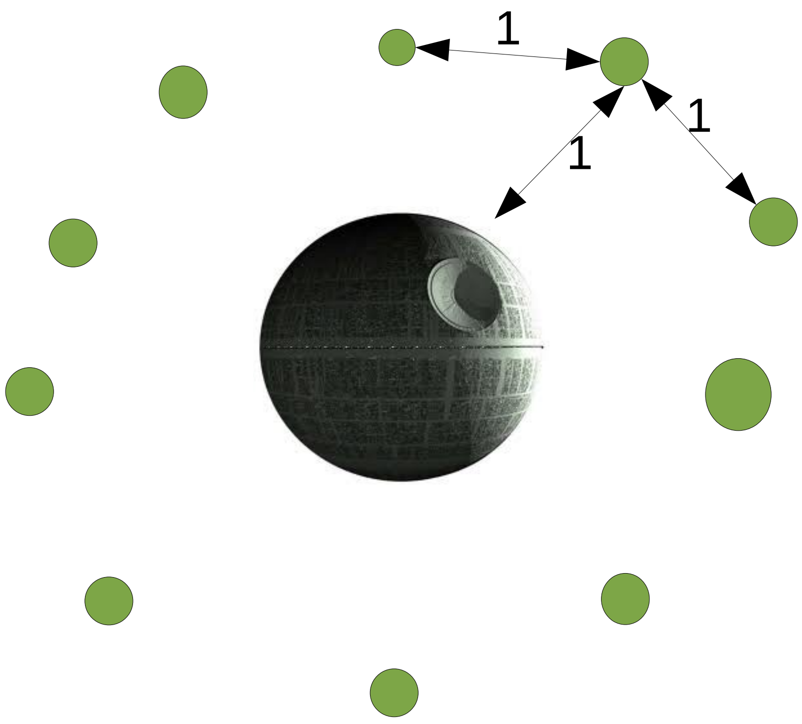


B

Nosivost: 125 kontejnera
Potrošnja goriva: 6 t na 100 AJ

Primjer 1a

- Lokacije se nalaze u svim pravcima oko Zvijezde Smrti na prosječnoj udaljenosti 1 AJ.
- Rastojanje između dvije susjedne lokacije je takođe u prosjeku 1 AJ, slično kao na slici (ali su u 3D prostoru).

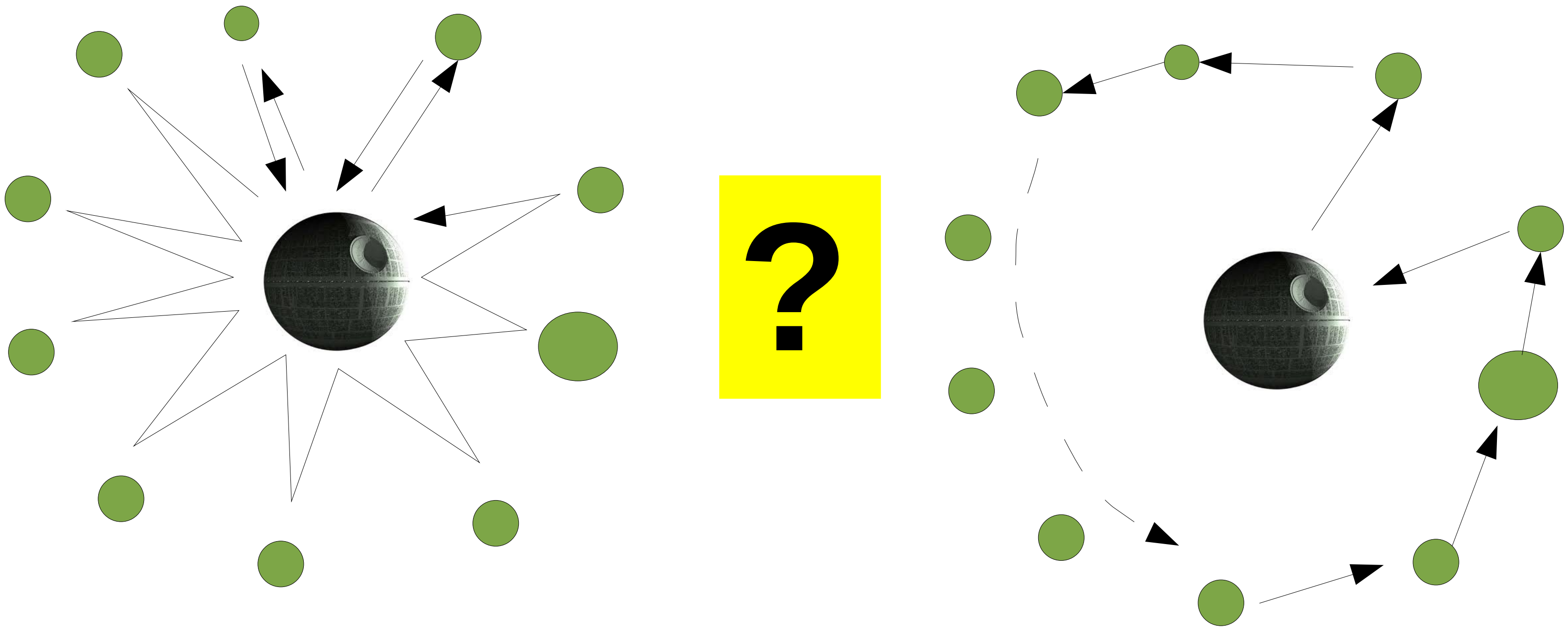


Koju letjelicu izabrati za isporuku navedenog tovara?

Šta se dešava kad imamo isporuku **n** kontejnera na **n** lokacija, gdje je **n** veliki broj?

