Uvod v računalništvo (UvR)

Učinkovitost algoritmov

Danijel Skočaj
Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Literatura: Invitation to Computer Science, poglavje 3

v1.0
Št. leto 2013/14

Cilji predavanja

- Razumeti atribute algoritmov
- Razumeti smisel analize učinkovitosti algoritmov
- Ugotoviti red časovne kompleksnosti algoritmov
- Znati opisati, ilustrirati in uporabljati obravnavane algoritme
- Razumeti različne rede časovnih kompleksnosti
- Razumeti kaj so nerešljivi problemi in kako jih lahko deloma rešimo z aproksimacijo

Uvod

- Za isti problem lahko obstaja več rešitev
- Kako lahko ocenimo in primerjamo algoritme?
- Kaj je dober algoritem?

- Nakup avtomobila:
  - enostavnost uporabe
  - stil
  - učinkovita poraba goriva
- Ocenjevanje algoritma
  - enostavnost razumevanja
  - eleganca
  - časovna in prostorska učinkovitost

Atributi algoritmov

- Najpomembnejši atributi:
  - pravilnost
  - razumljivost
  - eleganca
  - učinkovitost

Pravilnost algoritmov

- Najpomembnejša lastnost: algoritem mora biti pravilen

- Poznamo problem? Poznamo vse prave rezultate?
- Kaj je pravilen rezultat?
  - PI: 3.14159, ali 3.1416, ali 3.14?

- „Ali rešujemo pravi problem?“
- „Ali rešujemo problem pravilno?“

Razumljivost in eleganca

- Razumljivost, jasnost algoritmov
- Enostavnost za vzdrževanje in nadgrajevanje
- Lažje je preverjati pravilnost
- Algoritmi se bodo uporabljali za druge namene
- Algoritme bodo uporabljali drugi

- Eleganca, lepe rešitve
- Primer: vsota števil od 1 do 100
  - razumljivost?

- Včasih sta si razumljivost in eleganca nasprotujoči
  - Včasih pa ne

Učinkovitost algoritmov

- Učinkovitost algoritma = učinkovita izraba virov
  - časovna učinkovitost/potratnost
  - prostorska učinkovitost/potratnost

- Prostorska učinkovitost
  - količina informacije, ki jo mora algoritem dodatno shraniti
- Časovna učinkovitost
  - merjenje časa izvajanja algoritma
    - s katerim računalnikom?
    - na katerih podatkih?
  - bolj sistematična analiza časovne učinkovitosti algoritmov
- Primerjanje algoritmov (benchmarking)
  - merjenje časa izvajanja algoritmov na standardnih množicah podatkov
  - primerjanje zmogljivosti računalnikov

- Analiza algoritmov: merjenje učinkovitosti algoritmov

Primer: Zaporedno iskanje

- Naloga: v neurejenem seznamu poišči dano vrednost

> *"Given a target value and a random list of values, find the location of the target in the list, if it occurs, by checking each value in the list in turn"*

1. Get values for *NAME*, n, $N_1, \ldots, N_n$ and $T_1, \ldots, T_n$
2. Set the value of i to 1 and set the value of *Found* to NO
3. While (*Found* = NO) and ($i \leq n$) do Steps 4 through 7
4. If *NAME* is equal to the ith name on the list, N_i, then
5. Print the telephone number of that person, T_i
6. Set the value of *Found* to YES

   Else (*NAME* is not equal to N_i)
7. Add 1 to the value of i
8. If (*Found* = NO) then
9. Print the message 'Sorry, this name is not in the directory'
10. Stop

Primer: Zaporedno iskanje

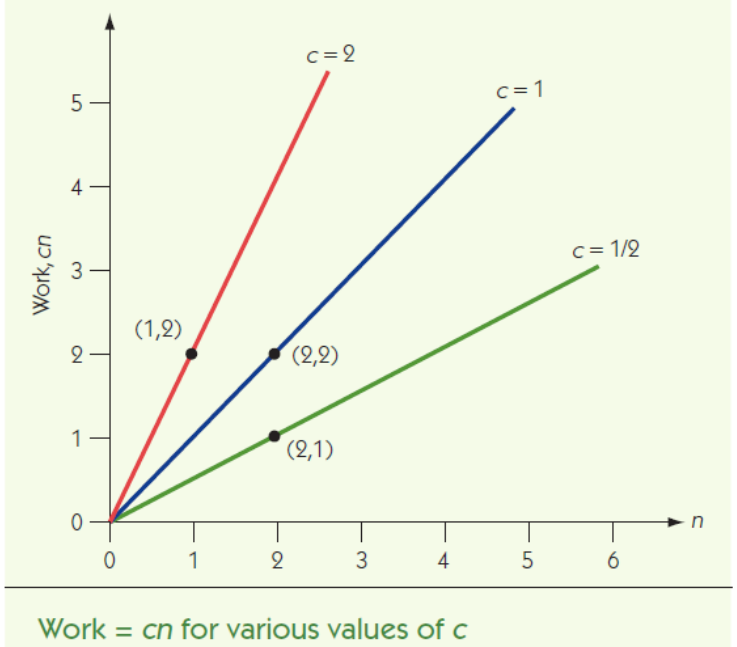
- Štejemo glavne enote dela – operacije, ki so najbolj pomembne za izvršitev naloge in se pogosto pojavljajo
  - primerjanje ciljnega imena NAME z vsakim imenom na seznamu
- Za kakšen vhod (NAME) štejemo operacije
- Obravnavamo tri primere:
  - Najboljši primer (best case) – najmanjša možna zahtevnost
    - iskano ime najdemo s prvim primerjanjem
  - Najslabši primer (worst case) – največja možna zahtevnost
    - iskano ime najdemo z zadnjim primerjanjem (ali ga ne najdemo)
  - Povprečni primer (average case) – odvisna od verjetnosti različnih scenarijev
    - če so vsi primeri enako verjetni: n/2

Best Case	Worst Case	Average Case
1	*n*	*n*/2

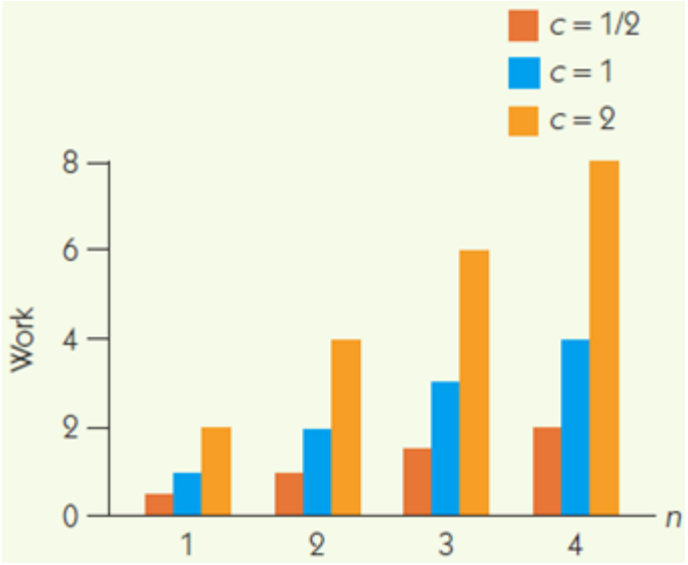
- Zelo pomembno je kako se algoritem obnaša pri velikih n!
- Algoritem za zaporedno iskanje je prostorsko zelo učinkovit

Red velikosti – razred n

- Θ(n): množica funkcij, ki rastejo linearno
  - linearna časovna kompleksnost
  - v odvisnosti od velikosti problema (št. vhodnih podatkov)
  - sprememba v velikosti je konstantna z večanjem n



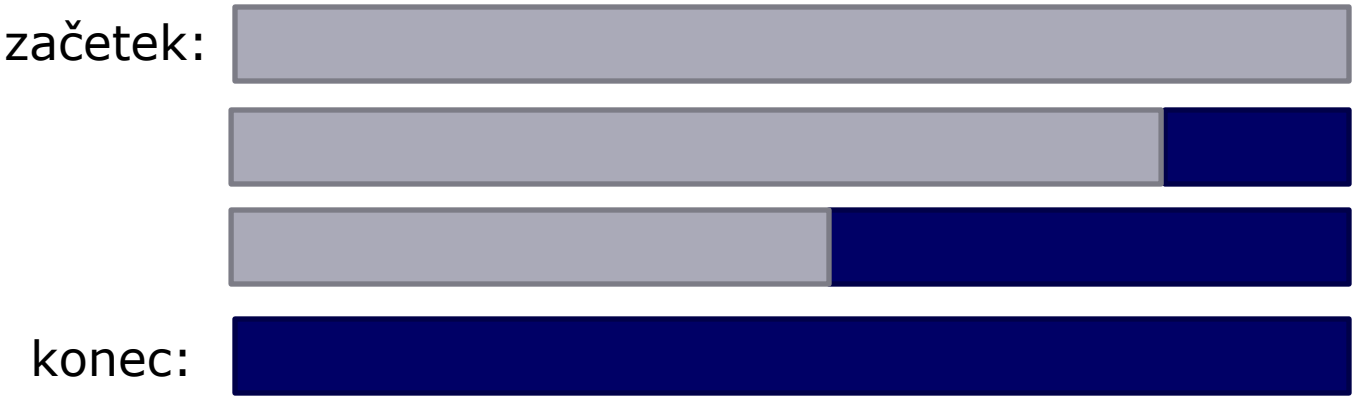
Work = *cn* for various values of *c*



Primer: Sortiranje z izbiranjem

- Sortiranje: naloga urejanja neurejenih elementov v urejeno zaporedje (npr. po velikosti, po abecedi)
- Zelo pogosta naloga, zelo koristne rešitve
- Zelo veliko pristopov (različno učinkovitih)

