Primer: **najveći zajednički delilac**
Problem izračunavanja najvećeg zajedničkog delioca dva pozitivna cela broja **x** i **y**. Najveći zajednički delilac za par pozitivnih celih brojeva **x** i **y** označava se sa **nzd(x, y)** i predstavlja najveći ceo broj koji bez ostatka deli oba broja **x** i **y**. To jest, najveći zajednički delilac je broj **d = nzd(x, y)** koji deli i **x** i **y** bez ostatka, pa **x = md** i **y = nd** za neke brojeve m, n ∈ N ,
1 i svaki drugi takav zajednički delilac brojeva x i y je manji od d.
Da li je ovo dobra definicija? Drugim rečima, da li svaki par pozitivnih celih brojeva ima najveći zajednički delilac? Jasno je da svaki par pozitivnih celih brojeva ima broj 1 za zajedničkog delioca, a najveći broj koji istovremeno može da deli **x** i **y** bez ostataka je onaj broj koji je minimum od **x** i **y**. Dakle, **nzd(x, y)** uvek postoji i leži negde u intervalu između **1** i **min{ x, y }**.
Ovo zapažanje možemo iskoristiti za postupak kojim se traži **nzd(x, y)** tako što se redom proveravaju svi potencijalni kandidati za najveći zajednički delilac brojeva **x** i **y**. Ovi kandidati su uzastopni celi brojevi od broja jedan do minimuma brojeva **x** i **y**. Pri tome se ti brojevi proveravaju obrnutim redom da li je neki od njih zajednički delilac za **x** i **y**, sve dok se ne otkrije prvi zajednički delilac. Pošto se provera obavlja obrnutim redom od **1**
( sa N označavamo skup {0,1,2,... } nenegativnih celih brojeva.)
Dizajn i analiza algoritama najveće do najmanje mogućnosti, prvi nađeni zajednički delilac biće ujedno i najveći.

```
// Ulaz: pozitivni celi brojevi x i y
// Izlaz: nzd(x, y)
    algorithm gcd(x, y)
        d = min{x,y};
        while ((x % d != 0) || (y % d != 0)) do
                d = d - 1;
        return d;
```

Primetimo da se izlaznim uslovom te petlje proverava da li **d** nije delilac broja **x** ili broja **y** bez ostatka. Ako je to slučaj, telo petlje se ponavlja; u suprotnom slučaju, petlja se završava. Da bismo videli zašto će izlazni uslov naposletku postati netačan, primetimo da je početno **d = min{ x, y }**, a da se **d** u svakoj iteraciji smanjuje za **1**. Ovim uzastopnim smanjivanjem će **d** u najgorem slučaju postati jednako **1**, izlazni uslov će onda biti netačan pošto je istovremeno **x % 1 = 0** i **y % 1 = 0**, pa će se petlja tada završiti.

**Vreme izvršavanja:**
Očigledno je vreme izvršavanja ovog algoritma proporcionalno broju iteracija **while** petlje, jer se ostale naredbe izvršavaju za konstantno vreme.
Taj broj iteracija sa svoje strane zavisi od ulaznih brojeva **x** i **y**. Najbolji slučaj je kada su ulazni brojevi jednaki, **x = y**, jer je onda broj iteracija jednak **0**. U slučaju kada su ulazni brojevi različiti, broj iteracija je **min{ x, y } − 1**, pa je najgori slučaj kada se ulazni brojevi razlikuju za jedan.
Pošto je vreme izvršavanja algoritma **gcd** linearna funkcija od manjeg od dva ulazna broja, da li to znači da je on brz algoritam? Odgovor je **negativan**.

Ključno pitanje za analizu svakog aritmetičkog algoritma je: šta je zapravo veličina ulaza za taj algoritam? Na primer, ako algoritam **gcd** primenimo na dva broja **x** i **y** od, recimo, **100** cifara, onda ne možemo realistično očekivati da se aritmetičke operacije koje učestvuju u izračunavanju **nzd(x,y)** mogu obaviti za jednu jedinicu vremena. Stoga u aritmetičkim algoritmima moramo sami realizovati celobrojnu aritmetiku „veće tačnosti”, odnosno koristiti posebne algoritme za rad sa velikim brojevima.