Primjer 1a: analiza

- Letjelica A – nosivost 1
  - morala bi ići do svake lokacije i nazad



- Letjelica B – nosivost 125
  - može utovariti svih 100 kontejnera i isporučiti na sve lokacije bez vraćanja

Primjer 1a – proračun pređenog puta

- U slučaju letjelice A imamo:

$$f1 = \underbrace{(1+1)+(1+1)+ \ldots +(1+1)}_{\text{100-puta (n puta)}}=100 * 2 = 200$$

U slučaju **n** lokacija bismo imali: f1(n)=2n

- U slučaju letjelice B imamo:

$$f2 = \underbrace{1+1+ \ldots +1}_{\text{100-puta (n puta)}}+1=101$$

U slučaju **n** lokacija bismo imali: f2(n)=n+1

Primjer 1a - zaključak

- Potrošnja letjelice A:

200 AJ * 2 t/100AJ = 4 t goriva

- Potrošnja letjelice B:

101 AJ * 6 t/100AJ = 6 t goriva

f1 i f2 sa prethodnog slajda

Za **n** lokacija:

Potrošnja A: p1(n) = 2/100 * f1(n) = 0.04 * n

Potrošnja B: p2(n) = 6/100 * f2(n) = 0.06 * n + 0.06

Dakle, i jedna i druga imaju **linearnu ovisnost** o **n** ali im je nagib različit.

Njihov odnos ostaje približno isti kako n raste.

Složenost: Primjer 1b

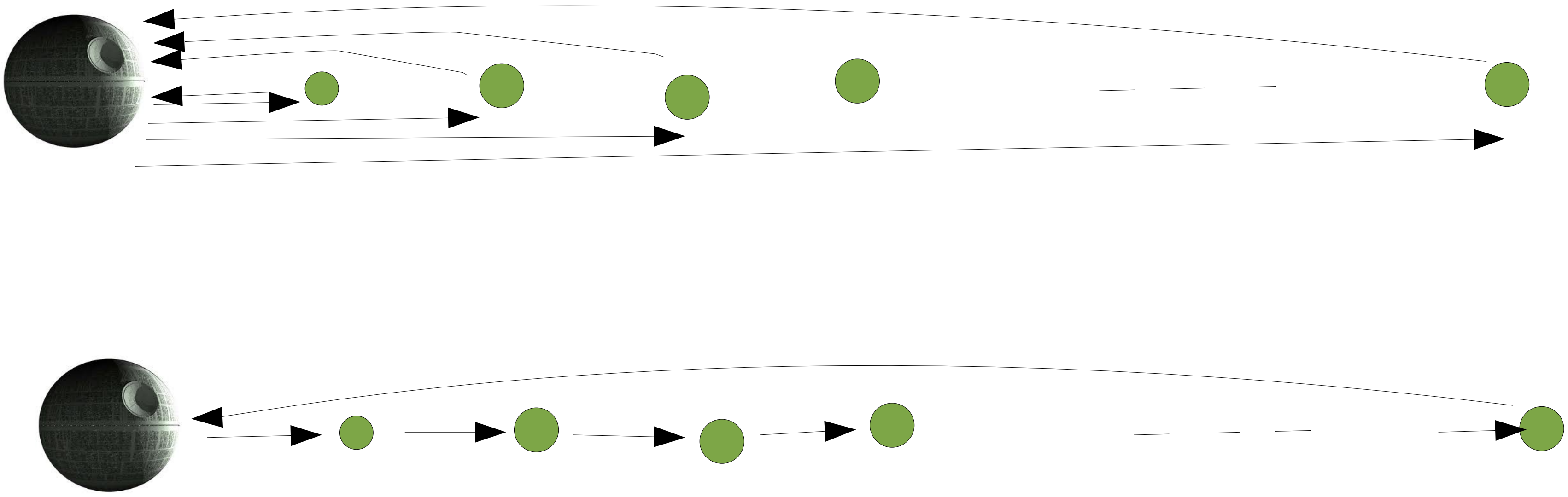
- Potrebno je isporučiti istu količinu tovara ali se ovaj put sve lokacije nalaze na istom pravcu od Zvijezde Smrti ka rubu galaksije na svakih 1 AJ.

Koju letjelicu izabrati za isporuku navedenog tovara?

Šta se dešava kad imamo **n** lokacija, gdje je **n** veliki broj?

Primjer 1b

- Letjelica A
  - morala bi opet ići do svake lokacije i nazad



- Letjelica B
  - opet može utovariti svih 100 kontejnera i isporučiti na sve lokacije bez vraćanja

Primjer 1b – proračun pređenog puta

- U slučaju letjelice A imamo:

$f1 = (1+1)+(2+2)+ (3+3)... +(100+100)=$

U slučaju **n** lokacija bismo imali:

$f1(n) = (1+1)+(2+2)+ (3+3)... +(n+n)=2n(n+1)/2$

$f1(n) = n^2+n$

- U slučaju letjelice B imamo:

$$f2 = \underbrace{1+1+ \ldots +1}_{\text{100-puta (n puta)}} + 100=200$$

U slučaju **n** lokacija bismo imali: $f2(n)=2n$

Primjer 1b - zaključak

- Potrošnja letjelice A:

$(100^2 + 100) * 2$ t/100AJ = 10100 *2/100=202 t goriva

- Potrošnja letjelice B:

200 AJ * 6 t/100AJ = 12 t goriva

- Za **n** lokacija:

Potrošnja A: $p1(n) = 2/100 * f1(n) = 0.02(n^2+n)$

Potrošnja B: $p2(n) = 6/100 * f2(n) = 0.12\ n$

Dakle, u slučaju A imamo kvadratnu zavisnost a u slučaju B opet linearnu.

