

# GitHub

Doc. Dr .Mehmet Akif Cifci

- Viyana Teknik Üniversitesi (Avusturya)
- Klaipeda Üniversitesi (Litvanya)
- Bandırma Onyedi Eylül Üniversitesi

To follow and connect

<https://github.com/themanoftalent>

<https://www.linkedin.com/in/themanoftalent/>

<https://www.researchgate.net/profile/Mehmet-Akif-Cifci>

# ALGORİTMALAR





# Algoritma Nedir?

Algoritmalar bir problemi çözmek veya belirli bir görevi yerine getirmek için kullanılan adım adım talimatların bir koleksiyonudur. Algoritmaların belirli özellikleri vardır ve geniş bir kullanım alanı bulunmaktadır. Bu nedenle, algoritmaların anlaşılması ve etkili bir şekilde kullanılması bilgisayar biliminin ve diğer alanların temelini oluşturur.

---

# Algoritma Özellikleri

- Açıklık:  
Bir algoritmanın anlaşılabilir ve açık bir şekilde tanımlanması gerekmektedir.
  - Sonluluk:  
Her adımdan sonra algoritmanın bir sonraki adıma geçmesi ve sonunda durması gerekmektedir.
-

# Algoritma Özellikleri

- **Etkinlik:**  
Algoritmanın her adımının belirli bir süre içinde tamamlanması gerekmektedir.
  - **Çıktı Üretme:**  
Herhangi bir belirli girdi için algoritmanın doğru bir çıktı üretmesi gerekir.
-

# Algoritma Kullanım Alanları

- Algoritmaların geniş bir kullanım alanı vardır. Örneğin, **matematikte**, bir problemi çözmek için kullanılan bir algoritma olabilirler. Bilgisayar biliminde, bir programın nasıl yazılacağını belirlemek için algoritmalar kullanılır. **Finansal analizde** veya **veri bilimi alanında** da algoritmalar yaygın olarak kullanılır. Dolayısıyla, algoritmaların sadece bir problemi çözmekle kalmayıp aynı zamanda birçok alanda kullanılan önemli araçlar olduğunu söyleyebiliriz.
-

# Algoritma Tasarımı Temelleri

Algoritma tasarımı, bir problemi çözmek için algoritma oluşturma sürecidir. Bu süreç;

- Problemi anlama
- Algoritma tasarlama
- Algoritmanın analizi
- Algoritmayı uygulama

adımlarını içerir.

---

# Algoritma Tasarımında Dikkat Edilmesi Gerekenler

- **Verimlilik:**

Algoritmanın gereksiz yere kaynak tüketmeden verimli bir şekilde çalışmasını sağlar.

- **Doğruluk:**

Algoritmanın problemin gereksinimlerini tam olarak karşıladığından emin olmayı sağlar.

- **Anlaşılabilirlik:**

Algoritmanın kolayca okunabilir ve anlaşılabilir olmasını sağlar, bu da algoritmanın diğer geliştiriciler tarafından kullanılabilirliğini artırır.

---



# Akış Diyagramları

Sürecin akış hattı



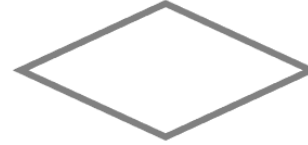
Başlangıç / Bitiş



Süreç adımı



Karar



Giriş çıkış



Belge



Önden  
tanımlanmış işlem



Hazırlık



Sayfa dışı  
bağlantı/konektör



# Akış Diyagramları

Akış diyagramları, bir algoritmanın adımlarını görsel olarak temsil etmek için kullanılan bir tekniktir. Bu diyagramlar, çizgiler, kareler, elipsler ve diğer semboller kullanılarak algoritmanın akışını gösterir.

---



# Pseudo-code

Pseudo-code, bir algoritmanın programlama diline bağlı olmayan, insanların anlayabileceği basit bir dilde ifade edilmesini sağlayan bir tekniktir. Pseudo-code, algoritmanın mantığını anlamak ve geliştirmek için kullanılır, ancak doğrudan bir programlama dili olarak uygulanamaz. Bu nedenle, pseudo-code'u gerçek bir programlama diline dönüştürmek gerekir.

