Algoritmi za urejanje podatkov

Primož Lah
UL PEF, DU Ma-Ra, 2. letnik
Matematične tehnologije
Mentor: dr. José Antonio Montero Aguilar

Januar 2026

PEF UNIVERZA V LJUBLJANI
Pedagoška fakulteta

Kazalo

1 Uvod

Velik del področja računalništva se ukvarja z obdelavo podatkov. Za učinkovido obdelavo podatkov pa je pogosto ključno, da so ti podatki pravilno urejeni. Tu pridejo v igro algoritmi za urejanje podatkov (*ang. sorting algorithms*). Ti algoritmi uredijo podatke v nek določeni vrstni red, da jih lahko naprej obdelujemo.

Čeprav so algoritmi za urejanje podatkov računalniški koncept, nekatere od njih pogosto uporabljamo tudi v vsakdanjem življenju. Ko na primer sortiramo paket igralnih kart, nezavedno uporabljamo nek algoritem za urejanje. Za lažjo predstavo algoritmov, ki bodo v nadaljevanju predstavljeni, si jih lahko predstavljate tudi, kot da urejamo paket kart, ali pa jih tako celo sami preizkusite.

Tekom te seminarske naloge boste spoznali kaj so algoritmi za urejanje podatkov, O-notacijo in nekaj klasičnih primerov takih algoritmov. Pri primerih algoritmov boste spoznali njihovo delovanje in učinkovitost, na koncu pa bo predstavljena še primerjava vseh navedenih algoritmov glede na njihovo učinkovitost.

2 Osnovne definicije

2.1 Algoritmi za urejanje podatkov

Kot pove že samo ime, je *algoritem za urejanje podatkov* postopek, s katerim elemente seznama uredimo po določenem vrstnem redu. Glavna prednost urejanja podatkov je, da je na urejeni množici podatkov bolj učinkovito izvajanje drugih algoritmov (npr. iskanje, grupiranje, itd.). Rezultat urejanja podatkov mora biti:

1. Urejen monotono
    - Npr. če je seznam urejen naraščajoče, mora biti vsak naslednji element večji ali enak prejšnjemu
2. Permutacija vhodnega nabora elementov
    - Izhodni nabor elementov mora vsebovati natanko vse elemente, ki so v vhodnem

Nekateri algoritmi so sicer izdelani za zaporedni dostop do podatkov, vendar jih večina predpostavlja, da so podatki shranjeni v bazi, ki omogoča naključni dostop.

Algoritme za urejanje podatkov lahko razdelimo glede na več kriterijev:

- Tip podatkov
    - Numerični
    - Leksikografski
- Lokacija hrambe podatkov med izvajanjem
    - Notranji
    - Zunanji
- Princip urejanja
    - Zamenjave
    - Vstavljanje
    - Urejanje po delih
    - itd.
- Prostorsko zahtevnost
    - Urejanje na mestu
    - Kopiranje nabora elementov
- Časovno zahtevnost
    - O-notacija

2.2 O-notacija

Učinkovitost algoritmov za urejanje podatkov najbolj pogosto merimo glede na njihovo časovno zahtevnost. *Časovna zahtevnost* je funkcija, ki nam pove, kako se čas izvajanja algoritma povečuje z večanjem nabora vhodnih podatkov. *O-notacija* je v zasnovi matematični zapis, ki opisuje približno velikost funkcije oz. domene. V računalništvu pa se ta zapis uporablja za opisovanje časovne zahtevnosti programa.[1] Z O-notacijo zapišemo, kako (tj. s kakšno funkcijo v odvisnosti od števila vhodnih podatkov) se čas izvajanja algoritma poveča, ko povečujemo velikost nabora vhodnih podatkov.

Ta zapis nam ne poda točnega časa izvajanja algoritma, namreč samo, kako se ta čas povečuje z več podatki. To je pomembna razlika, saj imata dva algoritma lahko enako časovno zahtevnost (tj. enako vrednost O), vendar je eden veliko hitrejši od drugega, ker ima na enakem naboru podatkov krajši čas izvajanja.

Za lažje razumevanje tega koncepta vzemimo za primer dva programa (A in B), ki imata oba časovno zahtevnost $O(n^2)$. Program A se pri 100 vhodnih podatkih izvede v 2 sekundah, B pa za obdelavo 100 vhodnih podatkov potrebuje 10 sekund. Oba programa imata $O(n^2)$, torej bosta za obdelavo $2n$ podatkov potrebovala $(2n)^2 = 4n^2$ časa. Torej, če program A 100 podatkov obdela v 2 sekundah, bo za 200 podatkov potreboval 16 $(4 \cdot 2^2)$ sekund. Program B, ki 100 podatkov obdela v 10 sekundah, pa bo za 200 podatkov potreboval 400 $(4 \cdot 10^2)$ sekund. Program A je torej veliko hitrejši kot B, vseeno pa imata enako časovno zahtevnost.