Za veliko **n** pријеđeni put f1(n) vrlo brzo raste => ukupna količina potrošenog goriva je velika bez obzira na malu relativnu potrošnju.

Darth Wader Transport – usporedba prijeđenog puta

▪

n	$2n$	n^2	$n : n^2$
1	2	1	1
2	4	4	0,5
5	10	25	0,2
10	20	100	0,1
50	100	2500	0,02
100	200	10000	0,01
1.000	2.000	1000.000	0,001
10.000	20.000	100.000.000	0,0001

Primjer 1b - zaključak

- Ako pogledamo izraze za prijeđeni put:

$f1(n) = n^2+n$

$f2(n) = 2n$

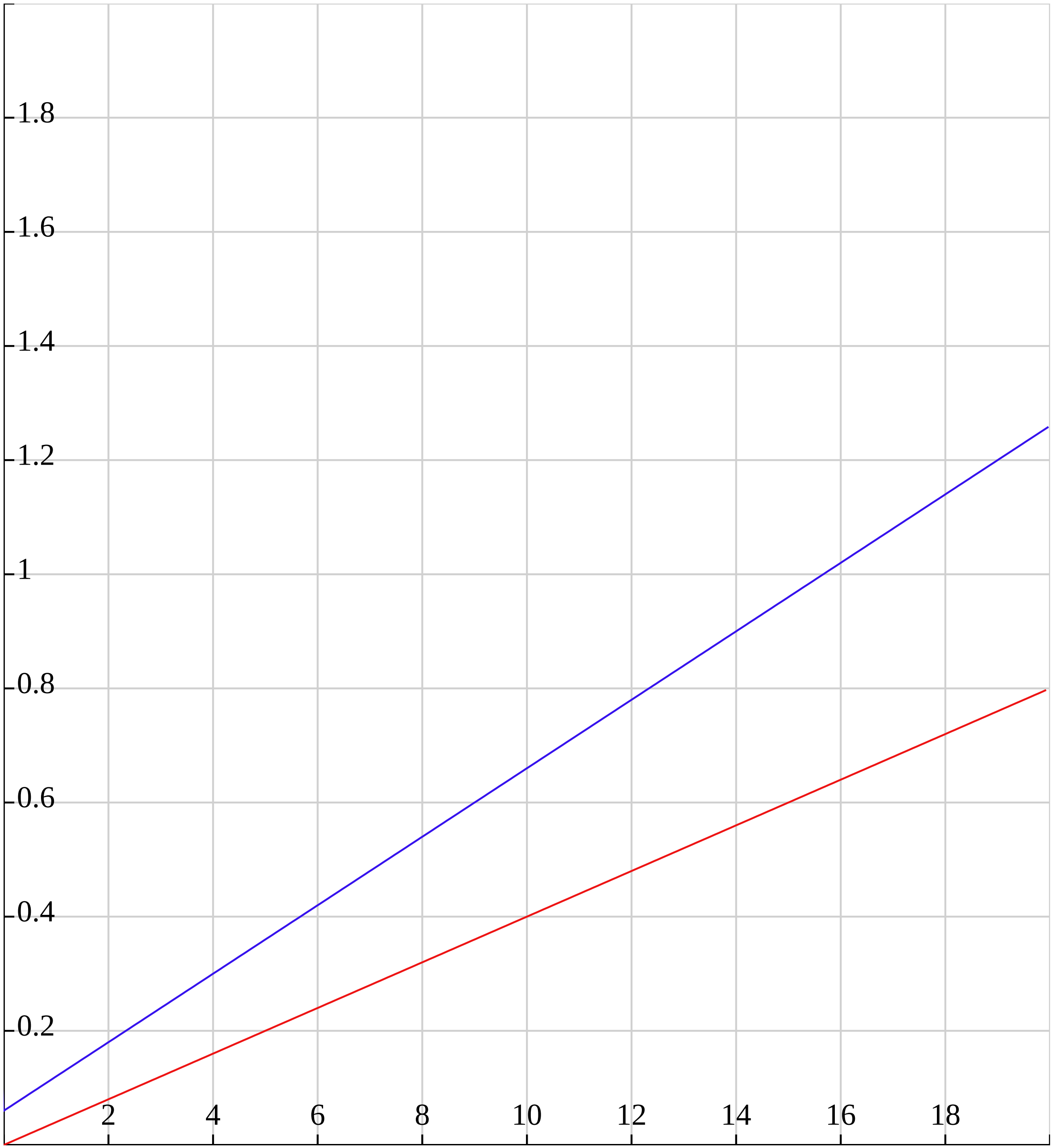
Vidimo da pri velikom **n** izraz **n^2** u izrazu za f1(n) postaje dominantan pa se drugi dio izraza, n, može zanemariti. Kažemo da f1 raste kvadratno u ovisnosti od n.

U f2 vidimo linearnu ovisnost o n.

Možemo zaključiti da će troškovi rasti u linearnoj zavisnosti od n i da nema strmog i naglog porasta vrijednosti funkcije, kao u slučaju f1.

