Tema 2: Algoritmanalys

Wilhelm Durelius `widu7139`

28 januari 2026

1 Utskrift

Avgör tidskomplexiteten med big O för pseudokoden nedan. I pseudokoden kan vi komma åt tecken från strängar med array-syntax i $O(1)$-tidskomplexitet.

```
1  func VerifyEmail(string email) {
2    if email.length() < 5
3        throw Exception("Email is too short")
4    bool containsDot = false
5    for int i = email.length() - 1; i >= email.length() - 5; i-- {
6        if email[i] == "." {
7            containsDot = true
8            break
9          }
10     }
11   if !containsDot
12       throw Exception("Email does not have a domain")
13 }
```

1.1 Svar

I ett worst-case scenario så körs loopen 5 gånger, resten av koden är $O(1)$. Med tanke på att loopen inte ökar i storlek när n ($email.length()$) ökar, utan toppar på 5, så har hela metoden en tidskomplexitet på $O(1)$, den är konstant.

2 Utskrift

Avgör tidskomplexiteten med big O för pseudokoden nedan. I pseudokoden använder vi StringBuilder för att skriva till strängar i $O(1)$-tidskomplexitet. Vi kommer även åt tecken i strängar med array-syntax i $O(1)$.

```
1  func SanitizeString(string text) {
2     StringBuilder newText = new StringBuilder()
3     for int i = 0; i < text.length(); i++ {
4      if text[i] == "/" {
5        break
6      } else if text[i] != "$" {
7        newText.write(text[i])
8        }
9     }
10    return newText.string()
11 }
```

2.1 Svar

I ett worst-case scenario så körs loopen N gånger ($text.length()$), resten av koden är $O(1)$. Med tanke på att loopen ökar i storlek i takt med storleken på N, så är tidskomplexiteten på metoden $O(n)$.

3 Utskrift

Avgör tidskomplexiteten med big O för pseudokoden nedan. Java-versionen av lista.subList(start, end) vi utgår ifrån returnerar i $O(1)$ tidskomplexitet.

```
1 int maxSize = 500
2 func StripMailList(List mailList) {
3    while mailList.size() > maxSize {
4     int mailSize = mailList.size()
5       mailList = mailList.subList(mailSize / 2, mailSize)
6    }
7    return mailList
8 }
```

3.1 Svar

I ett worst-case scenario så körs loopen $log(N)$ gånger ($\frac{N}{2}$ vid varje körning), resten av koden är $O(1)$. N behöver alltså dubblas för att loopen ska lägga till ett extra varv, vilket gör metoden till $O(\log n)$.

4 Utskrift

Avgör tidskomplexiteten med big O för pseudokoden nedan.

```
1  func CountEvenNumbers(List param) {
2    int evenCount = 0
3    int n = param.size()
4      if n % 2 == 0 {
5          n = n * n
6      }
7      for int i = 0; i < n; i++ {
8          if i % 2 == 0 {
9              evenCount++
10           }
11     }
12     return evenCount;
13 }
```

4.1 Svar

I ett worst-case scenario så körs loopen $n * n$ gånger, resten av koden är $O(1)$. Antalet varv loopen tar i detta fall ökar alltså kvadratiskt i takt med att parametern ökar. Därför är metodens tidskomplexitet $O(N^2)$.