



# Algoritmika

1. szeminárium



# Írányelv - növekedési rend

Alapelv: Elhagyjuk a konstansokat és az alacsonyabb rendű tagokat

Pl.  $6n + 9 + n^2 = O(n^2)$

Az algoritmus futási ideje  $O(n^2)$  -  $n$  a bemenet mérete (pl. Tömb hossza)

# Írányelv - növekedési rend

Alapelv: Elhagyjuk a konstansokat és az alacsonyabb rendű tagokat  
függ a környezettől

Pl.  $6n + 9 + n^2 = O(n^2)$

Az algoritmus futási ideje  $O(n^2)$  -  $n$  a bemenet mérete (pl. Tömb hossza)

# Írányelv - növekedési rend

Alapelv: Elhagyjuk a konstansokat és az alacsonyabb rendű tagokat

függ a környezettől

elég nagy bemenet esetén lényegtelen

Pl.  $6n + 9 + n^2 = O(n^2)$

Az algoritmus futási ideje  $O(n^2)$  -  $n$  a bemenet mérete (pl. Tömb hossza)

# A gyakorlatban

Hány műveletet végez az alábbi algoritmus?

```
int MaxDifference(int vals[], int n) {  
    int max = 0;  
    for (int i=0; i< n; i++)  
        for (int j=0; j<n; j++)  
            if (vals[i]-vals[j]>max)  
                max = vals[i]-vals[j];  
  
    return max;  
}
```


# A gyakorlatban

Hány műveletet végez az alábbi algoritmus?

```
int MaxDifference(int vals[], int n) {  
    int max = 0;  
    for (int i=0; i< n; i++)  
        for (int j=0; j<n; j++)  
            if (vals[i]-vals[j]>max)  
                max = vals[i]-vals[j];  
    return max;  
}
```



# A gyakorlatban

Hány műveletet végez az alábbi algoritmus?

```
int MaxDifference(int vals[], int n) {  
      $O(1)$   
    for (int i=0; i< n; i++)  
        for (int j=0; j<n; j++)  
            if (vals[i]-vals[j]>max)  
                max = vals[i]-vals[j];  
    return max;  
}
```

# A gyakorlatban

Hány műveletet végez az alábbi algoritmus?

```
int MaxDifference(int vals[], int n) {  
      $O(1)$   
    for (int i=0; i< n; i++)  
        for (int j=0; j<n; j++)  
            if (vals[i]-vals[j]>max)  
                max = vals[i]-vals[j];  
      $O(1)$   
}
```



# A gyakorlatban

Hány műveletet végez az alábbi algoritmus?

```
int MaxDifference(int vals[], int n) {
```

  $O(1)$

```
    for (int i=0; i< n; i++)
```

```
        for (int j=0; j<n; j++)
```

```
            if (vals[i]-vals[j]>max)
```

```
                max = vals[i]-vals[j];
```

  $O(1)$

```
}
```

# A gyakorlatban

Hány műveletet végez az alábbi algoritmus?

```
int MaxDifference(int vals[], int n) {  
    [redacted]  $O(1)$   
    for (int i=0; i< n; i++)  
        for (int j=0; j<n; j++)  
            [redacted]  $O(1)$   
    [redacted]  $O(1)$   
}
```

# A gyakorlatban

Hány műveletet végez az alábbi algoritmus?


```
int MaxDifference(int vals[], int n){
```

  $O(1)$

```
    for (int i=0; i< n; i++)
```

```
        for (int j=0; j<n; j++)
```

  $O(1)$


  $O(1)$

```
}
```


# A gyakorlatban

Hány műveletet végez az alábbi algoritmus?

```
int MaxDifference(int vals[], int n) {
```

  $O(1)$

```
    for (int i=0; i< n; i++)
```

  $O(n)$

  $O(1)$

```
}
```

# A gyakorlatban

Hány műveletet végez az alábbi algoritmus?


```
int MaxDifference(int vals[], int n) {
```



$O(1)$



$O(n^2)$



$O(1)$

```
}
```

# Irányelvek algoritmusok elemzésekor

Növekedési rend: - csak nagy bemenetekre érvényes

- az állandó tényezők és alacsonyabb rendű tagok miatt ez az értékelés hibás lehet kis bemenetre
- csak a nagy méretű feladatok "érdekesek"

Pl:  $n \log(n)$  "jobb mint"  $n^2$

# Futási idő

Feladat: Állapítsuk meg egy adott  $t$  számról, hogy eleme-e az  $A$ ,  $n$  elemű tömbnek.

```
for i = 1 to n do
    if A[i] == t then
        return TRUE
return FALSE
```

Mi az algoritmus futási ideje?

