$$\sum_{1=1}^{n} ca_i = c\sum_{i=1}^{n} a_i \qquad \sum_{i=1}^{n}(a_i + b_i) = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$$

ARITMETIČKI RED:

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = \Theta(n^2)$$

GEOMETRIJSKI RED: za $0<x<1$ $\Theta(1)$

$$\sum_{i=0}^{n} X^i = 1 + X + X^2 + \cdots + X^n = \frac{X^{n+1}-1}{X-1} = \Theta(X^n)$$

HARMONIJSKI RED: $n\geq 0$

$$\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln n = \Theta(\ln n)$$

KVADRATNI RED: $n\geq 0$

$$\sum_{i=1}^{n} i^2 = 1 + 4 + 9 + \cdots + n^2 = \frac{2n^3 + 3n^2 + n}{6} = \Theta(n^3)$$

ANALIZIRANJE PETLJI se vrši iznutra prema vani. Evo primjera jednog slozenijeg zadatka s vjezbi:

for j=3 to n do
for i=1 to j-3 do
for k=i to j/3 do
output ()

Prvu petlju oznacimo sa M(i) i tada vrijedi:

$\sum_{k=i}^{j/3} 1 = \frac{j}{3} - i + 1$, vrijedi kod konstanti „(gornja granica – donja granica + 1)* konstanta“.

Nakon toga rijesavamo sljededecu petlju K(i):

$\sum_{k=i}^{j/3} M(i) = \sum_{k=i}^{j/3} \left(\frac{j}{3} - i + 1\right)$, granice nisu onakve kakve pisu u samoj petlji jer treba voditi računa o tome da se u nekim slucajevima petlja nece izvrtiti. I na ktaju se analogno ovome izrazi zadnja petlja vodeći računa o ograničenjima.

UPOTREBA GRUBE GRANICE:

$$\sum_{i=1}^{n} i^2 \leq \sum_{i=1}^{n} n^2 = n^3$$

$$\textit{svaki izraz u sume se može zamjeniti s najvećim}$$

APROKSIMIRANJE KORIŠTENJEM INTEGRALA:

$$\int_0^n f(x)dx \leq \sum_{i=1}^{n} f(i) \leq \int_i^{n+1} f(x)dx$$

$$\sum_{i=1}^{n} i^2 \leq \int_1^{n+1} x^2 dx = \frac{x^3}{3}\bigg|\begin{matrix} n+1 \\ 1 \end{matrix} = \frac{(n+1)^3}{3} - \frac{1}{3} = \cdots$$

UPOTREBA INDUKCIJE: Neka je n cjelobrojna varijabla na kojoj se vrši indukcija. Teorem ili formula koju treba dokazati, akoju zovemo hipoteza, je funkcija od n. Hipotezu označimo sa HI(n). Dokaz indukcijom sastoji se od dva koraka. U prevom koraku pokažemo da postoji neki dovoljno mali n_0 za koji je hipoteza točna tj. Da je točno $HI(n_0)$. Ako je taj uvjet ispunjen prelazimo na sljedeći korak. On kaže da moramo pokazati d aje hipoteza točna za sve n-ove veće od n_0. Ptrpostavimo da je hipoteza točna za prvi manji od n, dakle HI(n-1) je točno i onda dokažemo da je i n točno.

ASIMPTOTSKO OZNAČAVANJE:

Θ – označavanje:

$\Theta(g(n)) = \{f(n)$: postoje pozitivne konstante c_1, c_2 i n_0 takve da je $0\leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, za sve $n\geq n_0\}$

O – označavanje:

$O(g(n)) = \{f(n)$: postoje pozitivne konstante c i n_0 takve da je $0\leq f(n) \leq cg(n)$, za sve $n\geq n_0\}$

Ω – označavanje:

$\Omega(g(n)) = \{f(n)$: postoje pozitivne konstante c i n_0 takve da je $0\leq cg(n) \leq f(n)$, za sve $n\geq n_0\}$

o-označavanje i ω-označavanje koriste se kod granica koje nisu asimptotski čvrste: $2n^2 = O(n^2)$-čvrsta, $2n = O(n^2)$-nije čvrsta -> $2n = o(n^2)$, $2n^2 \neq o(n^2)$ ili $2n^2 = \omega(n)$, $2n^2 \neq \omega(n^2)$.

