PRIMERI

Določi časovno zahtevnost:

n je velikost problema

```
s = 0;
for (int i = 1; i <= n; i++)
   for (int j = 1; j <= n; j++)
       s = s + t[i][j];
```

$$O(n^2)$$

PRIMERI

Določi časovno zahtevnost:

```
void p(int n, int m, int k) {
   s = 0;
   for (int i = 1; i <= n; i++)
       for (int j = 1; j <= m; j += k)
           s = s + t[i][j];
}
```

n, m in k določajo velikost problema

$$O(n \cdot \frac{m}{k})$$

Primeri

Določi časovno zahtevnost:

```
int i = n;
int r = 0;
while (i > 1) {
   r = r + 1;
   i = i / 2;
}
```

$$O(\log n)$$

PRIMERI

Določi časovno zahtevnost:

kot če bi bilo
vgnezdenih m zank

```
void p(int n, int m){
   if (m > 0)
       for (int i = 1; i <= n; i++)
           p(n, m-1);
}
```

$$O(n^m)$$

Primeri

Določi časovno zahtevnost:

```
void p(int n) {
   if (n > 0)
       for (int i = 1; i <= n; i++)
           p(n-1);
   else
       for (int i = 1; i <= 10; i++)
           System.out.println(i);
}
```

$$O(n!)$$

PRIMER: IZRAČUN FAKULTETE

Iterativno:

```
fakulteta = 1;
for(int i=1; i <= n; i++)
   fakulteta = fakulteta*i;
```

Rekurzivno:

```
private int fakulteta(int n) {
if (n==0) return 1;
    else  { return n * fakulteta(n-1); }
}
```

Potenciranje števila

```
private int potenca(int x, int p) {
if (p==0)
      return 1;
     else  {
      return x * potenca(x, p-1);
     }
}
```

HANOJSKI STOLPI

```
// premik n ploščic iz palice A na palico B z uporabo
// pomožne palice C
```

```
static public void hanoi(char A, char B, char C,int n) {
    if (n>0) {
        hanoi(A,C,B,n−1) ;
        System.out.println("premik iz " + A + " na " + B);
        hanoi(C,B,A,n−1) ;
    } // if
} // hanoi
```

Primer Fibonacci

Rekurzivno

```
public static int fib(int n) {
    if (n <= 2)
        return 1;
    else
        return fib(n-1)+fib(n-2);
}
```

Iterativno

```
public static int fib(int n) {
   int n1=1, n2=1; n3;
    for(int i=2; i < n; i++) {
        n3 = n1 + n2;
        n1 = n2;
        n2 = n3;
    }
    return n2;
}
```

Primer: Permutacije

```
static public void permutationsRec(int n0) {
   if (n0==0)
      writePermutation() ;
   else {
      for (int i=0, temp ; i < n0 ; i++) {
         temp = a[i] ; a[i] = a[n0−1] ; a[n0−1] = temp ;
         permutationsRec(n0−1) ;
         temp = a[i] ; a[i] = a[n0−1] ; a[n0−1] = temp ;
      }
   }
} // permutationsRec
```

Primer 1

Za dani program so bili izmerjeni naslednji časi izvajanja za različne velikosti vhodnih podatkov:

velikost podatkov	5	6	7	8	9	10
čas	13.8	83.6	388.6	1796.2	8753.6	44421.4
c		1.7 E-06	1.5E-06	8.0E-08	1.3E-09	1.7E-11
e		9.88	9.97	11.46	13.45	15.42

- polinomsko: $T(n) = c\ n^e$

Konstante NISO konstantne...

Primer 1

Za dani program so bili izmerjeni naslednji časi izvajanja za različne velikosti vhodnih podatkov:

velikost podatkov	5	6	7	8	9	10
čas	13.8	83.6	388.6	1796.2	8753.6	44421.4
c		1.7 E-03	8.3E-03	8.6E-03	5.6E-03	3.9E-03
e		2.60	2.22	2.21	2.28	2.34

- eksponentno: $T(n) = c\ 2^{en}$

Konstante SO vsaj do neke mere konstantne...

RAST FUNKCIJ ZAHTEVNOSTI

f(n)	f(n+1) - f(n)	=	O(g(n))	povečanje velikosti rešljivega problema v danem času z 10× hitrejšim rač.
$\log n$	$\log(n+1) - \log n$	=	$O(\frac{1}{n})$	n^{10}
n	1	=	$O(1)$	10 n
$n \log n$	$(n+1)\log(n+1) - n\log n$	=	$O(\log n)$	$< 10n$
n^2	$2n+1$	=	$O(n)$	$3.16n$
n^3	$3n^2+3n+1$	=	$O(n^2)$	$2.15n$
n^4	$4n^3+6n^2+4n+1$	=	$O(n^3)$	$1.78n$
2^n	2^n	=	$O(2^n)$	$\leq n+4$
$n!$	$n \times n!$	=	$O((n+1)!)$	$\leq n+1$