# Futási idő

Feladat: Állapítsuk meg egy adott  $t$  számról, hogy eleme-e az  $A$ ,  $n$  elemű tömbnek.

```
for i = 1 to n do
    if A[i] == t then
        return TRUE
return FALSE
```

Mi az algoritmus futási ideje?  $O(n)$



# Futási idő

Feladat: Állapítsuk meg egy adott  $t$  számról, hogy eleme-e az  $A$  vagy  $B$ ,  $n$  elemű tömböknek.

```
for i = 1 to n do
    if A[i] == t then
        return TRUE
for i = 1 to n do
    if B[i] == t then
        return TRUE
return FALSE
```

Mi az algoritmus futási ideje?

# Futási idő

Feladat: Állapítsuk meg egy adott  $t$  számról, hogy eleme-e az  $A$  vagy  $B$ ,  $n$  elemű tömböknek.

```
for i = 1 to n do
    if A[i] == t then
        return TRUE
for i = 1 to n do
    if B[i] == t then
        return TRUE
return FALSE
```

Mi az algoritmus futási ideje?  $O(n)$

# Futási idő

Feladat: Állapítsuk meg az  $A$  és  $B$ ,  $n$  elemű, tömbökről, hogy van-e közös elemük?

```
for i = 1 to n do
    for j = 1 to n do
        if A[i] == B[j] then
            return TRUE
return FALSE
```

Mi az algoritmus futási ideje?

# Futási idő

Feladat: Állapítsuk meg az  $A$  és  $B$ ,  $n$  elemű, tömbökről, hogy van-e közös elemük?

```
for i = 1 to n do
    for j = 1 to n do
        if A[i] == B[j] then
            return TRUE
return FALSE
```

Mi az algoritmus futási ideje?  $O(n^2)$

# Futási idő

Feladat: Állapítsuk meg az  $A$ ,  $n$  elemű, tömbről, hogy van-e olyan eleme ami többször fordul elő?

```
for i = 1 to n do
    for j = i+1 to n do
        if A[i] == A[j] then
            return TRUE
return FALSE
```

Mi az algoritmus futási ideje?

# Futási idő

Feladat: Állapítsuk meg az  $A$ ,  $n$  elemű, tömbről, hogy van-e olyan eleme ami többször fordul elő?

```
for i = 1 to n do
  for j = i+1 to n do
    if A[i] == A[j] then
      return TRUE
return FALSE
```

Mi az algoritmus futási ideje?  $O(n^2)$

# Futási idő

Mi az alábbi algoritmus futási ideje?

```
int f1(int n){  
    i =1;  
    while (i ≤ n)  
        i ← 2 * i;  
    return i;  
}
```

# Futási idő

Mi az alábbi algoritmus futási ideje?  $O(\log n)$

```
int f1(int n){  
    i =1;  
    while (i ≤ n)  
        i ← 2 * i;  
    return i;  
}
```



# Futási idő

Mi az alábbi algoritmus futási ideje?

```
void f2(int n){  
    j=n;  
    while(j>1){  
        i=1;  
        while (i ≤ n)  
            i=2*i;  
        j= j/3;  
    }  
}
```

# Futási idő

Mi az alábbi algoritmus futási ideje?  $O(\log^2 n)$

```
void f2(int n) {  
    j=n;  
    while (j>1) {  
        i=1;  
        while (i ≤ n)  
            i=2*i;  
        j= j/3;  
    }  
}
```

## Big-O Complexity

