DUOMENŲ STRUKTŪROS IR ALGORITMAI

MARIUS GŽEGOŽEVSKIS

ALGORITMŲ SUDĖTINGUMO SKAIČIAVIMAS LAIKO ATŽVILGIU

Pagrindinė naudojama metrika algoritmo sudėtingumui nustatyti yra **BIG-O** (Didysis O).

Didžiojo **O** reikšmė dažniausiai yra pateikiama eliminuojant visas konstantas, kurios realiai neturi jokios įtakos algoritmo veikimui arba funkcijos reikšmės augimui pagal tam tikrą parametrą pvz: **f(n) + 100**, konstanta 100 laiko atžvilgiu skaičiuojant sudėtingumą nesudarys jokios įtakos taigi O žymens rezultatas bus **O (n),** iki tol kol reikės palyginti keletą skirtingų algoritmų turinčių tą patį sudėtingumą pagal didžiojo O reprezentaciją, tada reikia lyginti taikant kitus metodus.

Algoritmų laiko sudėtingumas

Laiko sudėtingumo skaičiavimas vertina kiek laiko reiktų tam tikrai problemai su tam tikru duomenų dydžiu spręsti efektyviausiu algoritmu.

Tarkime, kad turint n bitų duomenų kiekį, problema išsprendžiama per n^2 žingsnių; tokia problema yra n^2 sudėtingumo.

Iš tiesų, kiekvienas algoritmo įgyvendinimas spręstų problemą skirtingu žingsnių skaičiumi, todėl sąlyginis žingsnių skaičius (eilė) žymima $O(n^2)$.

Asimptotinis žymėjimas

O žymėjimas

Asimptotiškai "viršutinė" riba.

- Apibrėžtis:$f(n) = O(g(n))$ jei egzistuoja konstantos

c ir n_0 tokios kad $cg(n) \geq f(n)$ visiems $n \geq n_0$. *O* dažniausia naudojamas algoritmo blogiausiam atvejui apibūdinti.

Ω žymėjimas

Asimptotiškai "apatinė" riba.

- Apibrėžtis:$f(n) = \Omega(g(n))$ jei egzistuoja konstantos

c ir n_0 tokios kad $cg(n) \leq f(n)$ visiems $n \geq n_0$. Ω dažniausia apibūdina algoritmo geriausią atvejį arba apatinę ribą.

Asimptotinis žymėjimas

Θ žymėjimas

Asimptotiškai "ankšta" riba.

- Apibrėžtis: $f(n) = \Theta(g(n))$ jei egzistuoja konstantos c_1, c_2 ir n_0, tokios kad $c_1 g(n) \leq f(n) \leq c_2 g(n)$ visiems $n \geq n_0$. Taip pat galioja savybė, $f(n) = \Theta(g(n)) \iff f(n) = O(g(n))$ ir $f(n) = \Omega(g(n))$.

o žymėjimas

Asimptotiškai "negriežta viršutinė" riba.

- Apibrėžtis: $f(n) = o(g(n))$ jei egzistuoja konstantos

$c > 0$ ir $n_0 > 0$, tokios kad $cg(n) > f(n)$ visiems $n \geq n_0$.