U konkretnom slučaju algoritma **gcd** radi se o algoritmima za operaciju dodele, računanje po modulu i smanjivanje vrednosti promenljive za jedan.
Ti algoritmi na ulazu dobijaju (velike) brojeve u obliku niza cifara i njihova vremenska složenost nije jedinična, nego zavisi od broja ovih cifara. Zbog toga se i vreme ovih algoritama, koje može biti poprilično, mora uzeti u obzir prilikom analize aritmetičkh algoritama.
Iz ove diskusije se može naslutiti da je prava mera veličine ulaza aritmetičkh algoritama jednaka zbiru broja cifara (tačnije, broja bitova u binarnom zapisu) njihovih ulaznih brojeva. Naime, sâm broj ulaznih brojeva nije dobra mera, jer je konstantan (na primer, algoritam **gcd** uvek ima dva ulazna broja). Ali broj bitova u binarnom zapisu ulaznih brojeva raste sa magnitudom ulaznih brojeva i time se na pravi način obuhvata vremenska složenost aritmetičkih operacija u zavisnosti od veličine brojeva koji u njima učestvuju.
Primenimo ovaj pristup u analizi algoritma **gcd** . Neka je **n1** broj bitova binarnog zapisa ulaznog broja **x** i **n2** broj bitova binarnog zapisa ulaznog broja **y.** Mera veličine ulaza tog algoritma **n** je dakle zbir brojeva bitova binarnog zapisa ulaznih brojeva: **n = n1 + n2** . Sada dakle treba, zavisno od ove prave mere veličine ulaza za algoritam **gcd** , oceniti njegovo vreme izvršavanja u najgorem slučaju.
U takvoj analizi algoritma **gcd** možemo čak zanemariti činjenicu da se osnovne aritmetičke operacije ne izvršavaju za jedinično vreme, jer to vreme samo doprinosi ukupnom vremenu izvršavanja, koje je u ovom slučaju ionako veliko i iznosi bar $\mathbf{2^{n/2}}$ . Da bismo ovo pokazali, primetimo najpre da je broj bitova binarnog zapisa svakog pozitivnog celog broja a jednak |log a|+1. 3 To znači da je n1 ≈ log x i n2 ≈ log y, odnosno x ≈ $2^{n1}$ i y ≈ $2^{n2}$ .
U najgorem slučaju algoritma **gcd** kada su ulazni brojevi susedni, recimo **x = y − 1**, broj njihovih bitova je otprilike podjednak, odnosno n 1 ≈ n 2 ≈ n/2. Kako je vreme izvršavanja tog algoritma T(n) proporcionalno vrednosti manjeg od dva ulazna broja, to je:
T(n) = Ω (y) = Ω( $2^{n2}$ )= Ω( $2^{n/2}$ ).
To pokazuje da se algoritam **gcd** izvršava za ekponencijalno vreme, što je beskorisno za velike brojeve od 100–200 cifara.

# Euklidov algoritam

**TEOREMA (EUKLID )**

**Ako su x i y pozitivni celi brojevi takvi da x ≥ y, onda je**
**nzd(x, y) = nzd(y, x%y).**

Uz primedbu da svaki pozitivan ceo broj x deli broj 0 bez ostatka i da zato nzd(x, 0) = x, najveći zajednički delilac dva broja možemo lako izračunati uzastopnim ponavljanjem jednakosti iz Euklidove teoreme. Neki primeri tog izračunavanja za razne parove brojeva su:

- nzd(1055, 15) = nzd(15, 5) = nzd(5, 0) = 5
- nzd(400, 24) = nzd(24, 16) = nzd(16, 8) = nzd(8, 0) = 8
- nzd(101, 100) = nzd(100, 1) = nzd(1, 0) = 1
- nzd(34, 17) = nzd(17, 0) = 17
- nzd(89, 55) = nzd(55, 34) = nzd(34, 21) = nzd(21, 13) =
  = nzd(13, 8) = nzd(8, 5) = nzd(5, 3) =
  = nzd(3, 2) = nzd(2, 1) = nzd(1, 0) = 1

Na osnovu ovih primera se može otkriti Euklidov postupak kojim se dobija najveći zajednički delilac dva broja: dati par brojeva se transformiše u niz parova, pri čemu je prvi broj narednog para jednak drugom broju prethodnog para, a drugi broj narednog para jednak je ostatku celobrojnog

deljenja prvog broja sa drugim brojem prethodnog para. Poslednji par u nizu je onaj kod koga je drugi broj jednak nuli, a onda je najveći zajednički delilac početnog para jednak prvom broju tog poslednjeg para.

Precizniji opis Euklidovog algoritma na pseudo jeziku je:

```
// Ulaz: celi brojevi x i y takvi da x ≥ y > 0
// Izlaz: nzd(x, y)
  algorithm euclid(x, y)
        while (y > 0) do
                z = x % y;
                x = y;
                y = z;
        return x;
```

Ako *a* i *b* nisu oba jednaka nuli, najveći zajednički delilac *a* i *b* se može izračunati korišćenjem najmanjeg zajedničkog sadržaoca (nzs) brojeva *a* i *b*:

$$\mathrm{nzd}(a,b) = \frac{a \cdot b}{\mathrm{nzs}(a,b)}.$$

ili

**nzd(*a*, *b*) · nzs(*a*, *b*) = *a* · *b*.**

Ova formula se često koristi za računanje najmanjeg zajedničkog sadržaoca: prvo se NZD izračuna pomoću Euklidovog algoritma, a zatim se proizvod dva broja podeli njihovim NZD.

nzd (12345, 153) .... 12345mod153=105
nzd (153, 105) .....  153mod105= 48
nzd(105, 48) ... 105mod48 = 9
nzd(48,9) ...  48mod9=3
nzd(9,3)  .....  9mod3=0
nzd(3,0)=3