O-notacija je torej samo v pomoč, da lahko programer hitrejše oceni učinkovitost programa.

Seveda so programi vedno sestavljeni iz več funkcij. Pomembno se je zavedati, da je program le toliko učinkovit kot najšibkejši člen v verigi. Če ima torej program 10 funkcij s časovno zahtevnostjo $O(n)$ in eno z zahtevnostjo $O(n^2)$, potem ima celoten program časovno zahtevnost $O(n^2)$.

3 Pogosti aloritmi za urejanje podatkov

Obstaja veliko različnih algoritmov za urejanje podatkov. So različno učnkoviti in najbolj primerni za razne nabore podatkov. V nadaljevanju so predstavljeni nekateri najbolj pogosti. *Imena algoritmov bodo napisana v angleščini, saj to tudi bralcu olajša iskanje novih informacij, ker je angleških virov na to temo veliko več.*

3.1 Bubble sort

Bubble sort (oz. mehurčno urejanje) uredi podatke po velikosti, tako da primerja sosednje pare elementov med seboj.

[1] O-notacija se včasih uporabi tudi za zapis prostorske zahtevnosti. V tej seminarski nalogi bo uporabljena samo za zapis časovne zahtevnosti, ki je veliko bolj pogost.

Bubble sort najprej primerja prva dva elementa (*elementa 0 in 1*) v vhodnem seznamu. Če je prvi večji od drugega, jih zamenja, v nasprotnem primeru pa jih pusti v enakem vrstnem redu. Nadaljuje na naslednji par elementov (*elementa 1 in 2*). Ponovno ju primerja in zamenja vrstni red, če je prvi večji od drugega. Tako nadaljuje, dokler ne doseže konca seznama. S tem je algoritem premaknil največji element do konca seznama. Ponovno začne na začetku seznama in ponovi enak postopek. Ko algoritem pride skozi celoten seznam, brez da bi kakšen par elementov zamenjal, je seznam urejen.

Bubble sort algoritem je zelo enostaven za razumevanje in implementiranje, vendar je tudi zelo neučinkovit. Če ima vhodni seznam 10 elementov, bo v najslabšem primeru (če je seznam ravno nasprotno urejen) 10-krat opraviti 10 operacij. Torej ima bubble sort algoritem v povprečju časovno zahtevnost $O(n^2)$, kar je ena izmed slabših pogostih časovnih zahtevnosti. Tudi v najboljšem primeru bo bubble sort moral izvesti n operacij, torej ima v najboljšem možnem primeru časovno zahtevnost $O(n)$.

3.2 Insertion sort

Insertion sort (oz. urejanje z navadnim vstavljanjem) ureja podatke na podoben način, kot ga običajno uporabljamo ljudje. Deluje podobno, kot nekdo, ki ureja igralne karte v roki.

Insertion sort najprej pogleda element *1* in ga primerja z elementom *1*. Če sta v napačnem vrstnem redu, jih zamenja. Zatem pogleda element *2* in ga primerja z elementom *1*. Če sta v napačnem vrstnem redu, jih zamenja in se nato vrne nazaj na par elementov *0* in *1*, na katerih ponovno ponovi enak korak. Ta proces nadaljuje dokler ne pride do konca seznama. Tako ko naleti na element, ki ni na pravem mestu, ta element premakne proti začetku seznama, dokler ni *vstavljen* med manjši element pred njim in večji element za njim.

Insertion sort algoritem je v večini primerov hitrejši od algoritma bubble sort, vendar pa ima prav tako časovno zahtevnost $O(n^2)$.

3.3 Quick sort

Quick sort (oz. hitro urejanje) v delovanju spominja na proces bisekcije.

Quick sort si izbere enega od elementov v seznamu in ga določi za os. Vse manjše elemente prestavi pred os, vse večje pa za njo. Zatem si znotraj manjše strani izbere novo os in ponovno ostale elemente postavi na eno ali drugo stran osi, glede na velikost. Enako ponovi tudi na drugi strani prvotne osi. Enak postopek ponavlja naprej, dokler ni seznam urejen.

Quick sort algoritem ima v najslabšem primeru prav tako časovno zahtevnost $O(n^2)$. Vendar pa je njegova časovna zahtevnost takšna res le v najslabšem primeru, torej ko je začetni seznam nasprotno urejen. Povprečna časovna zahtevnost quick sort algoritma pa je $O(n \log(n))$.

3.4 Selection sort

Selection sort (oz. urejanje z navadnim izbiranjem) ureja seznam s postopnim premikanjem najmanjšega od preostalih elementov na začetek.