Asimptotinis žymėjimas

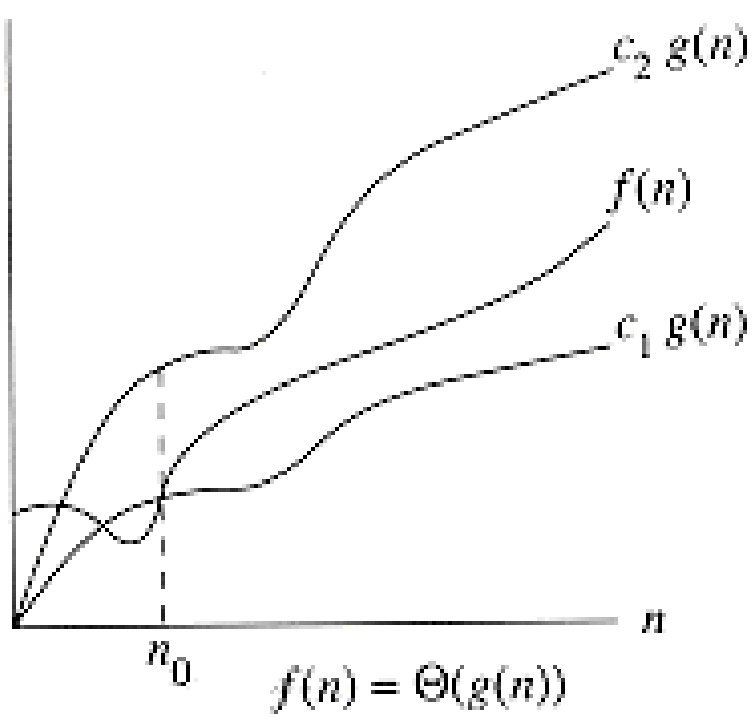
ω žymėjimas

Asimptotiškai "negriežta apatinė" riba.

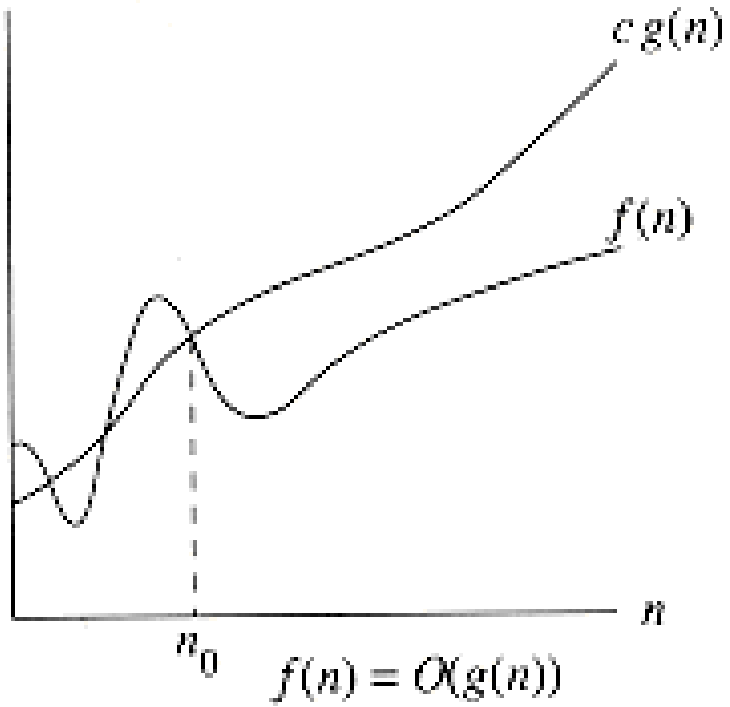
- Apibrėžtis: $f(n) = \omega(g(n))$ jei egzistuoja konstantos

$c > 0$ ir $n_0 > 0$, tokios kad $cg(n) < f(n)$ visiems $n \geq n_0$.