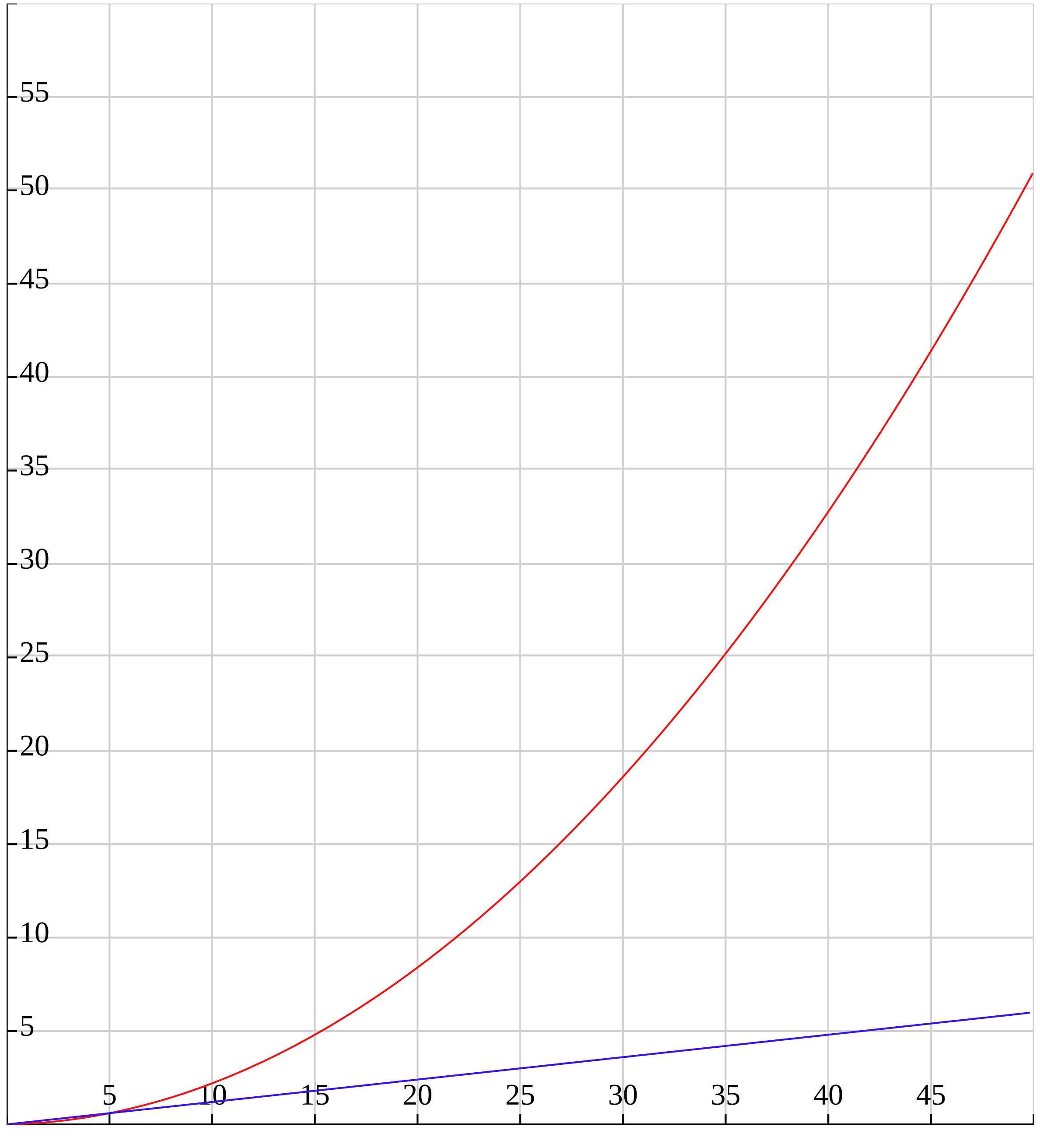
Darth Wader Transport - zaključak

- U primjeru 1a možemo procijeniti da je zavisnost pređenog puta, pa samim time i troškova, linearna u oba slučaja.
- Da bismo tačno procijenili koji metod dostave (algoritam) je jeftiniji (ma šta to značilo), moramo ući u detalje svakog od njih (relativna potrošnja letjelica, troškovi utovara i sl.).
- Nakon analize, ušteda može biti znatna.