- Sortiranje z izbiranjem:
  - vedno znova preišči neurejen del seznama
  - v vsakem prehodu izberi največji element
  - premakni izbrani element na konec urejenega seznama



Primer: Sortiranje z izbiranjem

1. Get values for *n* and the *n* list items
2. Set the marker for the unsorted section at the end of the list
3. While the unsorted section of the list is not empty, do Steps 4 through 6
4. Select the largest number in the unsorted section of the list
5. Exchange this number with the last number in the unsorted section of the list
6. Move the marker for the unsorted section left one position
7. Stop

Selection sort algorithm

Primer: Sortiranje z izbiranjem

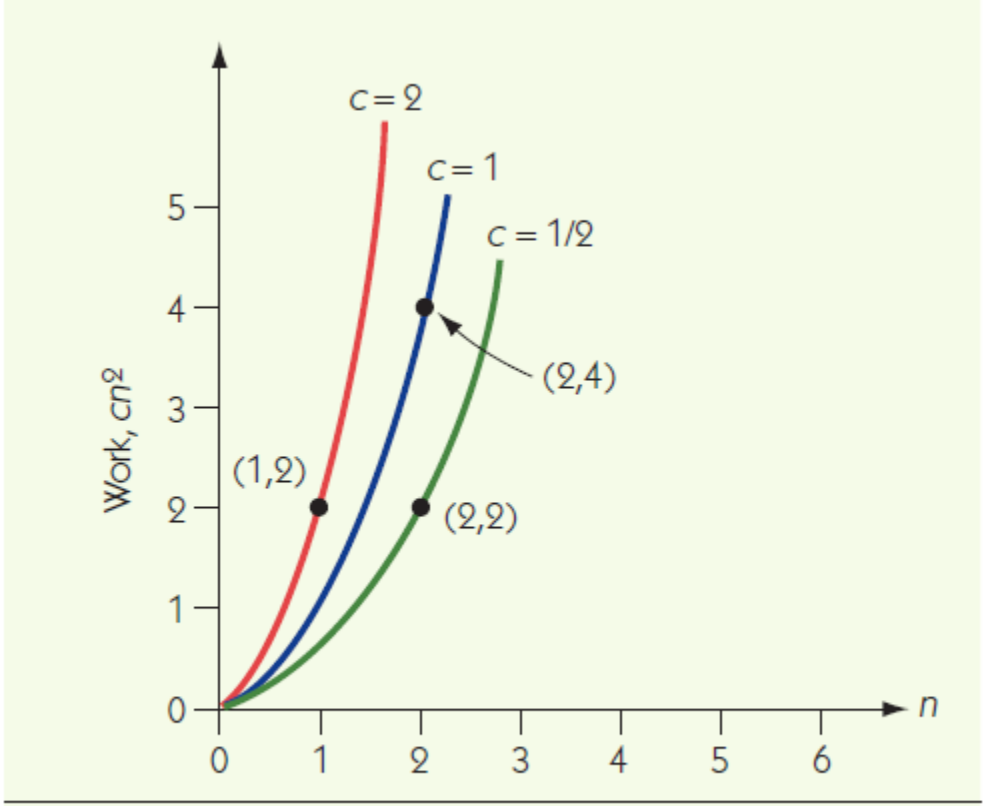
- Glavne enote dela: skrite v koraku „najdi največjega“
- S časom se ta čas manjša:
  - $(n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2$
- Kvadratična časovna kompleksnost $\Theta(n^2)$

Length *n* of List to Sort	n^2	Number of Comparisons Required
10	100	45
100	10,000	4,950
1,000	1,000,000	499,500

Comparisons required by selection sort

Red velikosti – razred n^2

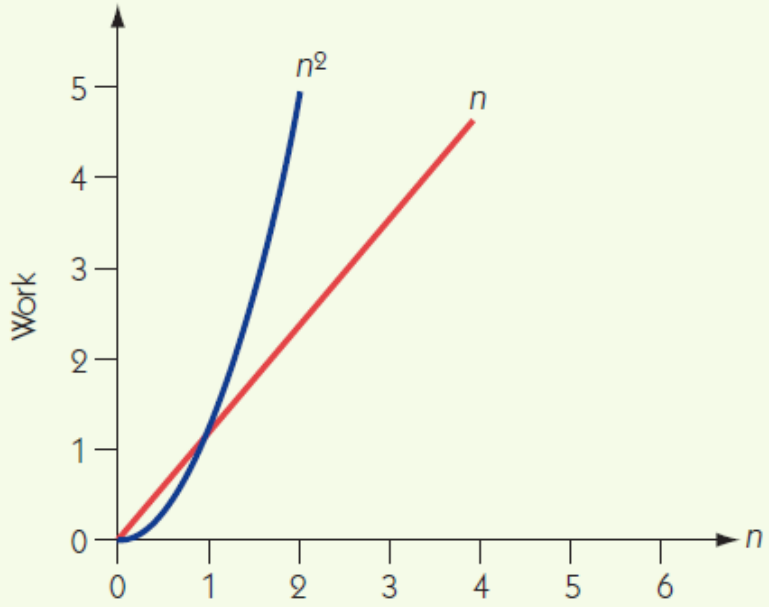
- Red velikosti n^2: $\Theta(n^2)$ – algoritem opravi cn^2 dela, da sprocesira n elementov



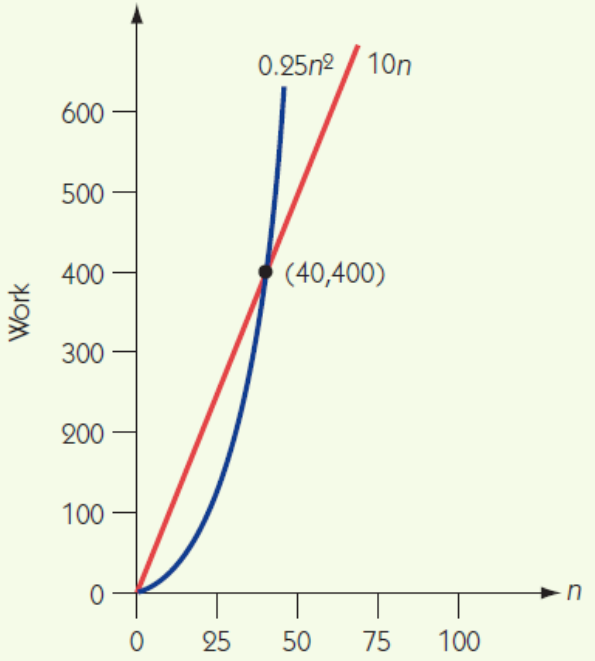
Work = cn^2 for various values of c

Red velikosti – razred n^2

- Vsaka funkcija reda velikosti n^2 ima od neke točke naprej večje vrednosti od funkcije reda velikosti n
- ne glede na konstantni faktor



A comparison of n and n^2



For large enough n, $0.25n^2$ has larger values than $10n$

Red velikosti – razred n^2

- Vsaka funkcija reda velikosti n^2 ima od teke točke naprej večje vrednosti od funkcije reda velikosti n
- zato konstantni faktor zanemarimo

	Number of Work Units Required	
	Algorithm A	Algorithm B
n	$0.0001n^2$	$100n$
1,000	100	100,000
10,000	10,000	1,000,000
100,000	1,000,000	10,000,000
1,000,000	100,000,000	100,000,000
10,000,000	10,000,000,000	1,000,000,000

A comparison of two extreme $\Theta(n^2)$ and $\Theta(n)$ algorithms

Analiza algoritmov

- Primerjamo algoritme, ki opravijo enako nalogo in vrnejo isti rezultat
- Primerjamo časovne kompleksnosti
- Primerjamo prostorske kompleksnosti