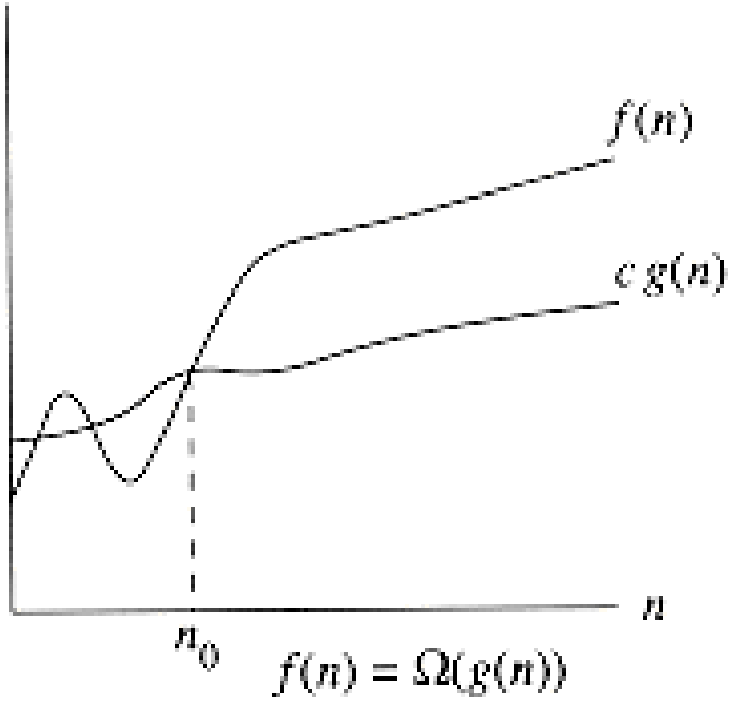
Θ - Theta, **O** - BiG O, **Ω** - Omega



(a)



(b)



(c)

O žymėjimai - dažniausi algoritmų sudėtingumo žymėjimai ir jų pavadinimai:

Žymėjimas	Sudėtingumas	Klasė
$O(1)$	konstantinis	Polinominė (P)
$O(\log n)$	logaritminis	
$O([\log n]^c)$	polilogaritminis	
$O(n)$	tiesinis	
$O(n \cdot \log n)$	supratiesinis	
$O(n^2)$	kvadratinis	
$O(n^c)$	polinominis, kartais geometrinis	
$O(c^n)$	eksponentinis	Eksponentinė (NP)
$O(n!)$	faktorialas	
$O(n^n)$	?	

Pavyzdžiai

Paprasčiausių algoritmų sudėtingumo pavyzdžiai:

- Paieškos algoritmas tiesinėje nesurūšiuotų duomenų struktūroje - $O(n)$
- Paieškos algoritmas tiesinėje surūšiuotų duomenų struktūroje - $O(\log n)$
- Rūšiavimo algoritmas tiesinėje duomenų struktūroje - $O(n \cdot \log n)$

Rūšiavimo algoritmų sudėtingumas

Dažnai greitam darbui su duomenimis būtina duomenis surūšiuoti, bet esant dideliems duomenų kiekiams labai svarbu ir pačio **rūšiavimo algoritmo sudėtingumas** - atlikimo greičio priklausomybė nuo duomenų kiekio.

Duomenų rūšiavimo uždavinys apibrėžiamas taip:

Turint N elementų seką (a_1, a_2, ..., a_N), reikia išdėstyti šiuos elementus taip, kad gautume naują N elementų seką (a_1', a_2', ..., a_N'), tenkinančią salygą $a_i <= a_j$, kai i<j.

Probleminė sritis algoritmų analizėje

Algoritmų analizėje duomenų rūšiavimo problema laikoma pačia svarbiausia, nes tai viena dažniausiai pasitaikančių operacijų programavime.

Efektyvus rūšiavimo algoritmo pasirinkimas gali turėti netgi lemiamą įtaką programos vykdymo spartai didėjant duomenų kiekiui.

Probleminė sritis algoritmų analizėje

Paprasčiausių rūšiavimo algoritmų (**išrinkimo rūšiavimo algoritmas**, **įterpimo rūšiavimo algoritmas**, **burbulo metodas**) sudėtingumas yra kvadratinis (žymima $O(N^2)$).

Dažnai greičiausiu laikomo **greitojo rūšiavimo (quicksort) algoritmo** sudėtingumas daugeliu atveju yra O(N log N), tačiau rūšuiojant beveik surūšiuotus duomenis, šio algoritmo sudėtingumas siekia $O(N^2)$.

Probleminė sritis algoritmų analizėje

Algoritmo sudėtingumas svarbus tik esant dideliam duomenų kiekiui. Palyginimui pateikiama lentelė kaip skiriasi procesoriaus operacijų skaičius didėjant duomenų kiekiui, naudojant du skirtingus algoritmus - pirmasis naudoja **5 N^2** operacijų, o antrasis **- 20 N log N**.