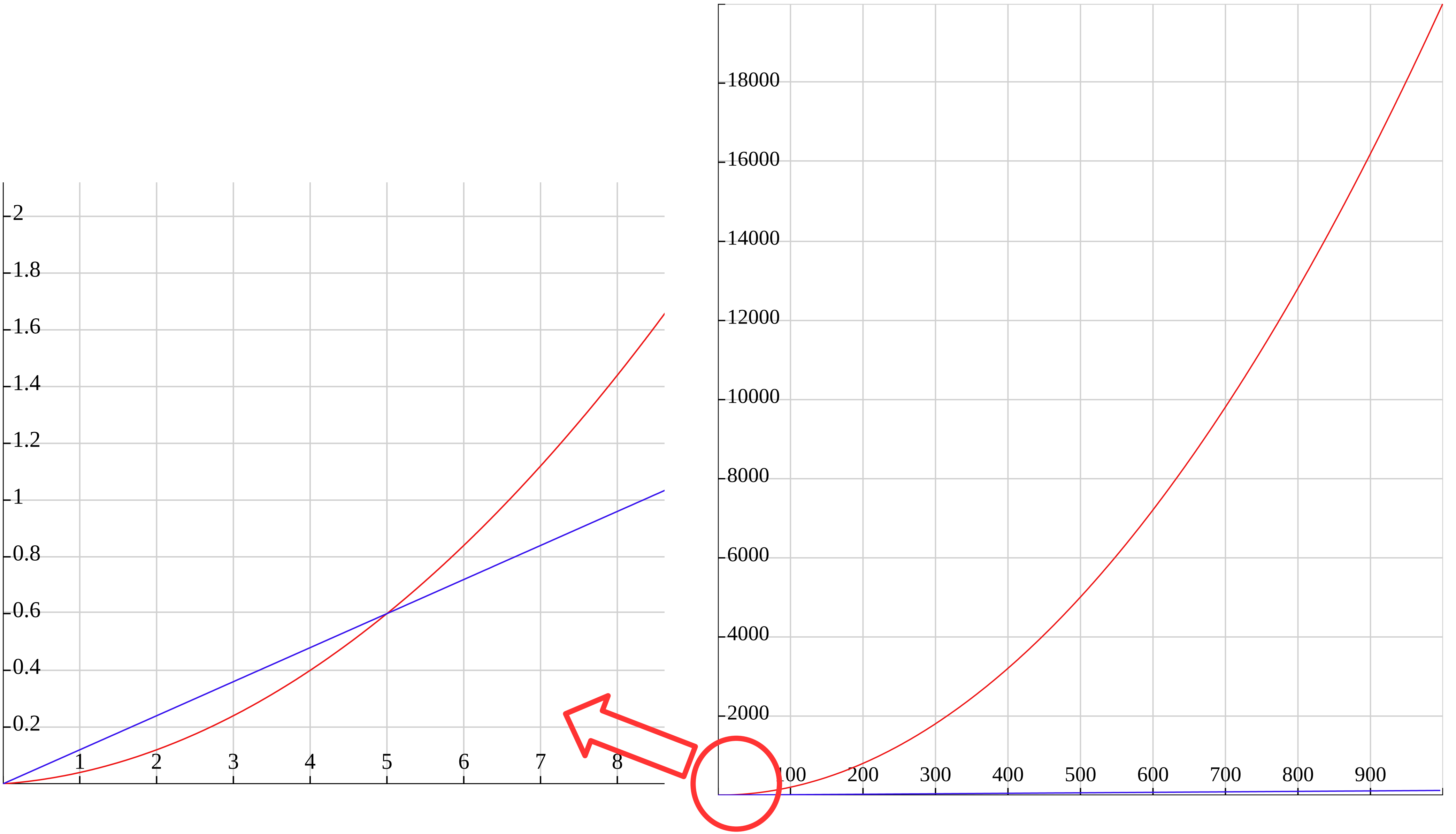


Darth Wader Transport - zaključak

- U primjeru 1b smo vidjeli da pređeni put letjelice A raste kao n^2, dok kod letjelice B ovisi linearno o n.
- Na osnovu karakteristike funkcija n^2 i n procjenjujemo da će za veliko n letjelica (algoritam) B biti **uvijek** povoljnija, bez obzira na relativnu potrošnju i sve ostale troškove.
- Ušteda je ogromna.



Darth Wader Transport - zaključak



Složenost (kompleksnost) algoritma - asimptotska složenost

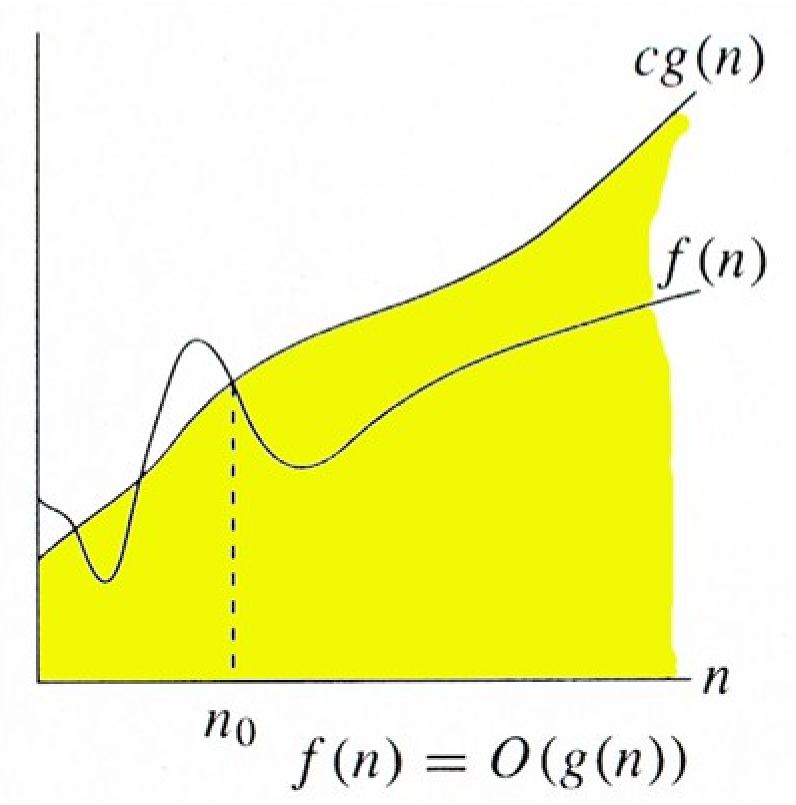
- Služi za procjenu efikasnosti i vremenskog trajanja ili zauzimanja memorijskog prostora algoritma.
- Procjena složenosti, odnosno broja operacija, izražava se u ovisnosti od broja ulaznih podataka (obično oznaka n).
- Postoji više načina označavanja porasta broja operacija u zavisnosti od broja ulaznih podataka (asimptotske granice):
  - O – notacija, za izražavanje gornje granice vremena izvođenja algoritma,
  - Ω – notacija, za izražavanje donje granice vremena izvođenja algoritma,
  - Θ – notacija, za izražavanje vremena izvođenja algoritma kod kojih su isti O i Ω,
  - ostali.

Složenost (kompleksnost) algoritma

- Ove notacije se koriste na način da se porast funkcije (trajanja algoritma) u zavisnosti od broja ulaznih podataka prikaže uz pomoć jednostavnije funkcije tako da je lakše porediti dva algoritma.
- U računarstvu, navedene notacije su se pokazale najkorisnijim.
- Od svih, O-notacija je najviše u upotrebi jer je najpraktičnija i najlakše se određuje.