Primer: Čiščenje podatkov

- Čiščenje nezaželenih (označenih) podatkov iz niza:

> *"Given a collection of age data, where erroneous zeros occur, find and remove all the zeros from the data, reporting the number of legitimate age values that remain"*

- Tri različne rešitve
- Algoritme bomo primerjali z analizo algoritmov

Primer: Čiščenje podatkov, algoritem 1

- Algoritem 1: Premakni v levo
  - gremo čez seznam
  - ko najdemo 0, pomaknemo vse elemente na desni za en element levo

```
1.  Get values for n and the n data items
2.  Set the value of legit to n
3.  Set the value of left to 1
4.  Set the value of right to 2
5.  While left is less than or equal to legit do Steps 6 through 14
6.      If the item at position left is not 0 then do Steps 7 and 8
7.          Increase left by 1
8.          Increase right by 1
9.      Else (the item at position left is 0) do Steps 10 through 14
10.         Reduce legit by 1
11.         While right is less than or equal to n do Steps 12 and 13
12.             Copy the item at position right into position (right – 1)
13.             Increase right by 1
14.         Set the value of right to (left + 1)
15. Stop
```

Primer: Čiščenje podatkov, algoritem 1

- Časovna učinkovitost:
  - Štejemo primerjave, ko iščemo 0 in premike elementov
  - Najboljši primer:
    - brez 0
    - samo preverjamo vse vrednosti
    - red velikosti $\Theta(n)$
  - Najslabši primer:
    - vse vrednosti so 0
    - najprej premaknemo n-1 elementov, nato n-2 elementov, itn.
    - red velikosti $\Theta(n^2)$

- Prostorska učinkovitost:
  - samo pomožne spremenljivke
  - nič pomembnega razen vhoda

Primer: Čiščenje podatkov, algoritem 2

- Algoritem 2: Kopiraj
  - gremo čez seznam
  - kopiraj dobre elemente na drugi (na začetku prazni) seznam

1. Get values for *n* and the *n* data items
2. Set the value of *left* to 1
3. Set the value of *newposition* to 1
4. While *left* is less than or equal to *n* do Steps 5 through 8
5. If the item at position *left* is not 0 then do Steps 6 and 7
6. Copy the item at position *left* into position *newposition* in new list
7. Increase *newposition* by 1
8. Increase *left* by 1
9. Stop

The copy-over algorithm for data cleanup

Primer: Čiščenje podatkov, algoritem 2

- Časovna učinkovitost:
  - Štejemo primerjave, ko iščemo 0 in premike elementov
  - Najboljši primer:
    - brez 0
    - samo preverjamo vse vrednosti, jih ne kopiramo
    - red velikosti $\Theta(n)$
  - Najslabši primer:
    - vse vrednosti so 0
    - preverimo vse vrednosti in vse kopiramo
    - red velikosti $\Theta(n)$

- Prostorska učinkovitost:
    - Najboljši primer: vse 0, ne potrebujemo nič dodatnega prostora
    - Najslabši primer: nobene 0, potrebujemo n dodatnega prostora

- Kompromis med časovno in prostorsko učinkovitostjo

Primer: Čiščenje podatkov, algoritem 3

- Algoritem 3: Bližanje kazalcev
  - levi kazalec gre proti desni in se ustavi na 0
  - desni kazalec gre z desne proti levi
  - ko je najdena vrednost 0 se zamenjata vrednosti
  - ustavi, ko se kazalca srečata

1. Get values for *n* and the *n* data items
2. Set the value of *legit* to *n*
3. Set the value of *left* to 1
4. Set the value of *right* to *n*
5. While *left* is less than *right* do Steps 6 through 10
6. If the item at position *left* is not 0 then increase *left* by 1
7. Else (the item at position *left* is 0) do Steps 8 through 10
8. Reduce *legit* by 1
9. Copy the item at position *right* into position *left*
10. Reduce *right* by 1
11. If the item at position *left* is 0, then reduce *legit* by 1
12. Stop

Primer: Čiščenje podatkov, algoritem 3

- Časovna učinkovitost:
  - Štejemo primerjave, ko iščemo 0 in premike elementov
  - Najboljši primer:
    - brez 0
    - levi kazalec gre gladko do desnega, samo preverjamo vse vrednosti
    - red velikosti $\Theta(n)$
  - Najslabši primer:
    - vse vrednosti so 0
    - preverimo vse vrednosti in vse kopiramo z desne
    - red velikosti $\Theta(n)$

- Prostorska učinkovitost:
  - samo pomožne spremenljivke
  - nič pomembnega razen vhoda

- Ta verzija je tako časovno kot prostorsko zelo učinkovita

Primer: Čiščenje podatkov

	1. Shuffle-left		2. Copy-over		3. Converging-pointers	
	Time	Space	Time	Space	Time	Space
Best case	$\Theta(n)$	n	$\Theta(n)$	n	$\Theta(n)$	n
Worst case	$\Theta(n^2)$	n	$\Theta(n)$	$2n$	$\Theta(n)$	n
Average case	$\Theta(n^2)$	n	$\Theta(n)$	$n \leq x \leq 2n$	$\Theta(n)$	n

Analysis of three data cleanup algorithms

Primer: Binarno iskanje

- Iskanje vrednosti po urejenem seznamu
- Preglej srednji element in potem preišči ustrezno polovico preostalih podatkov

> *"Given a target value and an ordered list of values, find the location of the target in the list, if it occurs, by starting in the middle and splitting the range in two with each comparison"*

Primer: Binarno iskanje

1. Get values for *NAME*, n, $N_1, \ldots, N_n$ and $T_1, \ldots, T_n$
2. Set the value of *beginning* to 1 and set the value of *Found* to NO
3. Set the value of *end* to n
4. While *Found* = NO and *beginning* is less than or equal to *end* do Steps 5 through 10
5. Set the value of m to the middle value between *beginning* and *end*
6. If *NAME* is equal to N_m, the name found at the midpoint between *beginning* and *end*, then do Steps 7 and 8
7. Print the telephone number of that person, T_m
8. Set the value of *Found* to YES
9. Else if *NAME* precedes N_m alphabetically, then set *end* = $m - 1$
10. Else (*NAME* follows N_m alphabetically) set *beginning* = $m + 1$
11. If (*Found* = NO) then print the message 'I am sorry but that name is not in the directory'
12. Stop

Binary search algorithm (list must be sorted)

Primer: Binarno iskanje

- Časovna učinkovitost:
  - Štejemo primerjave s ciljno vrednostjo
  - Najboljši primer:
    - iskana vrednost je že prva sredinska vrednost
    - samo 1 primerjava
  - Najslabši primer:
    - vrednosti ni na seznamu
    - ponovi toliko krat kolikor lahko razdeliš seznam na pol
    - red velikosti $\Theta(\log n)$

- Prostorska učinkovitost:
  - samo pomožne spremenljivke
  - nič pomembnega razen vhoda

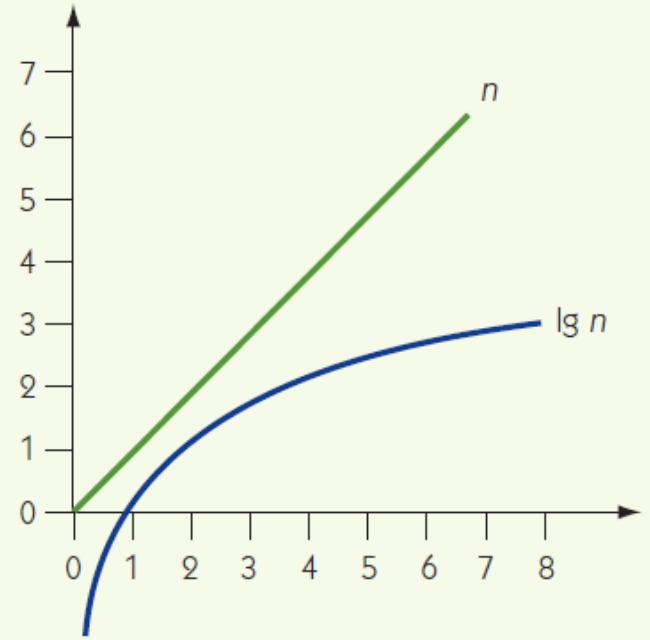
- Nismo žrtvovali prostorske za časovno učinkovitost
- Smo pa žrtvovali splošnost – potrebujemo urejen seznam!