OGRANIČENO PRAVILO:

Θ – označavanje:

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, gdje mora vrijediti c>0 ali ne beskonačan

O – označavanje:

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, gdje mora vrijediti c≥0 ali ne beskonačan

Ω – označavanje:

$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$, gdje je rezultat striktno pozitivan ili beskonačan

---

REKURZIJA

METODA TRAŽENJA UZORKA:

Neka imamo formulu

$$T(n) = \begin{cases} 1, n = 1 \\ T(n/2) + T(n/2) + n, za\ ostale \end{cases}$$

Razvijmo sada gornju rekurziju:

T(1) = 1, n=1
T(2) = T(1)+T(1)+2 = 1+1+2 = 4, n=2
T(3) = T(2)+T(1)+3 = 4+1+3 = 8, n=3
T(4) = T(2)+T(2)+4 = 4+4+4 = 12, n=4
T(5) = T(3)+T(2)+5 = 8+4+5 = 17, n=5
...
T(8) = T(4)+T(4)+8 = 12+12+8 = 32, n=8
...
T(16) = T(8)+T(8)+16 = 32+32+16 = 80, n=16
...
T(32) = T(16)+T(16)+32 = 80+80+32 = 192, n=32

Pravilo po kojem se ponašaju vremena trajanja, na prvi pogled nije lako odrediti. Kako rekurzijom djelimo broj podataka uvjek s dva, za očekivati je da će za taj slučaj pravilo biti lakše pronaći. Razmotrimo dakle omjer T(n)/n za potencije broja 2:

T(1)/1 = 1, T(2)/2 = 2, T(4)/4 = 3, T(8)/8 = 4, T(16)/16 = 5, T(32)/32 = 6. Ovo sugerira da bi za potenciju broja dva rješenje bilo oblika T(n) = nldn+n. Dobiveno rješenje nije dokaz, ali daje koristan početni korak.

I na kraju samo izbacimo sva nespretna zaokruživanja te dobijemo:

$$T(n) = \begin{cases} 1, n = 1 \\ 2T(n/2) + n, za\ ostale \end{cases}$$

PROVJERA POMOĆU INDUKCIJE:

<u>Tvrdnja:</u> za sve n≥1, gdje je n potencija broja 2 vrijedi T(n) = nldn+n.

<u>Dokaz:</u>

Počeni uvjet : (n=1) -> T(1) = 1*ld1+1 = 1, dakle odgovara T(1)=1.

Korak Indukcije: neka je n>1, pretpostavimo da izraz T(n') = n'ldn'+n' vrijedi za sve n'<n. Želimo dokazati da u tom slučaju formula vrijedi i za n. Da bi to dokazali izrazit ćemo T(n) preko manjih vrijednosti ta što ćemo iskoristiti definiciju T(n) = 2T(n/2)+n. Sad imamo da je n/2<n, što znači da možemo primjeniti hipotezu koja kaže da je T(n/2) = n/2*ld(n/2)+n/2. Sada taj izraz uvrstimo u početni i dokažemo da je tvrdnja točna.

METODA ITERACIJE:

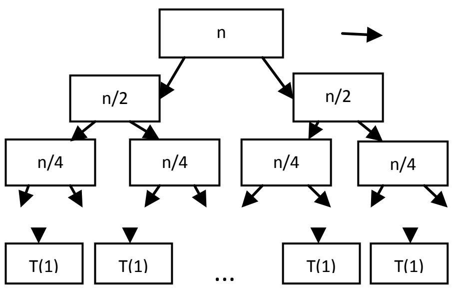
- Počinjemo tako da razvijamo definiciju sve dok ne vidimo uzorak:

T(n) = 2T(n/2)+n
= 2(2T(n/4)+n/2)+n = 4T(n/4)+2n
= 4(2T(n/8)+n/4)+2n = 8T(n/8) + 3n
....
$T(n) = 2^k T(n/2^k)+kn$

- Sada smo dobili uzorak. Slijedeće je, obzirom da vrijedi T(1) = 1, da odaberemo da $n/2^k = 1$ => k=ldn. Onda to uvrstimo u izraz dobiven uzorkom i dobijemo rješenje
- Rješenje potom preovjerimo metodom indukcije

METODA REKURZIVNOG STABLA:

Broj razina: Iz graničnog uvjeta rekurzije slijedi (npr. n=1) $n/(\text{broj podjela})^i = 1$ -> $i = \log_{\text{broj podjela}} n$. Ukupan broj podjela je $1+ \log_{\text{broj podjela}} n$ i to je ujedno i gornja granica sume.



