Primer – binarna pretraga

- Jedinica mere – broj prolazaka kroz petlju

```
BinSearch(a, n, key)
   lo = 1
   hi = n
   while (lo <= hi)
      mid = (lo + hi) / 2
      if (key < A(mid)) hi = mid - 1
      else if (key > A(mid)) lo = mid + 1
      else return mid // Trazeni element niza je sa indeksom mid
   return 0
```

$$F(n) = \log_2 n = O(\log n)$$

Druge notacije

- **Definicija.** $f(n) = \Omega\big(g(n)\big)$ kada $n \to \infty$, ako postoje c i n_0 takvi da je $f(n) \geq c \cdot g(n)$ za svako $n \geq n_0$.
- **Definicija.** $f(n) = \theta(g(n))$ kada $n \to \infty$, ako postoje c_1, c_2 i n_0 takvi da je $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ za svako $n \geq n_0$.
- $\Omega\big(g(n)\big)$ - donja granica za $f(n)$
- $\theta(g(n))$ – tesna granica za $f(n)$
- Ako je $f(n) = \theta(g(n))$, onda važi da je $f(n) = O(g(n))$ i $f(n) = \Omega(g(n))$

Druge notacije

- $n^2, 3n^2, n^2 - 5000, 5n^2 + 3n \in \Omega(n^2)$
- $n^3, 5n^3, 10n^3 - 6, n^2 \log n \ \in \Omega(n^2)$

- $(n+1)^2 = O(n^2) = \Omega(n^2) = \Theta(n^2)$

- **Najgori slučaj** nije gori od $O\big(g(n)\big)$
- **Najbolji slučaj** nije bolji od $\Omega(g(n))$

Primer

```
for (int x = 0; x < n; x++)
{
        isMin = true;

        for (int y = 0; y < n; y++)
        {
        if( array[x] > array[y])
                isMin = false;
        }

        if(isMin)
        break;
}
return array[x];
}
```

- $O(n^2)$
- $\Omega(n)$

Rekurzija

- Prebrajamo operacije poređenja u *if*-u, vraćanja vrednosti i sabiranja.

```
int recursiveFun1(int n)
{
    if (n <= 0)
        return 1;
    else
        return 1 + recursiveFun1(n-1);
}
```

$$f(n) = 3 + f(n-1) =$$
$$= 3 + 3 + f(n-2) =$$
$$= 3n + f(0) =$$
$$= 3n + 2 = O(n)$$

Rekurzija

- Prebrajamo operacije poređenja u *if*-u, vraćanja vrednosti i sabiranja. Radi jednostavnosti, uzmimo da je n deljivo sa 5.

```
int recursiveFun2(int n)
{
    if (n <= 0)
        return 1;
    else
        return 1 + recursiveFun2(n-5);
}
```

$$f(n) = 3 + f(n-5) =$$
$$= 3 + 3 + f(n - 2 \cdot 5) =$$
$$= 3 \cdot \frac{n}{5} + f(0) =$$
$$= 3 \cdot \frac{n}{5} + 2 = O(n)$$

Rekurzija

- Prebrajamo operacije poređenja u *if*-u, vraćanja vrednosti i sabiranja.

```
int recursiveFun3(int n)
{
    if (n <= 1)
        return 1;
    else
        return 1 + recursiveFun3(n/5);
}
```

$$
\begin{aligned}
f(n) &= 3 + f(n/5) = \\
&= 3 + 3 + f(n/5^2) = \\
&= 3 \cdot \log_5 n + f(1) = \\
&= 3 \cdot \log_5 n + 2 = O(\log n)
\end{aligned}
$$

Rekurzija

- Prebrajamo operacije poređenja u *if*-u.

```
void recursiveFun4(int n, int m, int o)
{
    if (n <= 0)
    {
        printf("%d, %d\n",m, o);
    }
    else
    {
        recursiveFun4(n-1, m+1, o);
        recursiveFun4(n-1, m, o+1);
    }
}
```

$$
\begin{aligned}
f(n) &= 1 + 2 \cdot f(n-1) = \\
&= 1 + 2 \cdot (1 + 2 \cdot f(n-2)) = \\
&= 1 + 2 + 2^2 \cdot f(n-2) = \\
&= 1 + 2 + 2^2 + \cdots + 2^{n-1} + 2^n \cdot f(0) = \\
&= 2^n - 1 + 2^n \cdot 1 = \\
&= 2 \cdot 2^n - 1 = O(2^n)
\end{aligned}
$$

Rekurzija

- Prebrajamo broj prolaza kroz petlju. Radi jednostavnosti, uzmimo da je n deljivo sa 5.

```
int recursiveFun5(int n)
{
    for (i = 0; i < n; i ++) {
        // do something
    }

    if (n <= 0)
        return 1;
    else
        return 1 + recursiveFun5(n-5);
}
```

$$f(n) = n + f(n-5) =$$
$$= n + (n-5) + f(n - 2 \cdot 5) =$$
$$= n + (n-5) + \cdots + 10 + 5 + f(0) =$$
$$= \frac{n(n+5)}{10} = O(n^2)$$

Prostorna kompleksnost

- Mera količine skladišnog prostora potrebnog algoritmu
- Koliko memorije je potrebno algoritmu u bilo kom trenutku izvršavanja
- Najgori slučaj
- O-notacija, red rasta

Primeri

```
int sum(int x, int y, int z) {
	int r = x + y + z;
	return r;
}
```

- 3 jedinice prostora za parametre
- 1 jedinica prostora za lokalnu promenljivu
- $f(n) = 4 = O(1)$

```
int sum(int a[], int n) {
	int r = 0;
	for (int i = 0; i < n; ++i) {
		r += a[i];
	}
	return r;
}
```

- ***n*** jedinica za niz ***a***
- 3 jedinice za promenljive ***n***, ***r***, ***i***
- $f(n) = n + 3 = O(n)$

Primeri

```
void arrayAdd(int a[], int b[], int c[], int n) {
	for (int i = 0; i < n; ++i) {
		c[i] = a[i] + b[i]
	}
}
```

- $f(n) = 3n + 2 = O(n)$

```
void matrixMultiply(int a[], int b[], int c[][], int n) {
	for (int i = 0; i < n; ++i) {
		for (int j = 0; j < n; ++j) {
			c[i] = a[i] + b[j];
		}
	}
}
```

- $f(n) = n^2 + 2n + 3 = O(n^2)$

Primeri

- Funkcija A – M jedinica prostora
- Funkcija B – N jedinica prostora
- Funkcija A pozove funkciju B
  - Jednom
  - Više puta (sekvencijalno, ne rekurzivno)
- U oba slučaja
  - $f(m,n) = m + n = O(m + n)$

Primer

- Rekurzivna funkcija za izračunavanje faktorijela

```
long long fakt(int n) {
      if (n<=1)
            return 1;
      else
            return n * fakt(n-1);
```

- Funkcija samu sebe zove ***n*** puta
- U jednom trenutku ***n*** poziva na steku

$$f(n) = n = O(n)$$