Red velikosti – razred log n

- Red velikosti log n: Θ(log n) – algoritem opravi le c*log(n) dela, da sprocesira n elementov
  - narašča zelo počasi

n	lg *n*
8	3
16	4
32	5
64	6
128	7

Values for *n* and lg *n*



A comparison of *n* and lg *n*

Primer: Ujemanje vzorcev

- Ujemanje vzorcev

```
Get values for n and m, the size of the text and the pattern, respectively
Get values for both the text T1 T2 ... Tn and the pattern P1 P2 ... Pm
Set k, the starting location for the attempted match, to 1
While (k ≤ (n − m + 1)) do
    Set the value of i to 1
    Set the value of Mismatch to NO
    While both (i ≤ m) and (Mismatch = NO) do
        If Pi ≠ Tk+(i−1) then
            Set Mismatch to YES
        Else
            Increment i by 1 (to move to the next character)
    End of the loop
    If Mismatch = NO then
        Print the message 'There is a match at position'
        Print the value of k
    Increment k by 1
End of the loop
Stop, we are finished
```

Primer: Ujemanje vzorcev

- Časovna učinkovitost:
  - Štejemo primerjave
  - Najboljši primer:
    - vzorca ni v besedilu
    - prvega znaka vzorca ni v besedilu
    - KLMNPQRST <- ABC
    - samo n-m+1 primerjav
    - m<<n => Θ(n)
  - Najslabši primer:
    - vzorca ni v besedilu, vsi znaki vzorca razen zadnjega so v besedilu
    - AAAAAAAAAA <- AAAB
    - vzorec je v besedilu, na vsaki lokaciji
    - AAAAAAAAAA <- AAAA
    - samo m(n-m+1) primerjav
    - m<<n => Θ(m*n)

Primeri

Problem	Unit of Work	Algorithm	Best Case	Worst Case	Average Case
Searching	Comparisons	Sequential search	1	$\Theta(n)$	$\Theta(n)$
		Binary search	1	$\Theta(\lg n)$	$\Theta(\lg n)$
Sorting	Comparisons and exchanges	Selection sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Data cleanup	Examinations and copies	Shuffle-left	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
		Copy-over	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
		Converging-pointers	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Pattern matching	Character comparisons	Forward march	$\Theta(n)$	$\Theta(m \times n)$	

Order-of-magnitude time efficiency summary

Red velikosti – razred k^n

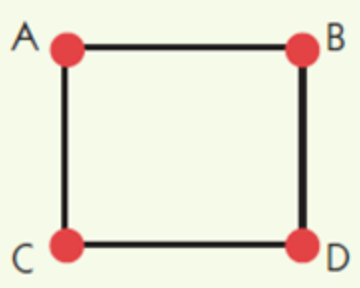
- Večina algoritmov je polinomsko omejenih (reda n^k)
- Nekateri niso
  - ne moremo jih rešiti v polinomskem času, niti pri velikih k!

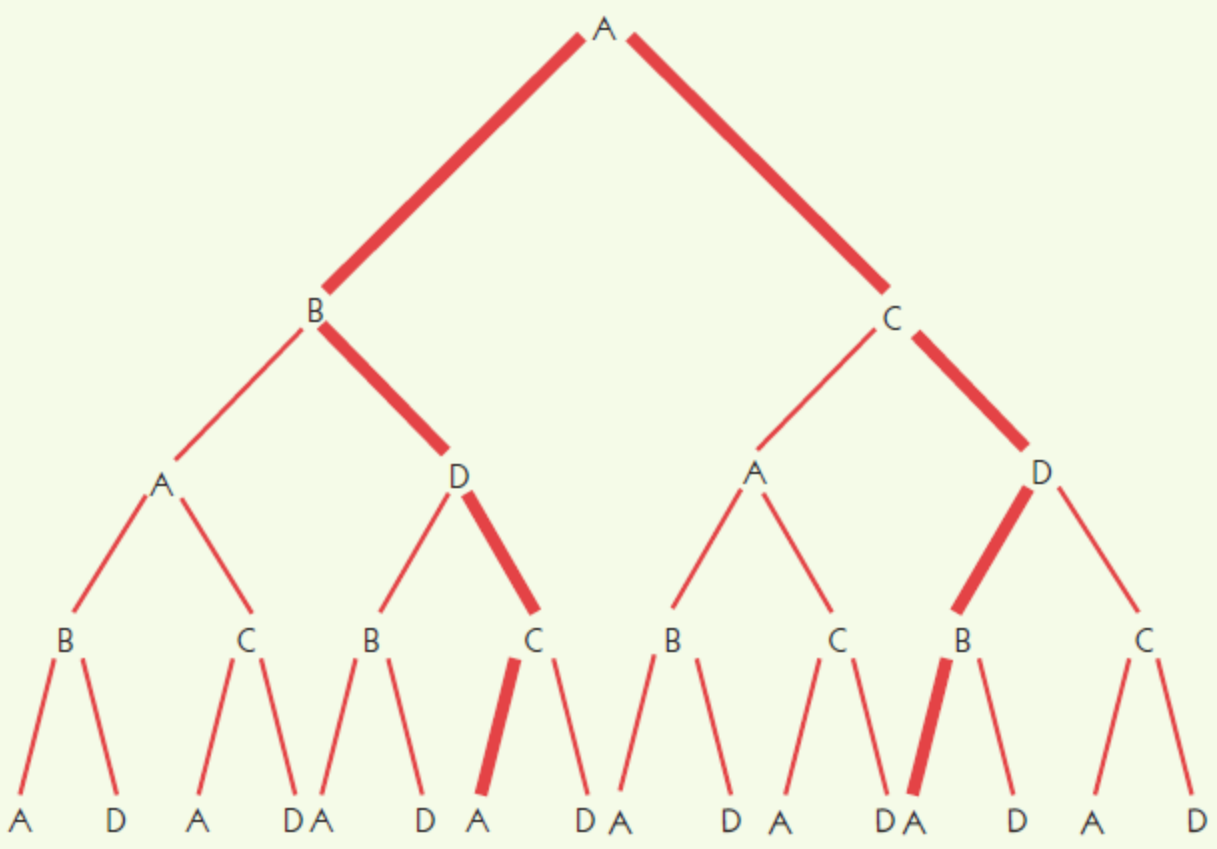
- Eksponentna časovna kompleksnost: $\Theta(k^n)$

Primer: Hamiltonov cikel

- Poišči Hamiltonov cikel:

> *"Given a graph, find a path that passes through each vertex exactly once and returns to its starting point."*