MASTER THEOREM:

Koristi se za rekurzije oblika $T(n) = aT(n/b)+n^k$:

1. ako je $a>b^k$, tada je $T(n) = \Theta(n^{\log_b a})$
2. ako je $a=b^k$, tada je $T(n) = \Theta(n^k \mathrm{ld} n)$
3. ako je $a<b^k$, tada je $T(n) = \Theta(n^k)$

MERGE SORT

Zadan je niz A[p..r]

```
MergeSort (array A, int p, int r)
    if (p<r)  //ispunjeno ako ima barem 2 elemnenta
        q = (p+r)/2
        MergeSort(A, p, q) //sortiranje A[p..q]
        MergeSort(A, q+1, r) //sortiranje A[q+1..r]
        Merge(A, p, q, r) // spajanje
```

Funkcija *Merge* ima slijedeći pseudo kod:

```
Merge (array A, int p, int q, int r)
    array B
    i=k=p //inicijalizacija indeksa
    j=q+1
    while (1<=q and j<=r) //dok oba potpolja nisu prazna
        if (A[i]<=A[j]) B[k++]=A[i++] //kopiraj iz lijevog pot.
        else B[k++]=A[j++] // kopiraj iz desnog potpolja
    while (i<=q) B[k++]=A[i++] //kopiraj sve što je ostalo u B
    while (j<=q) B[k++]=A[j++] //kopiraj sve štp je ostalo u B
    for i=p to r do A[i]=B[i] //kopiraj B u A
```

QUICK SORT (nije stabilan, u mjestu)

Zadan je niz A[p..r]

```
QuickSort (array A, int p, int r)
    if (r<=p) return  //0 ili jedan element return
    i = slučajni indeks iz [p..r] //odabiranje slučajnog indexa
    zamjeni A[i] s A[p] //stavljanje x na odgovarajuće mjesto
    q = Partition (p, r, A) //podjela A oko elementa x
    QuickSort(p, q-1, A) //sortiranje A[a..q-1]
    QuickSort(q+1, r, A) //sortiranje A[q+1..r]
```

Funkcija *Parition* ima slijedeći pseudo kod:

```
Partition (int p, int r, array A)
    x = A[p] //x je jednak prvom elementu
    q = p
    for s=p+1 to r do
        if (A[s]<x)
            q = q+1
            swap A[q] with A[s]
    swapA[p] with A[q] //stavi x na odgovarajuće mjesto
    return q //vrati poziciju elementa x
```

HEAP SORT (nije stabilan, umjestu)

```
HeapSort (int n, array A[1..n])
    BuildHeap(n, A) //izgradi heap strukturu
    m=n //inicijalno heeap struktura sadrži sve elemente
    while (m>=2)
        swapA[1] with A[m] //izvuci m-ti najveći
        m = m-1 //izbaci A[m] (koji je već složen iz heap stukt.)
        Heapify (A, 1, m) //složi element koji nije u strukt.
```

Funkcija *Heapify* ima slijedeći pseudo kod:

```
Heapify (array A, int i, int m)
    l = Left(i) //lijevo dijete (2i)
    r = Right (i) // desno dijete (2i)
    max = i
    if (l<=m and A[l]>A[max]) max=l //lijevo djete postoji i >
    if (r<=m and A[r]>A[max]) max=r //desno djete postoji i >
    if (max != i) //ako je neko djete veće
        swap A[i] with A[max] //zamjeni element s većim djetetom
        Heapify (A, max, m) //rekurzivno nastavi
```

Može se lako pokazati da pristup elementima u heap strukturi zahtjeva jednostavne aritmetičke operacije nad elementima polja: *Left(i): vraća 2i, Right(i): vraća 2i+1, Parent: vraća* $\lfloor i/2 \rfloor$, *IsLeaf(i): vraća Left(i)>m (tj ako lijevo djete i-tog čvora nije u stablu), IsRoof(i): vraća i==1.*

COUNTING SORT (stabilan, nije u mjestu)

Ideja: Za svaki element u polju odrediti njegov položaj u konačno sortiranom polju. Algoritam je jednostavan: (trebaju nam dva pomoćna polja) 1) inicijaliziramo R sa 0. 2) Prebrojimo koliko se puta svaki ključ pojavi. 3)Izračunamo R[x] kumulativno, tj odredimo broj elemenata kojima je ključ manji ili jednak tom ključu. Kopiramo zapis prema ključu na konačno sortiramo mjesto.

```
CountingSort (int n, int k, array A, array B)
    for x=1 to k do R[x]=0 //inicijaliziranje R
    for j=1 to  do R[A[j].key]++ //R[x]dodamo 1 ako je A[j].key=x
    for x=2 to k do R[x]+=R[x-1] //R[x] postaje položaj x-a
    for j=n to 1 do //pomicanje elemenata iz A u B
        x = A[j].key
        B[R[x]] = A[j]
        R[x]-- //da bi riješili problem duplih
```