O-notacija

- Formalna definicija:
- $f(n) \in O(g(n))$ ako postoje dvije pozitivne konstante c i n_0 takve da vrijedi $|f(n)| \leq c\,|g(n)|$ za sve $n \geq n_0$
  - traži se najjednostavnije g(n) za koje to vrijedi



O-notacija – gornja granica

- $f(n) \in O(g(n))$ se može čitati kao:

"Funkcija f(n) je reda najviše g(n)."

- To znači da porast funkcije f(n) nije većeg reda od porasta funkcije g(n).
- U praksi, pri računanju O(g(n)) uzimamo samo najznačajnije članove funkcije f(n) pri velikom n pa se takav račun pojednostavljuje.
- O-notacija daje dobru procjenu skalabilnosti algoritma pri većem broju ulaznih podataka.

O-notacija - primjeri

Konstantno vrijeme trajanja

- Ako algoritam uvijek ima isto vrijeme trajanja, bez obzira na broj ulaznih podataka (npr. koje je prvo slovo u zadanoj rečenici) onda takav algoritam ima O(1).
- Dokaz: ako je C vrijeme trajanja tog algoritma onda možemo napisati:
- $f(n) = C \leq C \cdot 1$ pa je prema definiciji $g(n) = 1$ i imamo složenost O(1)

Linearna složenost

- Suma elemenata cjelobrojnog niza:

```
for(i=0; i<n; i++) suma += x[i];
```

- Potrebno je n iteracija => O(n)

O-notacija - primjeri

- Suma elemenata na parnim indeksima. Niz ima n elemenata :

```
for(i=0; i<n; i+=2) suma += x[i];
```

- Potrebno je n/2 iteracija => O(n) jer je n/2 = 1/2 * n

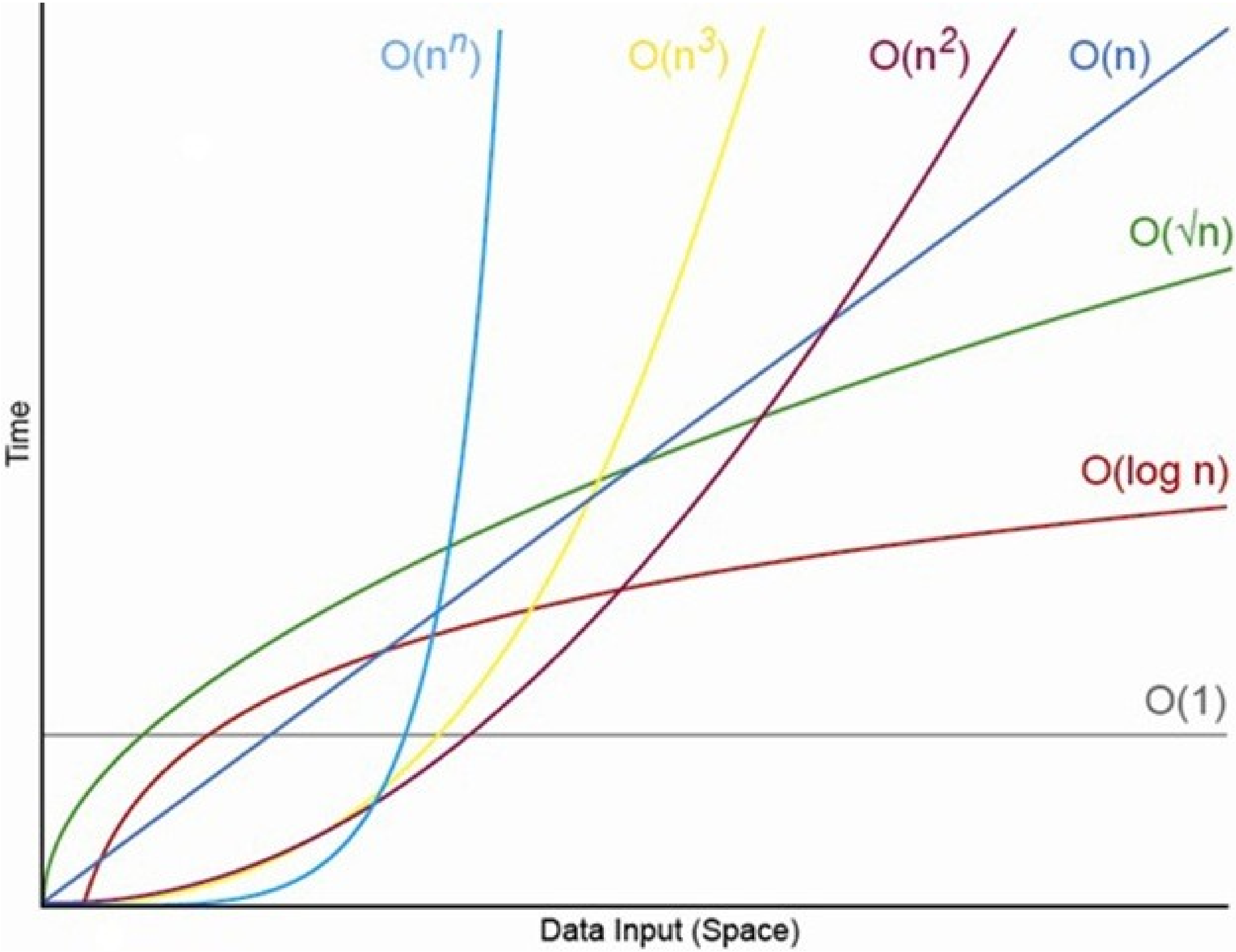
Kvadratna složenost

```
for(i=0; i<n; i++)
{
  for(j=0; j<n; j++)
  {
    //nešto što ima O(1)
  }
}
```