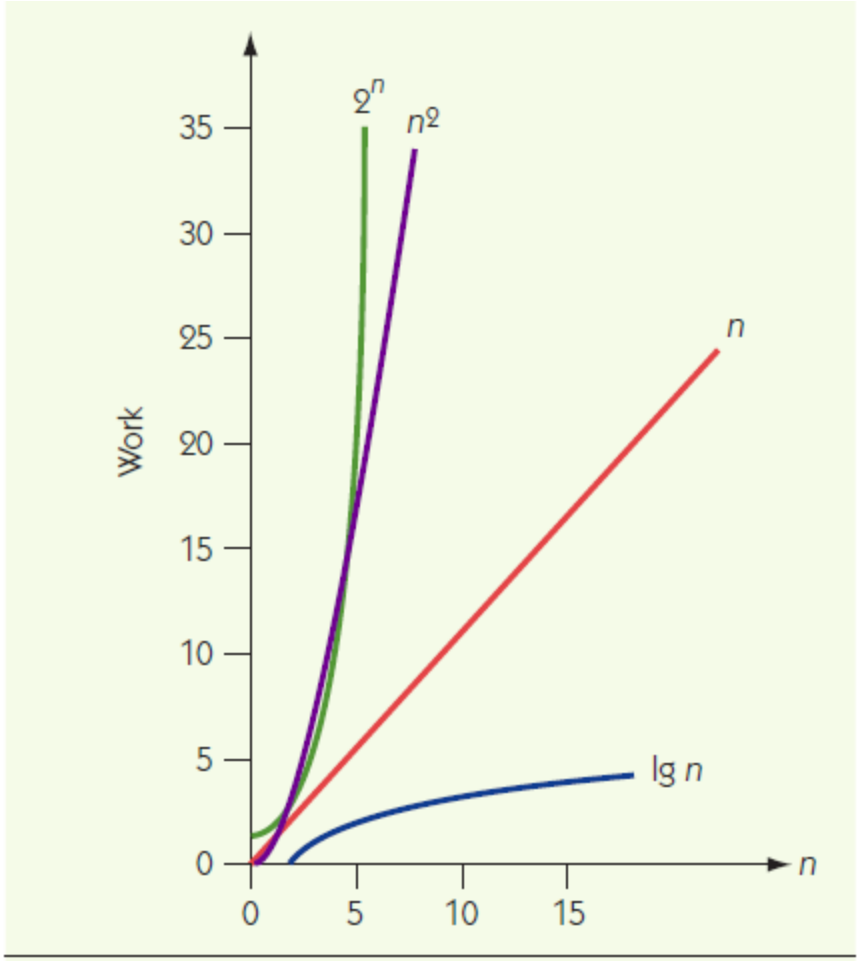




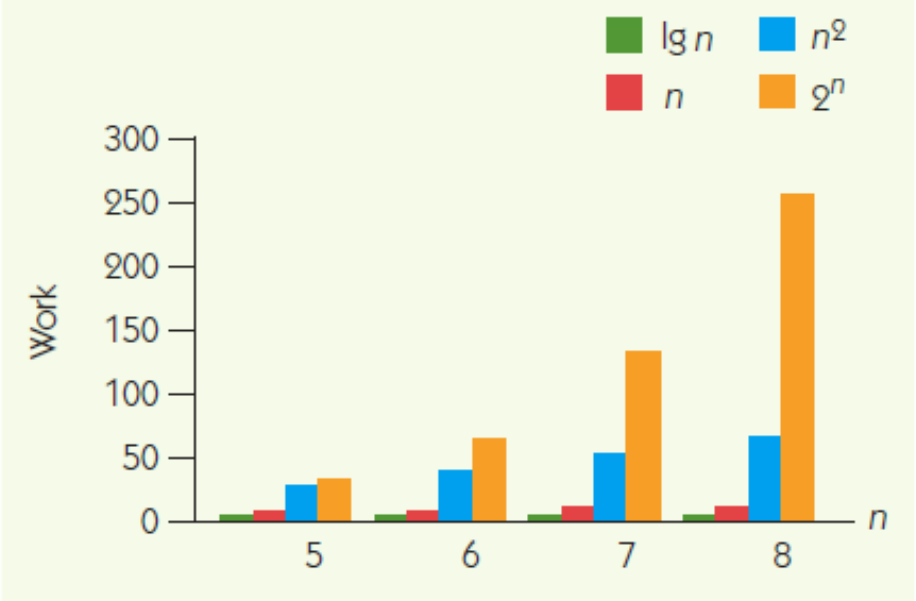
Red velikosti – razred k^n

- Ogromno število pregledovanj:
  - višina drevesa: n+1
  - število listov: 2^n = število poti, ki jih moramo pregledati
- Eksponentni algoritem: $\Theta(k^n)$
- Praktično nedosegljive rešitve (ang. intractable)
  - problemi , ki niso polinomsko omejeni
    - Hamiltonov cikel
    - Problem trgovskega potnika
    - Polnjenje nahrbtnikov
    - Šah

Primerjave redov velikosti



Comparison of lg n, n, n^2, and 2^n



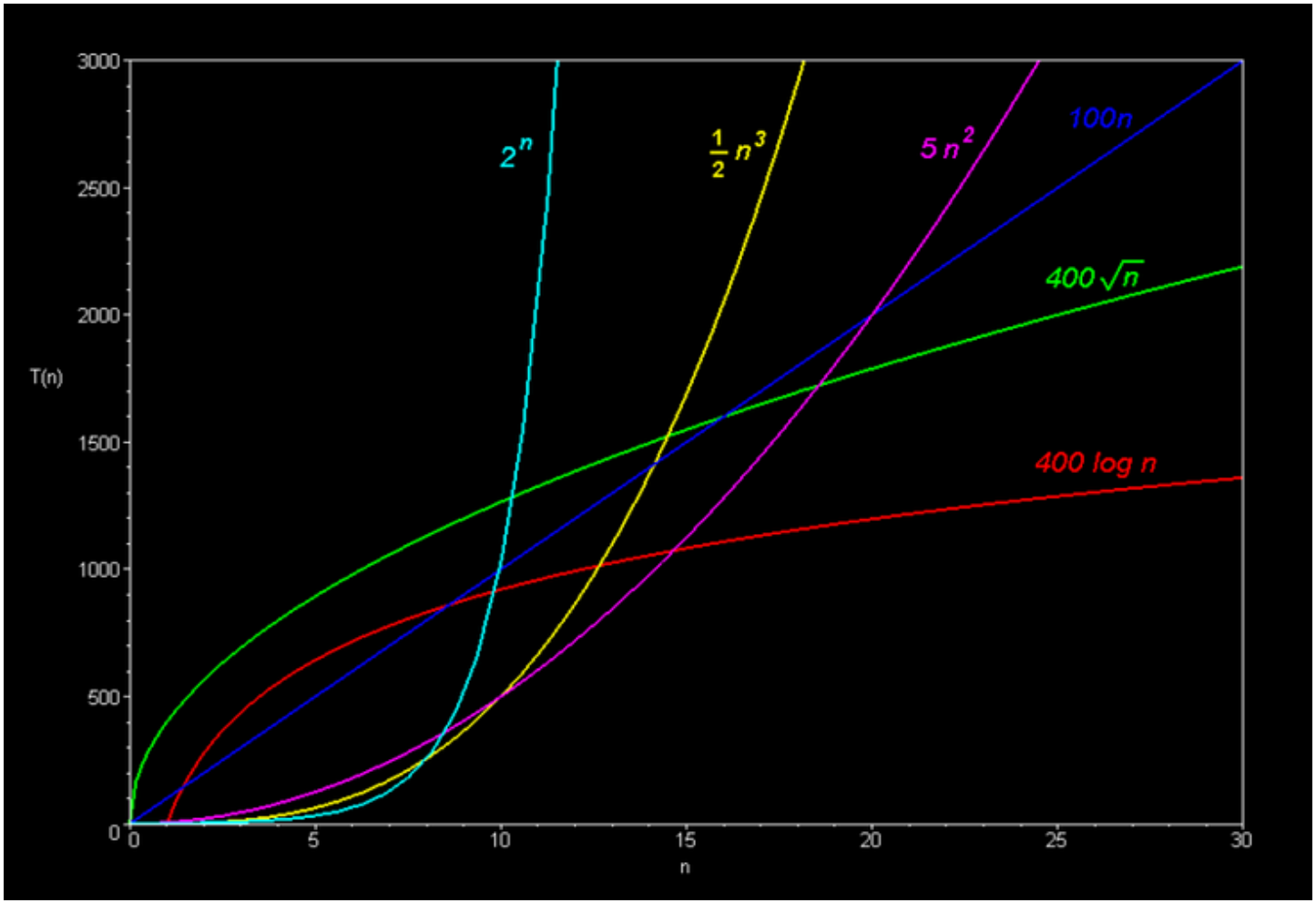
Comparisons of lg n, n, n^2, and 2^n for larger values of n

Primerjave redov velikosti

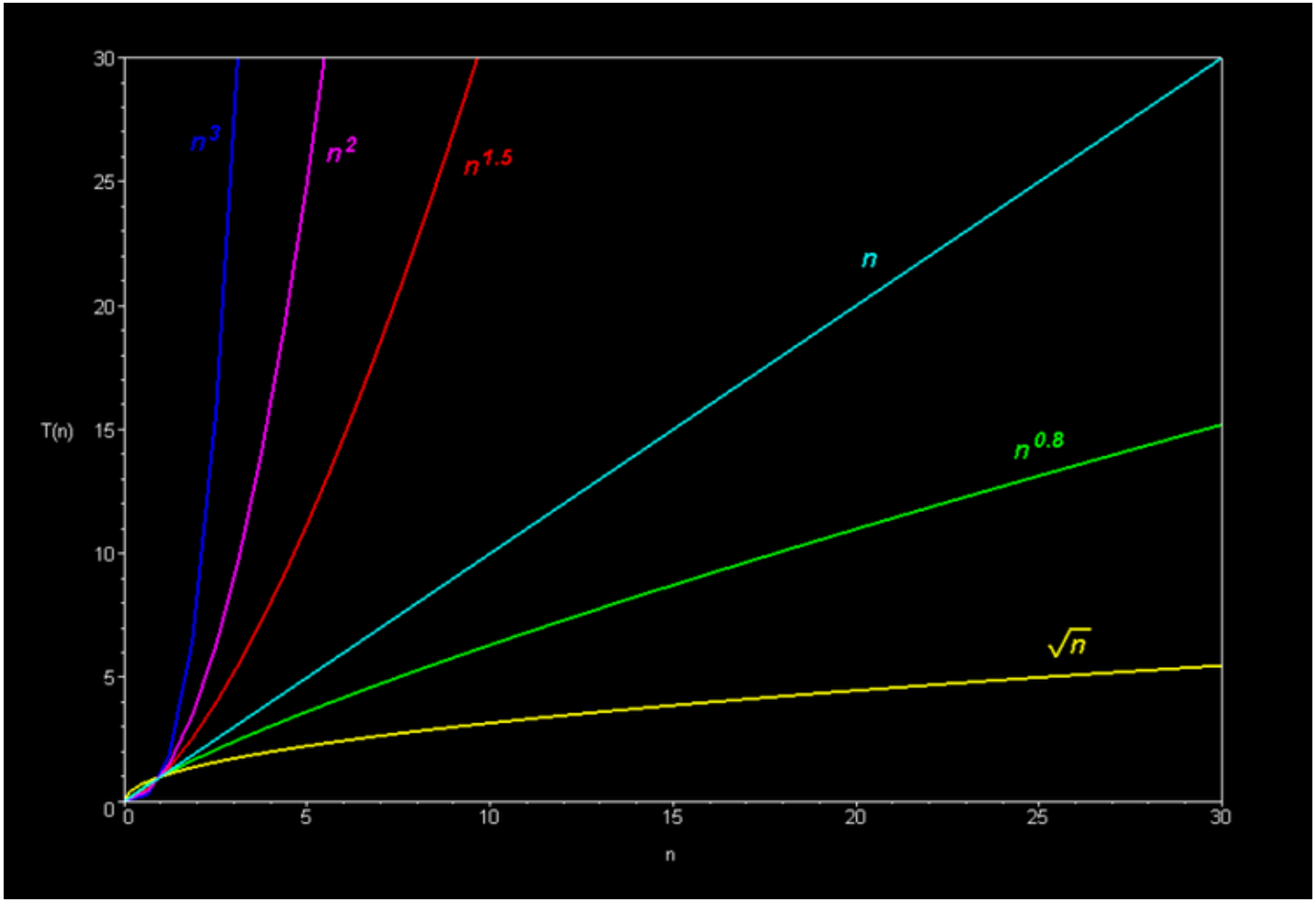
Order	n = 10	n = 50	n = 100	n = 1,000
lg *n*	0.0003 sec	0.0006 sec	0.0007 sec	0.001 sec
n	0.001 sec	0.005 sec	0.01 sec	0.1 sec
n^2	0.01 sec	0.25 sec	1 sec	1.67 min
2^n	0.1024 sec	3,570 years	4×10^{16} centuries	*Too big to compute!!*