Duomenų elementų skaičius	Santykinis laikas burbulo algoritmu	Santykinis laikas *quicksort* algoritmu
10	500	200
100	50 000	4 000
1000	5 000 000	60 000
10000	500 000 000	800 000

Probleminė sritis algoritmų analizėje

Jei duomenų kiekis **nedidelis**, mums dažniausiai visiškai nesvarbu, **kiek mikrosekundžių bus vykdomas rikiavimas**, tačiau esant didesniems duomenų kiekiams šis skirtumas yra milžiniškas.

Kita vertus, esant labai mažiems duomenų kiekiams efektyvesnis bus burbuliuko metodas.

Taip pat algoritmų sudėtingumas priklauso nuo duomenų savybių.

Pavyzdžiui, burbuliuko metodas bus daug greitesnis, jei bus bandoma rūšiuoti surūšiuotus duomenis - tada jo sudėtingumas yra **O(n)**.

Probleminė sritis algoritmų analizėje

Dažniausiai matuojamas algoritmų sudėtingumas **vidutiniu atveju**, dažnai blogiausiu atveju ir tik kartais - geriausiu.

Tas pats algoritmas, būdamas labai **greitas geriausiu atveju**, gali būti labai blogas **vidutiniu** ar **blogiausiu atveju**.

Duomenų struktūrų operacijos

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	-	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)

MASYVO RŪŠIAVIMO ALGORITMAI

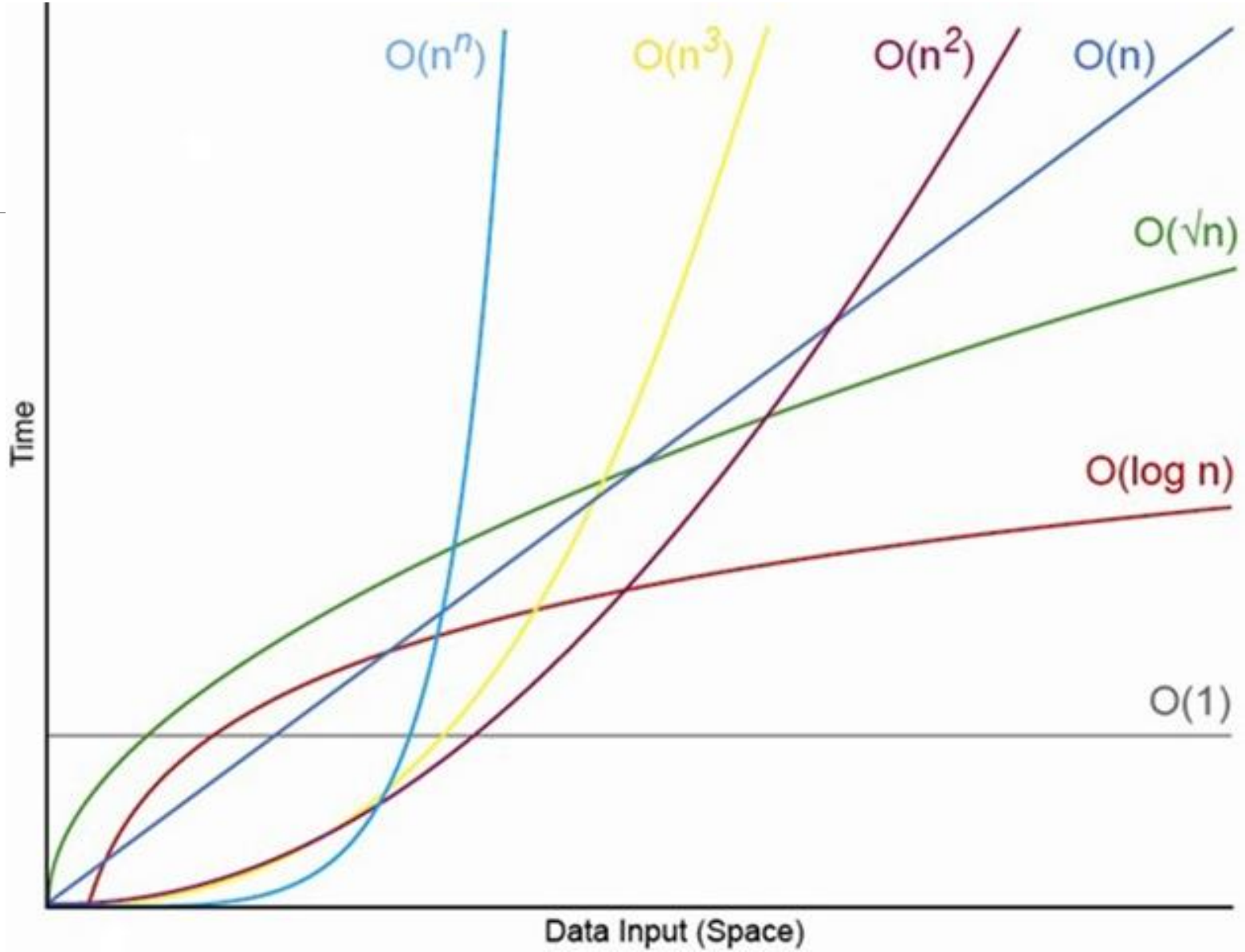
Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n log(n))	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