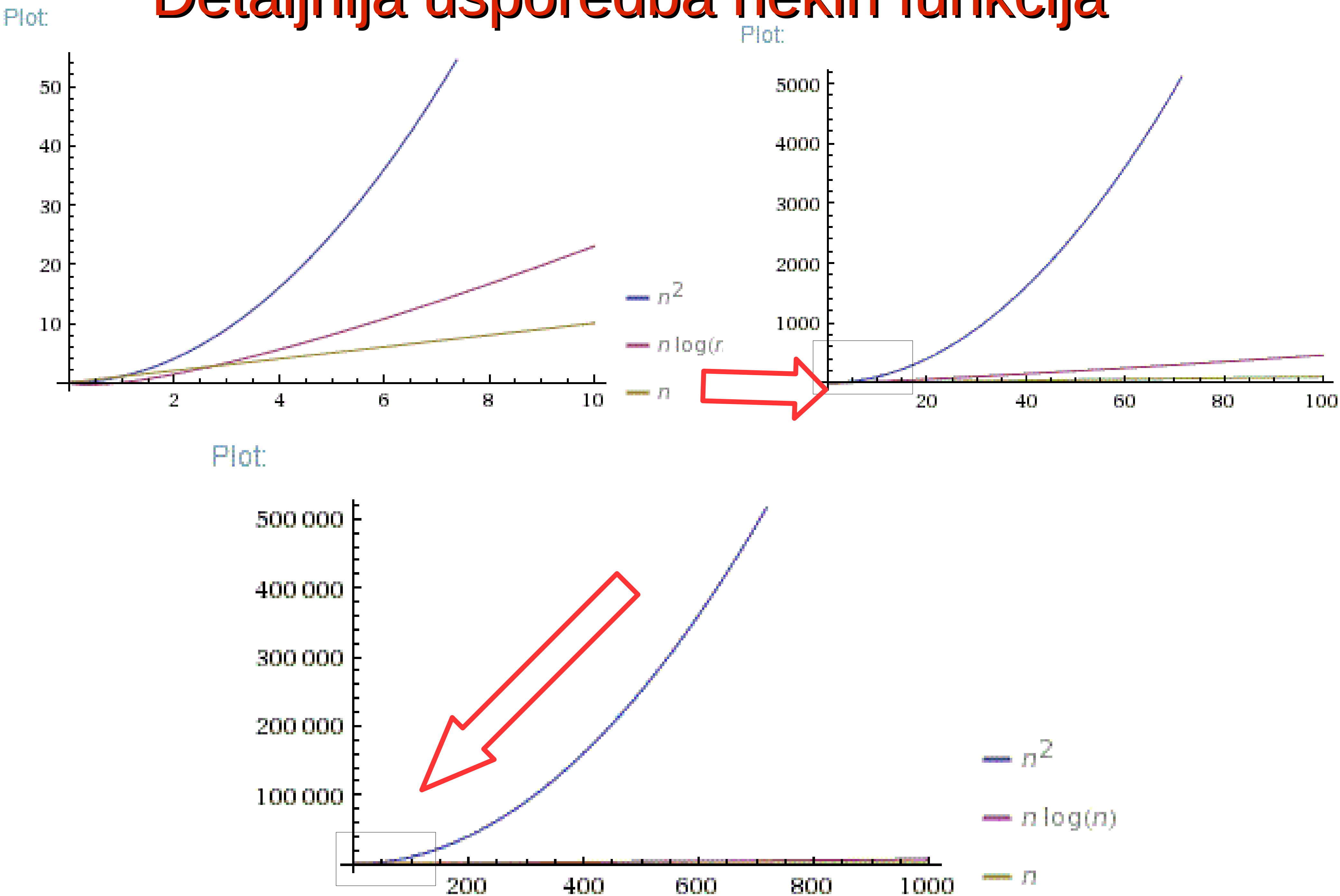
- n*n iteracija = n^2 iteracija => $O(n^2)$

