FIRAT ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ
BİLGİSAYAR BİLİMLERİNE GİRİŞ DERSİ
LABORATUVAR FÖYÜ

HAFTA: 7

KONU: ALGORİTMA VE DÖNGÜ DEĞERLENDİRMELERİ

Doç. Dr. MUHAMMET BAYKARA

Arş. Gör. Z. BEYZA METİN

Algoritma Değerlendirme ve Çözüm Yöntemleri

Giriş : Bir programcı olarak görevimiz sadece çalışan bir kod yazmak değil, *en iyi* çalışan kodu yazmaktır. Peki, bir kodu "iyi" yapan nedir? Eğer bir hedefe giden iki farklı yol keşfedersek hangi yolu seçmeliyiz?

Bu hafta, bu kritik soruların cevabını arayacağız. Önce algoritma yazacağız, sonra da yazdığımız algoritmayı bir mühendis gibi "değerlendireceğiz".

Algoritma Değerlendirmesi Nedir?	Çözüm Yöntemleri

• "Bir algoritmanın 'iyi' olmasını belirleyen iki ana kriter vardır:"
 	
• **1. Zaman Verimliliği (Hız):** Algoritma işini ne kadar *az adımda* (hızlı) bitiriyor?
 	
• **2. Alan Verimliliği (Hafıza):** Algoritma çalışırken bilgisayarın hafızasını (RAM) ne kadar *az* kullanıyor?
 	
• **Kritik Soru :** Bu föydeki kilit soru budur. "Eğer girdiğimiz sayı (`n`) çok büyürse (örn: 5 yerine 50.000 girersek) algoritmanın hızı ve hafıza kullanımı nasıl etkilenir?"
	• **İteratif Yaklaşım (Döngüsel):**
	
	• Bir işi tamamlanana kadar **tekrar tekrar** yapmaktır.
	
	• **Anahtar Kelimeler:** `for`, `while`, sayaç, birikimli sonuç (`toplam = toplam + ...`).
	
	• **Analoji:** Bir merdivenin 1. basamağından başlayıp, son basamağa kadar **tek tek** çıkmak.
	
	• **2. Rekürsif Yaklaşım (Özyineli):**
	
	• Büyük bir problemi, kendisinin daha küçük bir versiyonunu çözerek halletmektir. Bir fonksiyonun **kendi kendini çağırmasıdır**.
	
	• **Kritik Kurallar:**
	  • Mutlaka bir **"Temel Durum" (Base Case)** yani durma noktası olmalıdır.
	  • Mutlaka "Rekürsif Adım" (kendini çağırma) olmalıdır.
	
	• **Analoji:** 5!'i hesaplamak için 4!'in sonucunu beklemek. 4! de 3!'ü bekler... Ta ki 0! (temel durum) 1 sonucunu verene kadar.

Doç. Dr. MUHAMMET BAYKARA
Arş. Gör. Z. BEYZA METİN

Bu Haftanın Yol Haritası

1- Problemi **İteratif (Döngülü)** yöntemle tasarlayacaksınız.

2- Problemi **Rekürsif (Özyineli)** yöntemle tasarlayacaksınız.

3- Her iki yöntem için de 4 temel adımı (**Metinsel Algoritma**, **Sözde Kod**, **Akış Şeması** ve **Java Kodu**) föydeki ilgili boşluklara *siz* üreteceksiniz.

4- Çözümlerinizi belirli bir girdi için *adım adım çalıştırıp* (İzleme Tablosu / Çağrı Yığını) çıktısını *siz* tespit edeceksiniz.

5- Son olarak, ürettiğiniz iki farklı çözümü (İteratif vs Rekürsif) Hız ve Hafıza verimliliği açısından *siz* değerlendireceksiniz.

Problem 1: Bir Sayının Rakamlarını Toplama

Problem Tanımı: Bu bölümde, kullanıcıdan alınan pozitif bir tam sayının 472 sayısının rakamlarının toplamını (örn: 4 + 7 + 2 = 13) bulan algoritmayı iki farklı yöntemle tasarlayacağız.