---

# Pseudo-code ve Akış Diyagramları

Akış diyagramları ve pseudo-code, algoritmaların görsel ve metinsel temsillerini sağlayan önemli tekniklerdir. Akış diyagramları, algoritmanın adımlarını görsel olarak temsil ederken, pseudo-code daha anlaşılabilir bir dilde algoritmayı ifade eder. Bu teknikler, algoritmaların **geliştirilmesi**, **anlaşılması** ve **optimize edilmesi** süreçlerinde önemli bir rol oynar.

---

# Pseudo-code Nasıl Yazılır?

Pseudo-code yazarken, algoritmanın adımlarını basit bir dilde ifade etmek önemlidir. Her bir adım, algoritmanın bir parçasını veya belirli bir işlemi temsil eder. Bu adımlar genellikle doğal dil kullanılarak yazılır ve algoritmanın mantığını net bir şekilde ortaya koyar. Pseudo-code'un anlaşılabilirliği ve okunabilirliği için, açıklayıcı değişken ve ifadeler kullanılmalıdır.

---

# Pseudo-code Nasıl Yazılır?

Pseudo-code yazarken, algoritmanın akışını ve kontrol yapılarını göz önünde bulundurmak önemlidir. Bu, döngüler, karar yapıları ve işlem sırasını içerir. Bu yapılar, algoritmanın belirli koşullara veya durumlara nasıl tepki vereceğini ve nasıl işleyeceğini belirler. Pseudo-code'un bu yapıları kullanarak algoritmanın akışını açıklıkla ifade etmesi gerekir.

---

# Pseudo-code

## Örnek-1

Bu pseudo-code, bir karenin alanını hesaplayan basit bir algoritmayı temsil eder. Algoritmanın adı KareAlanıHesapla olarak belirlenmiştir. Algoritma, kullanıcıdan karenin bir kenar uzunluğunu girmesini ister. Ardından, girilen kenar uzunluğunu kullanarak karenin alanını hesaplar. Son olarak, hesaplanan alanı ekrana yazdırır.

java

Copy code

ALGORİTMA KareAlanıHesapla

// Girdi: Bir kenar uzunluğu

// Çıktı: Kare alanı

BAŞLA

kenarUzunluğu = kullanıcıdan\_al("Karenin kenar uzunluğunu giriniz")

kareAlanı = kenarUzunluğu \* kenarUzunluğu

ekrana\_yaz("Karenin alanı:", kareAlanı)

SON


# Pseudo-code

## Örnek-1

---

Pseudo-code, gerçek bir programlama diline bağlı olmadığı için, algoritmanın mantığını anlamak için basit bir dilde ifade edilmiştir. Bu pseudo-code'u gerçek bir programlama diline dönüştürmek mümkündür.

python

 Copy code

```
def kare_alanı_hesapla():  
    kenar_uzunluğu = float(input("Karenin kenar uzunluğunu giriniz: "))  
    kare_alanı = kenar_uzunluğu * kenar_uzunluğu  
    print("Karenin alanı:", kare_alanı)  
  
kare_alanı_hesapla()
```



# Algoritma Analizi ve Karmaşıklık

- **Algoritma Analizi:**

Bir algoritmanın performansını ve verimliliğini inceleyen bir süreçtir. Bu analiz, algoritmanın işlemci zamanı, bellek kullanımı ve diğer kaynaklar üzerindeki etkisini değerlendirir.

- **Karmaşıklık:**

Bir algoritmanın çalışma süresinin veya kullanılan kaynakların miktarının, problemin boyutu ile nasıl değiştiğini ifade eder. Karmaşıklığı belirlemek için zaman karmaşıklığı ve alan karmaşıklığı gibi kriterler kullanılır. İyi bir algoritma, en kötü durumda bile kabul edilebilir bir performans gösterir.

---



# İyi Bir Algoritma

- en kötü durumda bile kabul edilebilir bir performans gösterir.
  - problem boyutu arttıkça performansının kötüleşmediği veya en azından kabul edilebilir bir şekilde kötüleştiği bir özellik gösterir.
  - tasarımı, karmaşıklık analizi ile başlar ve algoritmanın etkili bir şekilde performans göstermesini sağlar.
-

# Karmaşıklık Örneği-1

Bir karmaşıklık örneği olarak, bir algoritmanın **bir dizideki en büyük veya en küçük öğeyi bulma** işlemini ele alalım. Bu algoritma, bir dizi üzerinde dolaşarak her öğeyi diğerleriyle karşılaştırarak en büyük veya en küçük öğeyi bulabilir. Bu tür bir algoritmanın zaman karmaşıklığı  $O(n)$  olacaktır çünkü en kötü durumda, dizinin tüm öğeleriyle karşılaştırma yapılması gerekecektir.