GRAFO OPERACIJOS

Node / Edge Management	Storage	Add Vertex	Add Edge	Remove Vertex	Remove Edge	Query
Adjacency list	O(|V|+|E|)	O(1)	O(1)	O(|V| + |E|)	O(|E|)	O(|V|)
Incidence list	O(|V|+|E|)	O(1)	O(1)	O(|E|)	O(|E|)	O(|E|)
Adjacency matrix	O(|V|^2)	O(|V|^2)	O(1)	O(|V|^2)	O(1)	O(1)
Incidence matrix	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|E|)

PIRAMIDĖS OPERACIJOS

Type	Time Complexity						
	Heapify	Find Max	Extract Max	Increase Key	Insert	Delete	Merge
Linked List (sorted)	-	O(1)	O(1)	O(n)	O(n)	O(1)	O(m+n)
Linked List (unsorted)	-	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)
Binary Heap	O(n)	O(1)	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(m+n)
Binomial Heap	-	O(1)	O(log(n))	O(log(n))	O(1)	O(log(n))	O(log(n))
Fibonacci Heap	-	O(1)	O(log(n))	O(1)	O(1)	O(log(n))	O(1)


O(n^n)
O(n^3)
O(n^2)
O(n)
O(√n)
O(log n)
O(1)
Time
Data Input (Space)

ALGORITMŲ SUDĖTINGUMO SKAIČIAVIMAS LAIKO ATŽVILGIU

Pavydžiui algoritmas yra **O (N)** ir kitas algoritmas yra toks pat **O (N)**, tada reikia detalizuoti ir gal kaikurios konstantos lemia algoritmo spartesnį, našesnį veikimą. PVZ: Turime ALGORITMĄ – **A** ir ALGORITMĄ – **B.**

A = 2n – O (n)

B = 500n – O (n)

Taigi, kuris algoritmas yra spartesnis ?

O(?)

```
for(i=0; i < N; i++)
{
    statement;
}
```

O(?)

```
for(i=0; i < N; i++)
{
   for(j=0; j < N;j++)
   {
      statement;
   }
}
```

O(?)

```
while(low <= high)
{
  mid = (low + high) / 2;
  if (target < list[mid])
    high = mid - 1;
  else if (target > list[mid])
    low = mid + 1;
  else break;
}
```

O(?)

```
void quicksort(int list[], int left, int right)
{
  int pivot = partition(list, left, right);
  quicksort(list, left, pivot - 1);
  quicksort(list, pivot + 1, right);
}
```

How to analyze Time Complexity?