Selection sort pogleda prvi element v seznamu. Če je ta najmanjši, nadaljuje v naslednji korak. V nasprotnem primeru pa med preostalimi elementi najde najmanjšega in ga zamenja s prvim. Nato nadaljuje na drugi element seznama in ga na enak način zamenja z najmanjšim od preostalih. Ta postopek ponavlja do zadnjega elementa seznama.

Čeprav selection sort algoritem na prvi pogled izgleda hitrejši od ostalih, ima tudi ta časovno zahtevnost $O(n^2)$. Še več, za razliko od prejšnjih algoritmov selection sort v nobenem primeru nima boljše časovne zahtevnosti, vedno je njegova časovna zahtevnost $O(n^2)$.

3.5 Counting sort

Vsi algoritmi, ki smo jih pogledali do sedaj, so *primerjalni algoritmi*, torej tekom svojega delovanja primerjajo dva elementa med seboj. Counting sort (oz. urejanje s štetjem) pa je *neprimerjalen algoritem*, torej na nobeni točki tekom delovanja ne bo primerjal dveh elementov vhodnega seznama.

Counting sort najprej poišče največji element seznama (*označimo* k) in si ustvari nov začasni seznam dolžine $k + 1$, kjer imajo vsi elementi vrednost 0.

$$\begin{aligned} \text{Začetni seznam: } & [2, 5, 3, 0, 2, 3, 0, 3] \\ \text{Začasni seznam: } & [0, 0, 0, 0, 0, 0] \end{aligned}$$

Potem pogleda prvi element seznama, recimo p. V začasnem seznamu vrednost na indeksu p poveča za 1.[a]

$$\begin{aligned} \text{Začetni seznam: } & [2, 5, 3, 0, 2, 3, 0, 3] \\ \text{Začasni seznam: } & [0, 0, 1, 0, 0, 0] \end{aligned}$$

Nadaljuje na drugi element vhodnega seznama in ponovi enak korak.

$$\begin{aligned} \text{Začetni seznam: } & [2, 5, 3, 0, 2, 3, 0, 3] \\ \text{Začasni seznam: } & [0, 0, 1, 0, 0, 1] \end{aligned}$$

Tako nadaljuje do konca seznama. Na tej točki ima začasni seznam na vsakem indeksu zapisano število pojavitev tega indeksa.

$$\begin{aligned} \text{Začetni seznam: } & [2, 5, 3, 0, 2, 3, 0, 3] \\ \text{Začasni seznam: } & [2, 0, 2, 3, 0, 1] \end{aligned}$$

Sedaj pa se algoritem le še sprehodi skozi začasni seznam in pri vsakemu

elementu x, ki ima indeks i v odgovor zapiše x-krat vrednost i.[b]

Začetni seznam: $[2, 5, 3, 0, 2, 3, 0, 3]$
Začasni seznam: $[2, 0, 2, 3, 0, 1]$
Odgovor: $[0, 0, 2, 2, 3, 3, 3, 5]$

[a] Indeksiranje seznamov v računalniku se začne z 0, zato je v tem primeru pri $p = 2$ povečan tretji element začasnega seznama, ki ima indeks 2.

[b] Torej prvi element začasnega seznama ima indeks 0 in vrednost 2, zato v odgovor zapiše 2-krat število 0. Drugi element začasnega seznama ima indeks 1 in vrednost 0, zato v odgovoru ni nobenega števila 1 in nadaljuje na naslednji element začasnega seznama.

Counting sort algoritem ima eno najboljših časovnih zahtevnosti, in sicer $O(n + k)$, kjer je k vrednost največjega elementa vhodnega seznama. Za prej omenjene *primerjalne algoritme* je matematično dokazano, da je najboljša povprečna časovna zahtevnost, ki jo lahko dosežejo $O(n \log(n))$, ker pa je counting sort *neprimerjalni algoritem* lahko doseže linearno časovno kompleksnost, ki pa je skoraj najnižja možna (boljši sta le $O(c)$ in $O(\log(n))$).

3.6 Bogosort

Na koncu pa predstavimo še zabaven primer računalniškega principa „*poskusi in preveri.*“ Bogosort (oz. naključno urejanje) je zelo preprost, a hkrati izjemno počasen algoritem za urejanje podatkov.

Bogosort najprej preveri, če je seznam urejen. Če je, je algoritem zaključen. Če ni, elementom določi naključni vrstni red (seznam *randomizira*). To ponavlja, dokler seznam ni urejen.