O-notacija – usporedba

$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < \ldots < O(2^n) < O(n!)$

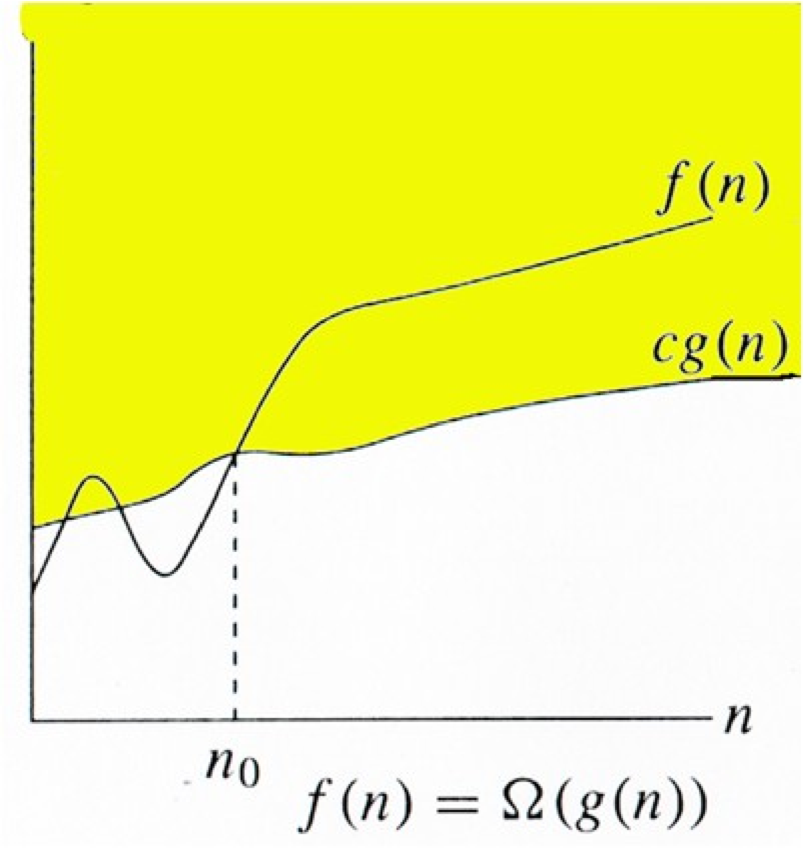


Detaljnija usporedba nekih funkcija



Ω -notacija

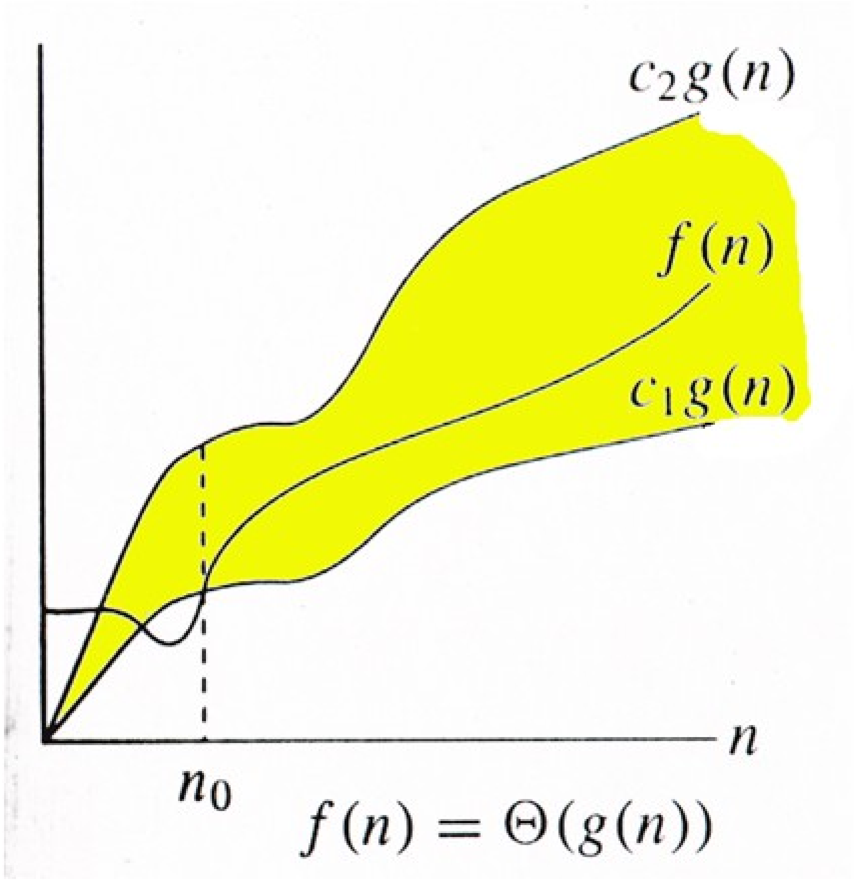
- Formalna definicija:
- f(n) ϵ Ω(g(n)) ako postoje dvije pozitivne konstante c i n_0 takve da vrijedi $|f(n)| \geq c\,|g(n)|$ za sve $n \geq n_0$



- Ovdje g(n) označava rast donje granice vremena trajanja algoritma.
- f(n) ϵ Ω(g(n)) može se čitati i kao: "f(n) je reda najmanje g(n)" kad je n veliki broj.

Θ -notacija

- Formalna definicija:
- $f(n) \in \Theta(g(n))$ ako postoje pozitivne konstante c_1, c_2 i n_0 takve da vrijedi $c_1\,|g(n)| \leq |f(n)| \leq c_2\,|g(n)|$ za sve $n \geq n_0$



- U ovom slučaju je $\Theta(g(n))$ isto kao $O(g(n))$ i isto kao $\Omega(g(n))$.
- $f(n) \in \Theta(g(n))$ može se čitati i kao "f(n) je reda kao g(n)" kad je n veliki broj.

Izbor podataka za analizu algoritma

- Ponašanje u najboljem slučaju (*best case*)
  - ponašanje sistema u optimalnim okolnostima
  - najčešće je ovaj slučaj u stvarnosti ili rijedak ili nerealan
- Ponašanje u najgorem slučaju (*worst case*)
  - najvažnija analiza; pokazuje skalabilnost algoritma u slučaju velikog broja ulaznih podataka ili u nekim kritičnim situacijama
- Prosječno (tipično) ponašanje (average *case*)
  - analiza za slučajne ulaze (zavisi od algoritma); u stvarnosti se algoritam ponaša bolje ili gore ali je potrebno procijeniti neko prosječno ponašanje u realnim uslovima

Primjer za najbolji, najgori i prosječan slučaj

- Analizirajmo gornju granicu vremena nekih operacija za niz od n cijelih brojeva (`int a[n]`)
- Pristup elementu po indeksu (`a[i]`):
  - bc: O(1)
  - wc: O(1)
  - ac: O(1)
- Traženje elementa po vrijednosti:
  - bc (element na početku niza): O(1)
  - wc (element na kraju niza – treba pregledati n elemenata): O(n)
  - ac (element u prosjeku na sredini niza – treba pregledati n/2 elemenata): O(n)