RADIX SORT: (uvjetno stabilan [zavisi o pomoćnom algoritmu], u mjestu)

Ideja: Neka se naša lista koju želimo sortirati sastoji os n brojeva od kojih svaki ima d znamenki (decimalni ili po bilo kojoj bazi). Pretpostavimo da imamo stabilan algoritam za sortiranje, kao što je CountingSort. Da bi sortirali brojeve jednostavno sortiramo od najmanje značajne znamenke prema značajnijoj. Kako je algoritam za sortiranje stabilan sortirani brojevi će zadržati pravilan raspored.

```
RadixSort (int n, int d, array A)
    for i=1 to d do
        sortiraj A //stabilno ->po i-tom značajnom bitu
```

576	49[4]	9[5]4	[1]76	176
494	19[4]	5[7]6	[1]94	194
194	95[4]	1[7]6	[2]78	278
296	57[6]	2[7]8	[2]96	296
278	29[6]	4[9]4	[4]94	494
176	17[6]	1[9]6	[5]76	576
954	27[8]	2[9]6	[9]54	954

LOGARITMI:

1. $\log_a(x\cdot y) = \log_a x + \log_a y$, **2.** $\log_a \frac{x}{y} = \log_a x - \log_a y$, **3.** $\log_a x^y = y \cdot \log_a x$, **4.** $\log_{a^n} a = \frac{1}{n}\log a$,

5. $\log_b a = \frac{1}{\log_a b}$ **6.** $\log_{a^n} x = \frac{1}{n}\log_a x$, **7.** <u>**$a^x = y$ =>**</u> $\boldsymbol{\log_a y = x}$, **8.** $\log_a 1 = 0$, **9.** $\log_a a = 1$, **10.** $a^{\log_a x} = x$, **11.** $\log_a x = \frac{\log_b x}{\log_b a}$

INTEGRALI:

$$\int x^n\, dx = \frac{x^{n+1}}{n+1} + C,\ n\neq -1$$

$$\int \ln x\, dx = x\ln x - x + C$$

$$\int \log_b x\, dx = x\log_b x - x\log_b e + C$$

$$\int \frac{dx}{x} = \ln|x| + C$$

$$\int e^x\, dx = e^x + C$$

$$\int \sin x\, dx = -\cos x + C$$

$$\int a^x\, dx = \frac{a^x}{\ln a} + C$$

$$\int \cos x\, dx = \sin x + C$$

$$\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin\frac{x}{a} + C$$

$$\int \frac{dx}{\sin^2 x} = -\mathrm{ctg}\, x + C$$

$$\int \frac{dx}{a^2 + x^2} = \frac{1}{a}\arctan\frac{x}{a} + C$$

$$\int \frac{dx}{a^2 - x^2} = \frac{1}{a}\mathrm{arth}\frac{x}{a} + C = \frac{1}{2a}\ln\left|\frac{a+x}{a-x}\right| + C$$

$$\int \frac{dx}{\cos^2 x} = \mathrm{tg}\, x + C$$

$$\int \frac{dx}{x^2 - a^2} = -\frac{1}{a}\mathrm{arcth}\frac{x}{a} + C = \frac{1}{2a}\ln\left|\frac{x-a}{x+a}\right| + C$$

$$\int \mathrm{sh}\, x\, dx = \mathrm{ch}\, x + C$$

ZADACI S VJEZBI: (čisto da se nađe)

1. zadatak koji ulazni niz raspoređuje u n parova di max suma para mora bit najmanja moguća

```
TraziParove(array A, int n)
    sortiraj A
    array B[1..n][2]
    max = 0
    for i=1 to n do
        B[i][1] = A[i]
        B[i][2] = A[2n+1-i]
        if (B[i][1] + B[i][2] > max)
            max = B[i][1] +B[i][2]
```

* Ako kod petlje imamo korak različit od 1, npr. i=i/2 tada uvodimo novu oznaku l koja za ovaj primjer iznosi $i=2^l$, a l = 0..ldn, pa su 0 i ldn granice sume koju računamo. Dakle baza logaritma je jednaka koraku.