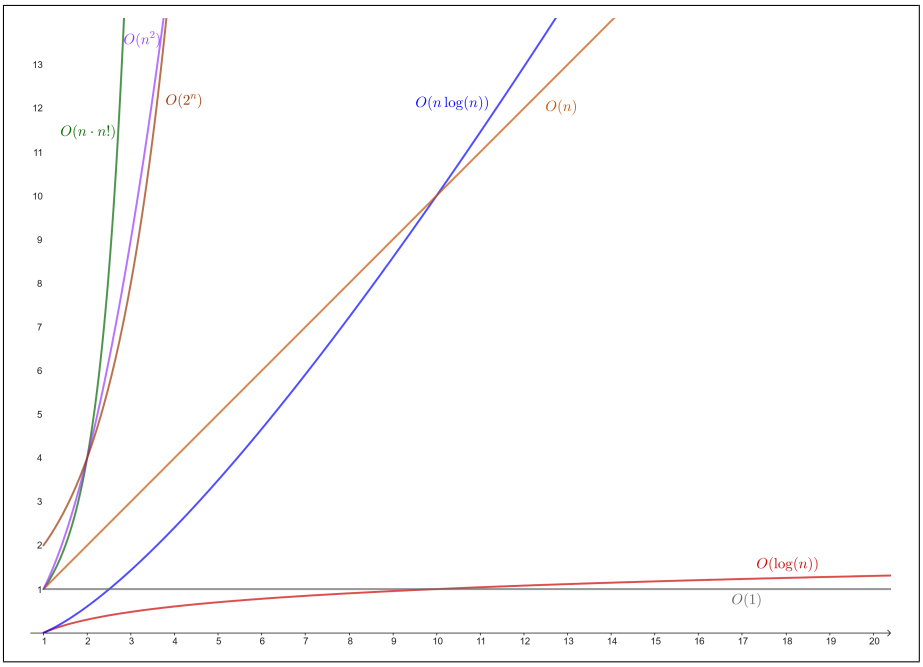
Bogosort algoritem je najslabši od vseh algoritmov, predstavljenih v tej nalogi. Njegova povprečna časovna zahtevnost je $O(n \cdot n!)$.

4 Primerjava časovnih zahtevnosti

Poleg časovnih zahtevnosti algoritmov, ki so bili predstavljeni v nalogi, poznamo tudi nekaj drugih. Najprej lahko naredimo pregled predstavljenih algoritmov in njihovih časovnih zahtevnosti, nato pa še grafični prikaz O-funkcij za lažjo primerjavo učinkovitosti programov:

Algoritem	Povprečna časovna zahtevnost
Bubble sort Insertion sort Selection sort	$O(n^2)$
Quick sort	$n \log(n)$
Counting sort	$O(n + k)$
Bogosort	$O(n \cdot n!)$

Tabela 1: Časovne zahtevnosti predstavljenih algoritmov



Slika 1: Grafi funkcij časovne zahtevnosti

Viri

[1] Brilliant.org. *Counting Sort*. [Na spletu; pridobljeno januar-2026]. 2026. URL: https://brilliant.org/wiki/counting-sort/.

[2] Brilliant.org. *Sorting Algorithms*. [Na spletu; pridobljeno januar-2026]. 2026. URL: https://brilliant.org/wiki/sorting-algorithms/.

[3] Geeks for Geeks. *Sorting Algorithms*. [Na spletu; pridobljeno januar-2026]. 2025. URL: https://www.geeksforgeeks.org/dsa/sorting-algorithms/.

[4] John Long. *Understanding Sorting Algorithms*. [Na spletu; pridobljeno januar-2026]. Dec. 2017. URL: https://medium.com/jl-codes/understanding-sorting-algorithms-af6222995c8.

[5] Tom Scott. *Why My Teenage Code Was Terrible: Sorting Algorithms and Big O Notation*. Jan. 2020. URL: https://www.youtube.com/watch?v=RGuJga2Gl_k.

[6] VisualGO. *Sorting*. [Na spletu; pridobljeno januar-2026]. 2015. URL: https://visualgo.net/en/sorting.

[7] Wikipedia contributors. *Big O notation — Wikipedia, The Free Encyclopedia*. [Na spletu; pridobljeno januar-2026]. 2025. URL: https://en.wikipedia.org/w/index.php?title=Big_O_notation&oldid=1330480633.

[8] Wikipedia contributors. *Sorting algorithm — Wikipedia, The Free Encyclopedia*. [Na spletu; pridobljeno januar-2026]. 2026. URL: https://en.wikipedia.org/w/index.php?title=Sorting_algorithm&oldid=1331211744.

[9] Wikipedija. *Algoritmi za urejanje podatkov — Wikipedija, prosta enciklopedija*. [Na spletu; pridobljeno januar-2026]. 2016. URL: https://sl.wikipedia.org/w/index.php?title=Algoritmi_za_urejanje_podatkov&oldid=4610223.

[10] Wikipedija. *Časovna zahtevnost — Wikipedija, prosta enciklopedija*. [Na spletu; pridobljeno januar-2026]. 2022. URL: https://sl.wikipedia.org/w/index.php?title=%C4%8Casovna_zahtevnost&oldid=5794524.