Running time depends upon:

- ✗ 1) Single vs multi processor
- ✗ 2) Read/write speed to memory
- ✗ 3) 32 bit vs 64-bit
- ✓ 4) Input
  - ↳ rate of growth of time

Model Machine

- Single processor
- 32 bit
- Sequential execution
- 1 unit time for arithmetical and logical operations
- 1 unit for assignment and return

= prev

```
Sum of List ( A, n)
{
1. total = 0
2. for i = 0 to n-1
3.     total = total + Ai
4. return total
}
```

	Cost	no. of times
1	$1\ (c_1)$	1
2	$2\ (c_2)$	$n+1$
3	$2\ (c_3)$	n
4	$1\ (c_4)$	1

$$T_{sumofList} = 1 + 2(n+1) + 2n + 1$$
$$= 4n + 4$$
$$T(n) = cn + c'$$

Where, $c = c_2 + c_3$

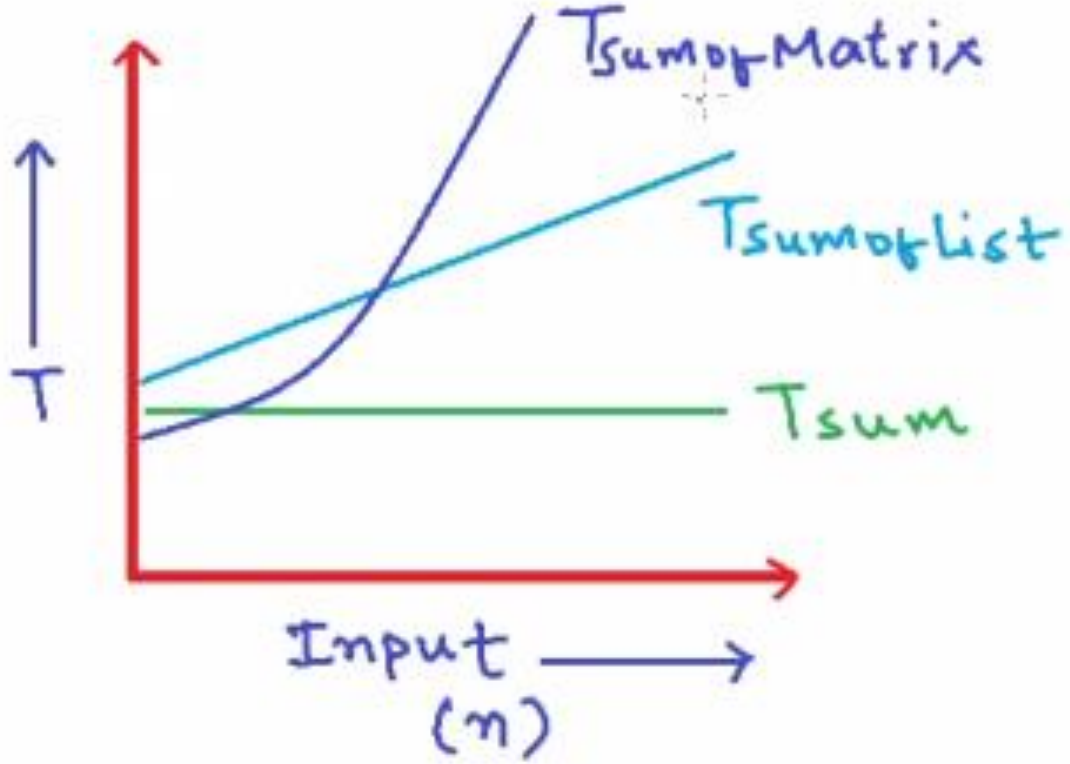
$c' = c_1 + c_2 + c_4$

prev

$T_{sum} = k$

$T_{sumoflist} = c \cdot n + c'$

$T_{sumofMatrix} = an^2 + bn + c$

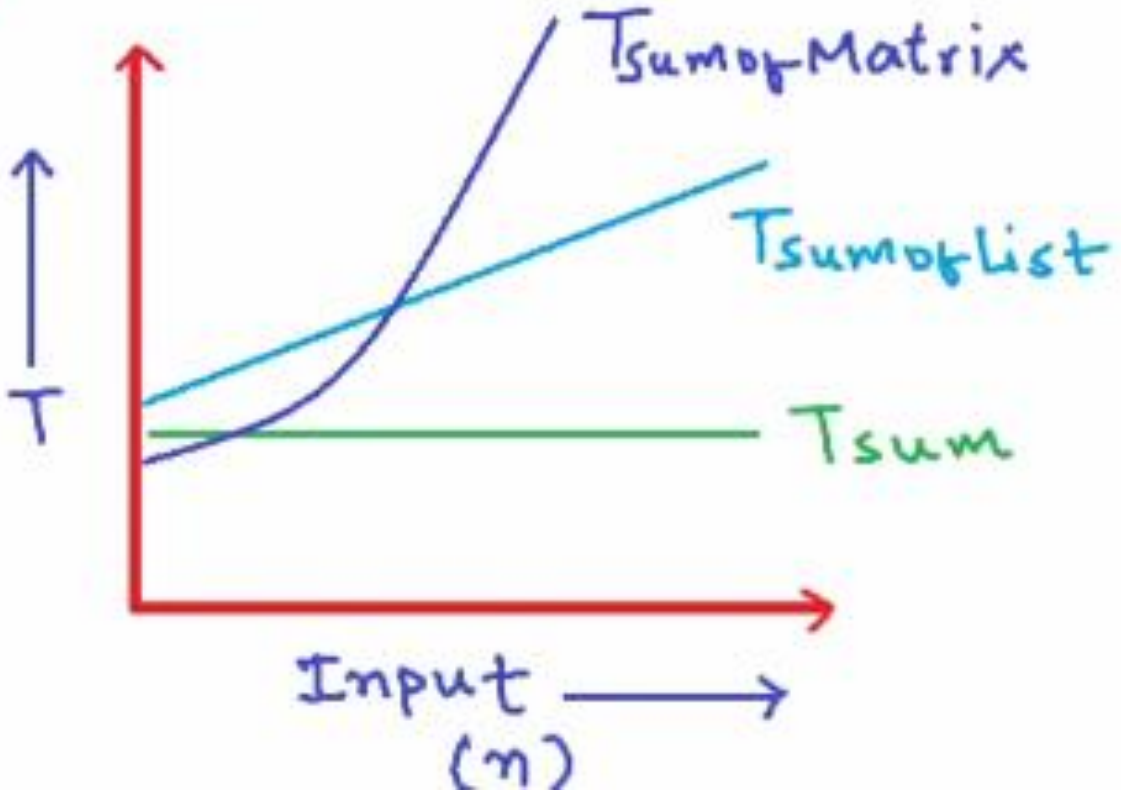


prev

$T_{sum} = k$ $O(1)$

$T_{sumoflist} = c \cdot n + c'$ $O(n)$

$T_{sumofMatrix} = an^2 + bn + c$ $O(n^2)$



Variantų perrinkimas

Spręsdami daugelį uždavinių naudojame bendrą principą – uždavinį dalijame į mažesnes užduotis, kurias išsprendę gauname viso uždavinio sprendinį.

Variantų perrinkimo principas ypač išpopuliarėjo, kai atsirado kompiuteriai. Naudodami šį metodą susiduriame su dviem svarbiausiais uždaviniais:

Variantų perrinkimas

1. Kaip padalinti uždavinį į baigtinį skaičių mažesnių užduočių (variantų).

2. Kaip sumažinti nagrinėjamų variantų skaičių, nes tiesioginis visų variantų patikrinimas gali būti neįvykdomas net su greičiausiais superkompiuteriais.

Variantų perrinkimas

Antrąją problemą sprendžiame naudodami *dinaminio programavimo*, *šakų ir rėžių* metodus, kuriuos aptarsime kituose poskyriuose.

Šiame poskyryje nagrinėsime tik vieną pavyzdį, kai svarbiausia yra sudaryti visų variantų aibę, o variantų perrinkimas yra atliekamas greitai.