A comparison of four orders of magnitude

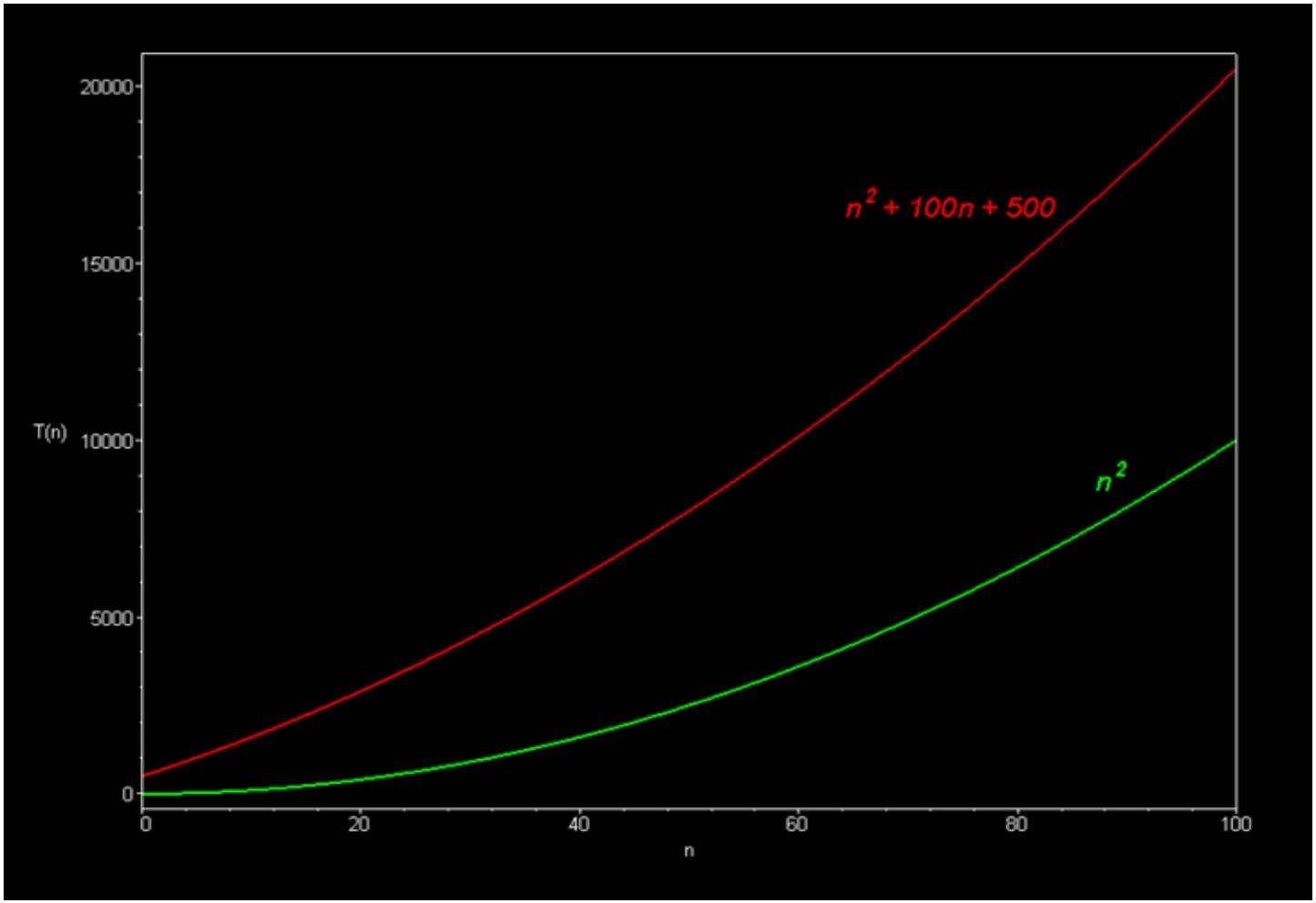
Primerjave redov velikosti



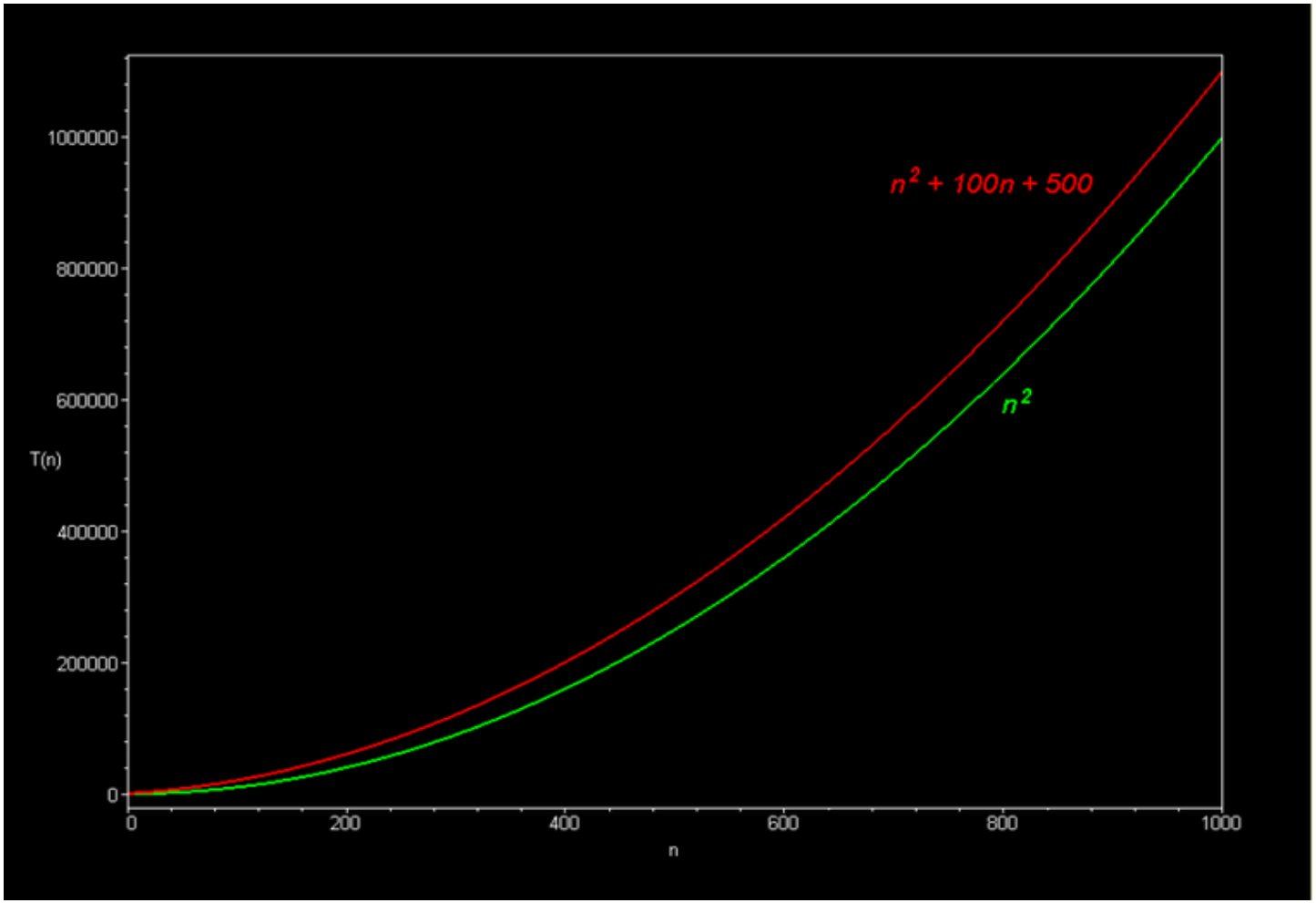
Primerjave redov velikosti



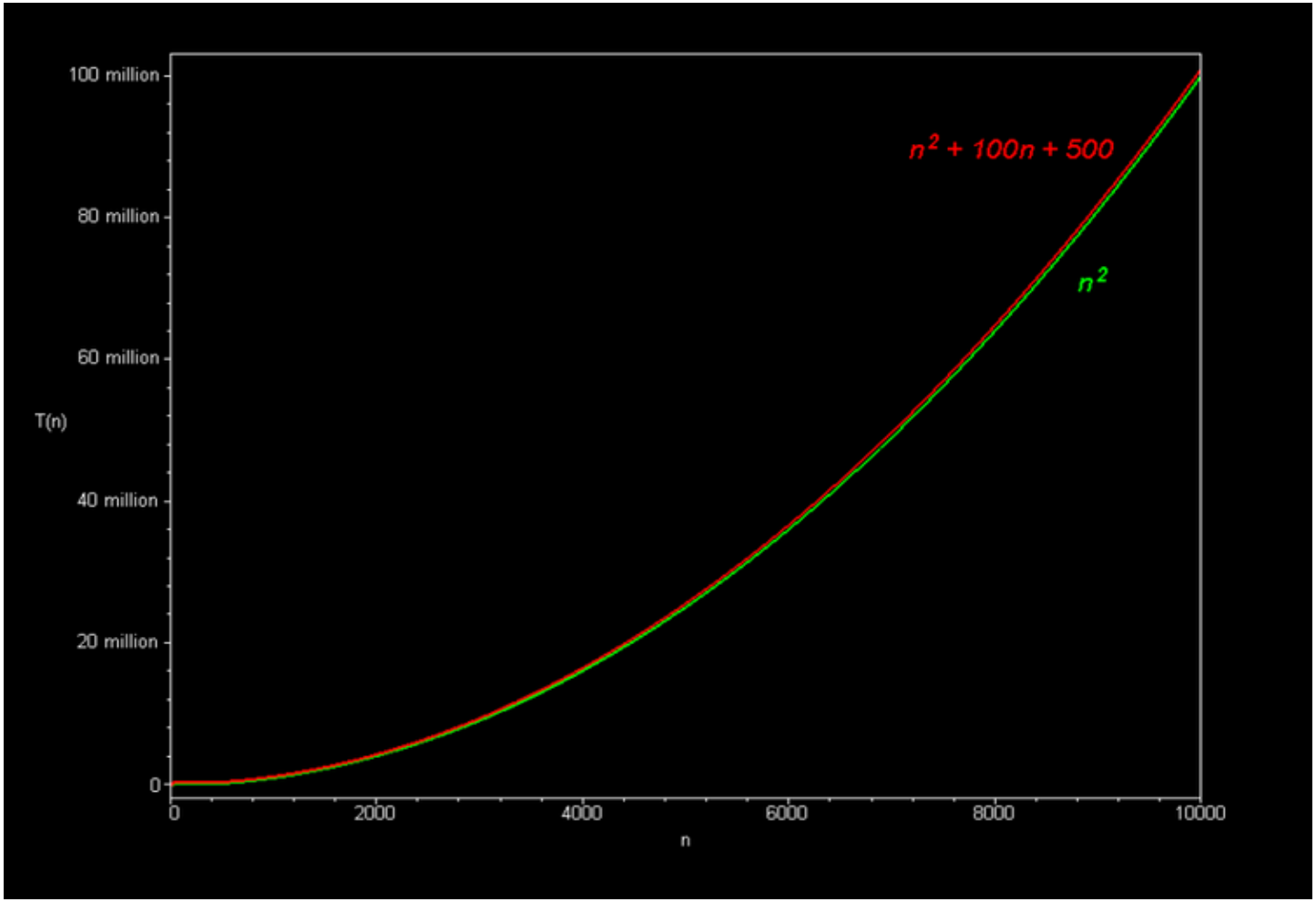
Primerjave redov velikosti



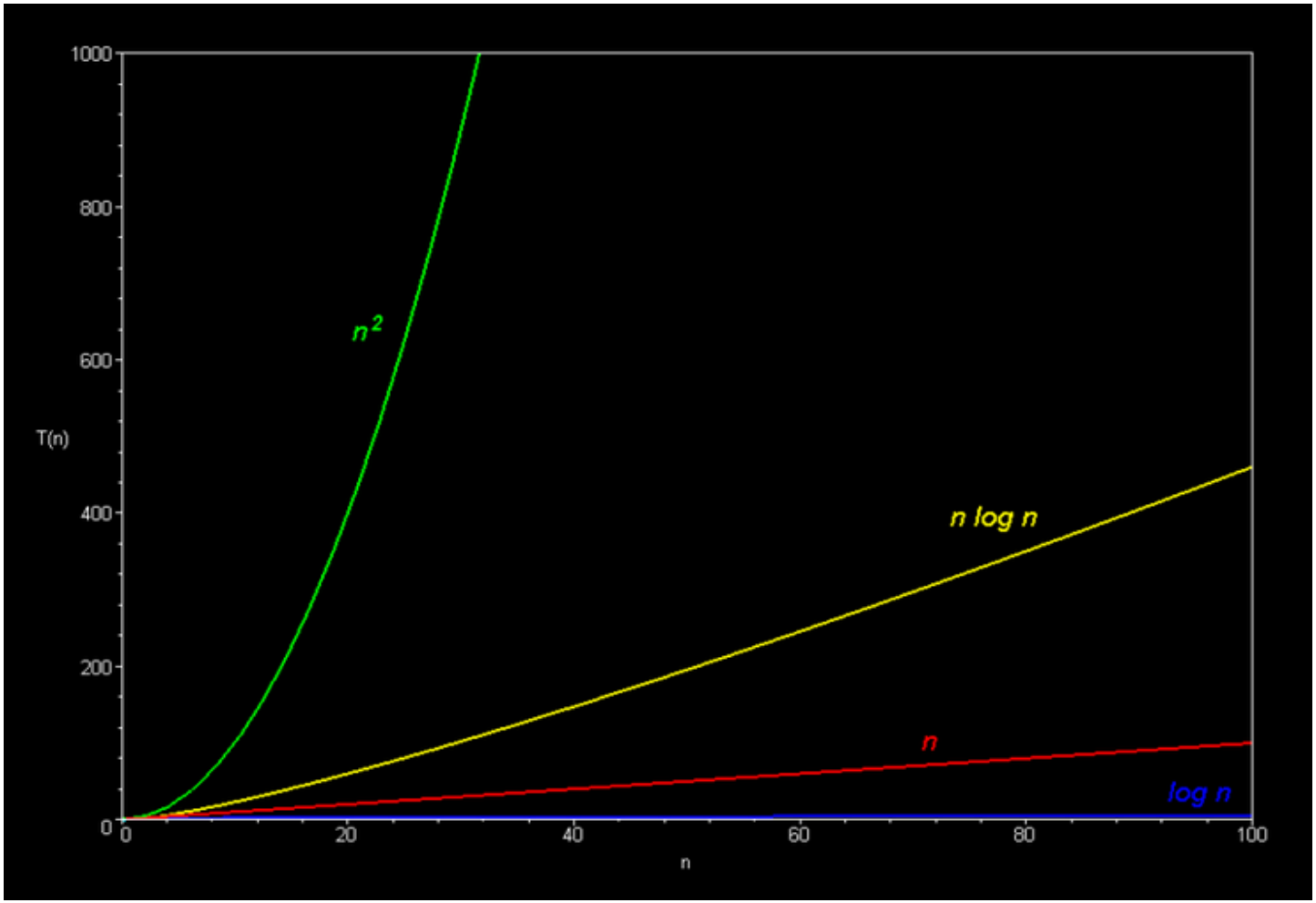
Primerjave redov velikosti



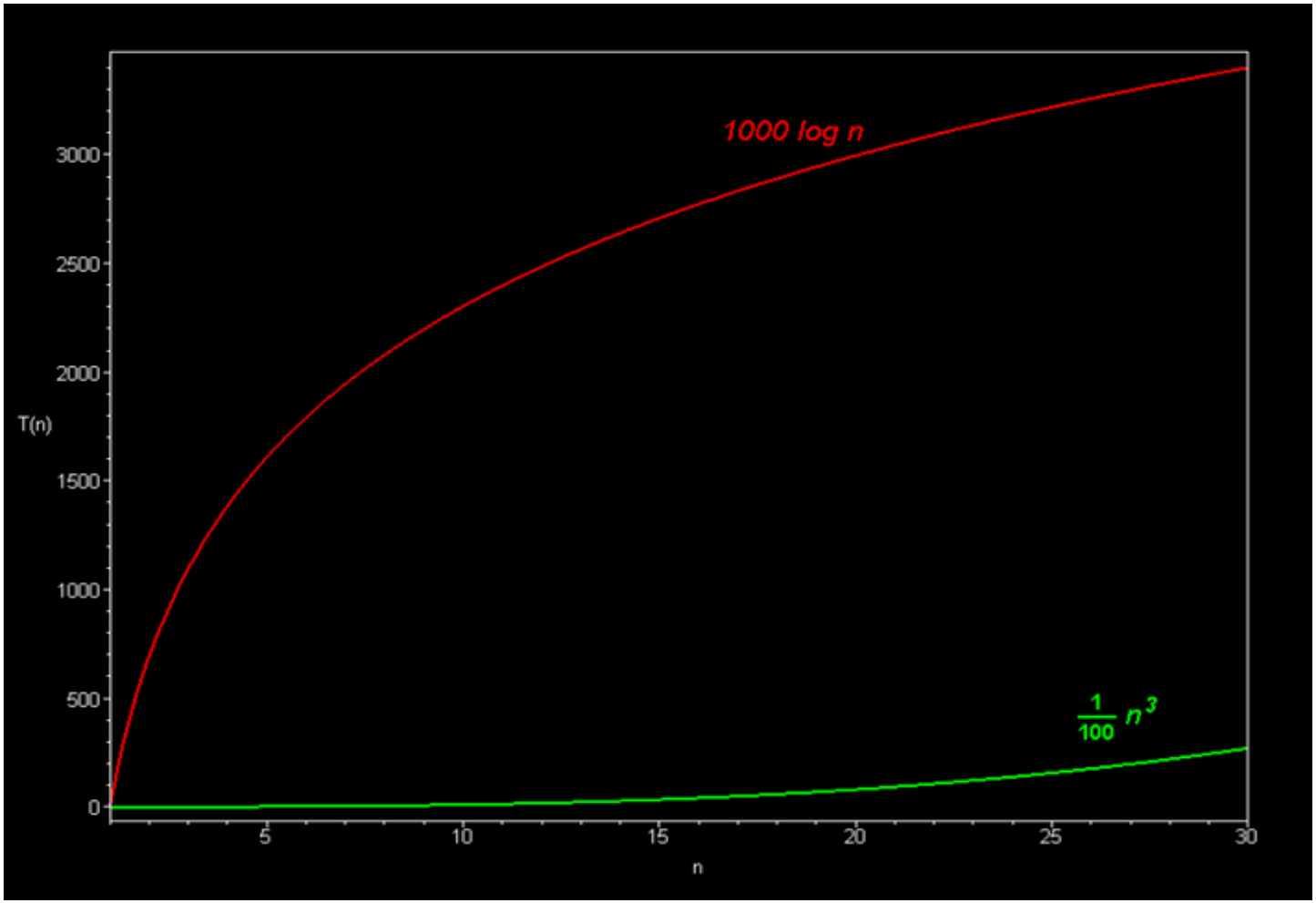
Primerjave redov velikosti



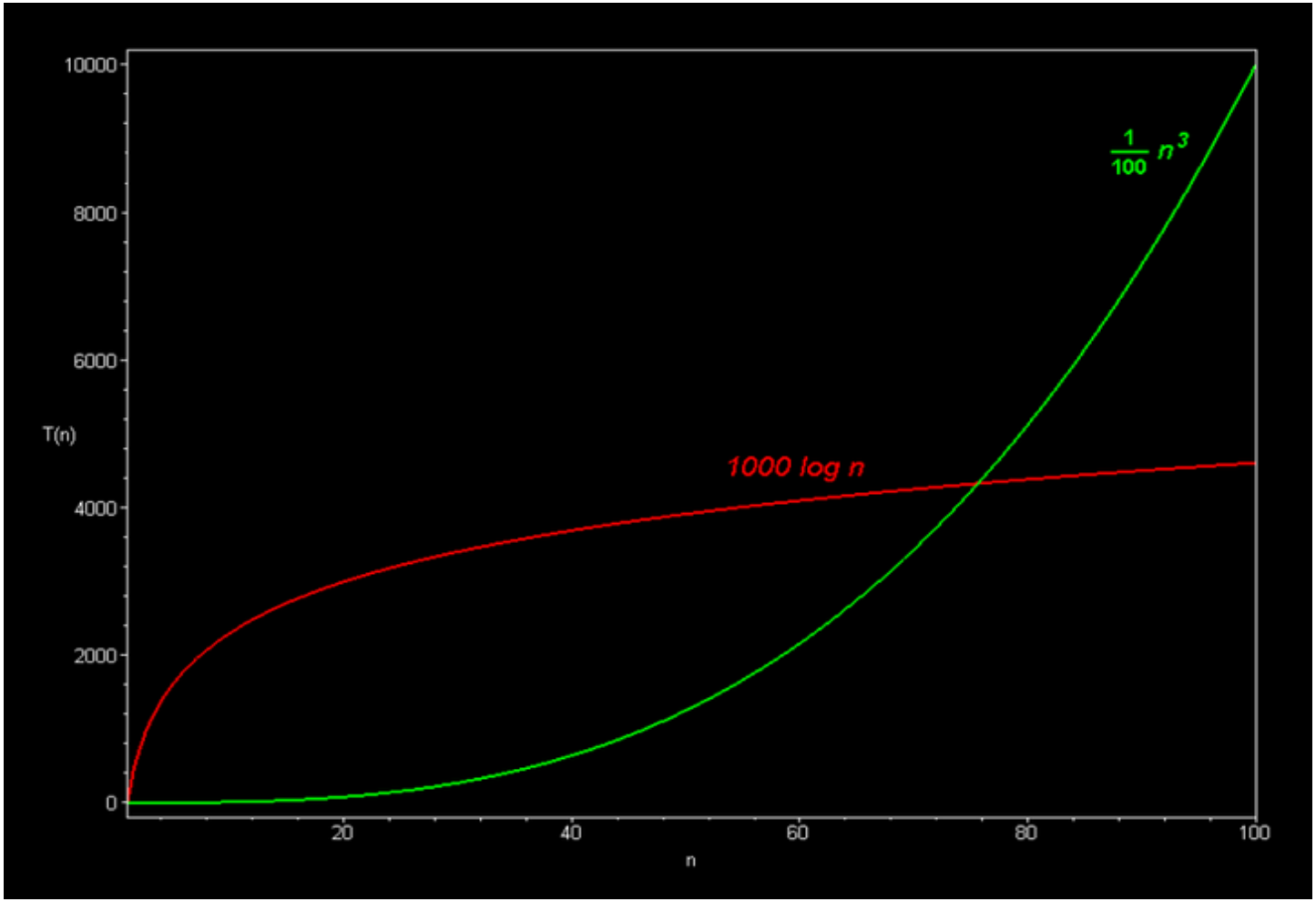
Primerjave redov velikosti



Primerjave redov velikosti



Primerjave redov velikosti



Približni algoritmi

- Približni algoritmi rešijo problem približno
  - ne zagotavljajo optimalne rešitve
  - ponavadi pa zagotovijo dovolj dobro rešitev (v sprejemljivem času)

- Primer: Polnjenje nahrbtnikov
  - v osnovi je to eksponentni problem
  - približni algoritem:
    - preveri vsak trenutni nahrbtnik
      - če gre nov paket v nahrbtnik, ga postavi vanj
    - če paket ne gre v noben nahrbtnik, dodaj nov nahrbtnik
  - dovolj dobra (in praktična) rešitev

Povzetek

- Algoritme moramo analizirati, jih evaluirati in primerjati med seboj
- Atributi:
  - pravilnost
  - učinkovitost
  - eleganca
  - razumljivost
- Med seboj primerljive algoritme primerjamo po časovni in prostorski učinkovitosti
- Redi kompleksnosti so funkcije velikosti vhodnih podatkov
  - $\Theta(\log n)$, $\Theta(n)$, $\Theta(n^2)$, $\Theta(2^n)$
- Rešitve za probleme z eksponentno zahtevnostjo so praktično nedosegljive