---

# Karmaşıklık Örneği-1

```
ALGORİTMA EnBuyukBul
```

```
BAŞLA
```

```
    enBuyuk = dizinin_ilk_ögesi
```

```
    DÖNGÜ i = 1'den başlayarak dizinin_son_ögesine_kadar
```

```
        EĞER dizinin_i.ögesi > enBuyuk
```

```
            enBuyuk = dizinin_i.ögesi
```

```
        SONSUZ
```

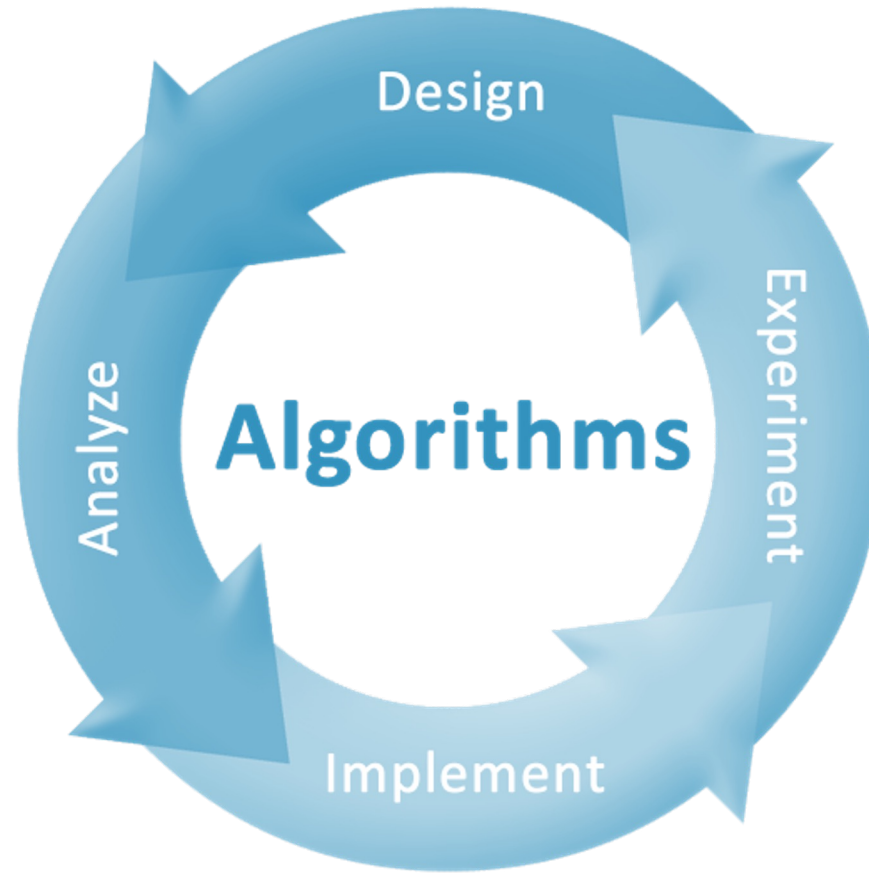
```
    ekrana_yaz("Dizinin en büyük ögesi:", enBuyuk)
```

```
SON
```

# Karmaşıklık Örneği-1

Ancak, daha etkili bir algoritma olan sıralama algoritmaları kullanarak bu işlemi gerçekleştirebiliriz. Örneğin, hızlı sıralama algoritması  $O(n \log n)$  karmaşıklığına sahiptir. Bu algoritma, diziyi bölerek ve her alt dizide sıralama işlemini gerçekleştirerek çalışır. Bu şekilde, dizideki en büyük veya en küçük öğeyi bulmak için bir dizi sıralandığında, sıralı dizinin ilk veya son öğesine bakmak yeterli olacaktır. Bu durumda, sıralama algoritmasının zaman karmaşıklığı daha düşüktür ve algoritmanın performansı daha iyidir.

---



**Doc. Dr .Mehmet Akif Cifci**

- Viyana Teknik Üniversitesi (Avusturya)
- Klaipeda Üniversitesi (Litvanya)
- Bandırma Onyedi Eylül Üniversitesi

**To Follow and Connect**

<https://github.com/themanoftalent>

<https://www.linkedin.com/in/themanoftalent/>

<https://www.researchgate.net/profile/Mehmet-Akif-Cifci>