Pavyzdys. Kaip sužinoti Petro vaikų amžių?

Susitiko du senokai nesimatę draugai – **Jonas** ir **Petras**.

Pokalbio metu paaiškėjo, kad ši diena yra ypatinga Petrui, nes visi **trys jo vaikai**, **Inga**, **Julija** ir **Justas**, šiandien švenčia savo gimtadienius.

Petras pasiūlė **Jonui**, geram matematikos žinovui, pabandyti atspėti, koks yra kiekvieno vaiko amžius.

Pavyzdys. Kaip sužinoti Petro vaikų amžių?

Norėdamas palengvinti užduotį jis nurodė, kad Justas yra nejaunesnis už seseris, o Inga neturi jaunesnės sesers.

Taip pat Petras pasakė, kad sudauginę visų trijų vaikų metus gauname skaičių **36**.

Šiek tiek pagalvojęs Jonas pareiškė, kad jam dar neužtenka informacijos.

Tada Petras nurodė, kad vaikų metų suma sutampa su namo, prie kurio jie stovi, langų skaičiumi.

Pavyzdys. Kaip sužinoti Petro vaikų amžių?

Jonas vėl pagalvojo ir pasakė, kad naujoji informacija tikrai labai svarbi, bet jos visgi dar nepakanka, kad galėtų pasakyti atsakymą. Todėl reikėtų mažos pagalbos.

Naujoji Petro pastaba buvo trumpa: vyriausiojo vaiko akys yra mėlynos.

Pavyzdys. Kaip sužinoti Petro vaikų amžių?

Sužinojęs tai, Jonas iš karto pasakė kiekvieno vaiko amžių! Pabandykime ir mes išspręsti šį uždavinį.

Iš pirmosios sąlygos sužinojome, kad trijų vaikų metų sandauga yra lygi **36**.

Nesunku patikrinti, kad yra tik aštuoni skirtingi variantai, kai išpildyta ši sąlyga, jie pateikti 1.1 lentelėje.

Pavyzdys. Kaip sužinoti Petro vaikų amžių?

1.1 lentelė. Aštuoni variantai, kai vaikų metų sandauga yra lygi 36

Vardas	V1	V2	V3	V4	V5	V6	V7	V8
Inga	1	1	1	1	1	2	2	3
Julija	1	2	3	4	6	2	3	3
Justas	36	18	12	9	6	9	6	4

Pavyzdys. Kaip sužinoti Petro vaikų amžių?

Kadangi turime padaryti prielaidą, kad **Jonas** žino, kiek langų turi namas, prie kurio susitiko draugai, tai iš antrosios sąlygos jis sužinojo ir kam lygi vaikų metų suma.

Tačiau tokios informacijos jam vis dar neužteko, kad galėtų pasakyti atsakymą.

Apskaičiuokime kiekvieno varianto vaikų metų sumą:

Pavyzdys. Kaip sužinoti Petro vaikų amžių?

$$1+1+36=38, \quad 1+2+18=21,$$
$$1+3+12=16, \quad 1+4+\ 9=14,$$
$$1+6+\ 6=13, \quad 2+2+\ 9=13,$$
$$2+3+\ 6=11, \quad 3+3+\ 4=10.$$

Pavyzdys. Kaip sužinoti Petro vaikų amžių?

Dabar tampa aišku, kad ši metų suma yra lygi **13**, nes visais kitais atvejais, pvz. jei vaikų metų suma būtų lygi **14** ar **21**, Jonas jau žinotų ir kiekvieno vaiko amžių.

Liko du variantai – **(1, 6, 6)** ir **(2, 2,9).**

Kadangi tik antruoju atveju Justas yra vyriausias vaikas (tai, kad

jo akys mėlynos, aišku, neturi jokios reikšmės), darome išvadą, kad

Petras **augina dvynukes Ingą ir Juliją**, kurioms sukako dveji metukai, ir devynerių metų sūnų Justą.