İpucu Kutusu (Java Operatörleri):

- Bir sayının **son rakamını** almak için "mod alma" (Modulo) operatörünü kullanabilirsiniz: `int sonRakam = 472 % 10;` (sonRakam = 2 olur)
- Bir sayının **son rakamını atmak** için "tam sayı bölmesi" (Integer Division) kullanabilirsiniz: `int kalanSayi = 472 / 10;` (kalanSayi = 47 olur)

YÖNTEM 1: İTERATİF (DÖNGÜLÜ) ÇÖZÜM

Bu yöntemde, sayının rakamlarını `while` döngüsü ve ipucu kutusundaki operatörleri kullanarak toplayacağız.

Görev 1.1: Metinsel Algoritma *(İpucu: Günlük konuşma diliyle, adım adım ne yapılması gerektiğini yazın. Örn: 1. Başla, 2. 'toplam' diye bir kutu oluştur ve içine 0 koy...)*

Cevap:

Görev 1.2: Sözde Kod (Pseudocode) *(İpucu: Metinsel algoritmayı, programlama mantığına (DÖNGÜ, DEĞİŞKEN) dönüştürün. Java değil, evrensel kod mantığı!)* `FONKSIYON rakamToplaIteratif(sayi): toplam = 0 DÖNGÜ (sayi > 0 OLDUĞU SÜRECE): ...`

Cevap:

Görev 1.3: Akış Şeması (Flowchart) (İpucu: Sözde koddaki her adımı doğru geometrik şekillerle (karar, işlem, döngü) çizin. Dijital olarak çizebilir veya kağıda çizip fotoğrafını buraya ekleyebilirsiniz.)

Cevap:

Görev 1.4: Java Kodu (İpucu: Sözde kodun Java dilindeki karşılığı. `while` döngüsü ve ipucu kutusundaki operatörleri (`%`, `/`) kullanarak boşlukları doldurun. Örnek kod aşağıdaki gibidir.)

```
public int rakamToplaIteratif(int sayi)

{ int toplam = ______;

while (sayi > ______) { int sonRakam = sayi % ______; toplam = toplam + ______;

sayi = sayi / ______; } return ______; }
```

Cevap:

YÖNTEM 2: REKÜRSİF (ÖZYİNELİ) ÇÖZÜM

Bu yöntemde, problemi 'Son Rakam' + 'Kalan Rakamların Toplamı' olarak ikiye böleceğiz. Aşağıda size verilen metinsel algoritmaya göre kalan boşlukları doldurunuz.

Görev 2.1: Metinsel Algoritma

- Başla (fonksiyon `sayi` ile çağrıldı)
- Karar: `sayi` 0'a eşit mi? (Temel Durum)
- Evet ise -> 0 döndür.
- Hayır ise -> `(sayi % 10) + rakamToplaRekursif(sayi / 10)` sonucunu döndür.

- Bitir.

Görev 2.2: Sözde Kod (Pseudocode)

Cevap:

Görev 2.3: Akış Şeması (Flowchart) (Dijital olarak çizin veya fotoğrafını ekleyin.)

Cevap:

Görev 2.4: Java Kodu (İpucu: Sözde kodun Java'daki karşılığı. `if-else` yapısını kullanarak boşlukları doldurun.)

Cevap:

GÖREV 3: ADIM ADIM ÇALIŞTIRMA (ÇIKTI TESPİTİ)

Şimdi yazdığınız iki metodu da `sayi = 472` girdisi için bilgisayar gibi çalıştıralım

İteratif Çözüm İzleme Tablosu (Trace Table)

? Olan boşlukları doldurunuz.

Döngü Adımı	sayi > 0 koşulu?	toplam Değeri	sayi Değeri
Döngüden Önce	-	0	472
1. Adım	Doğru *(472 > 0)*	2 *(toplam = 0 + (472 % 10)*	47*(Çünkü `sayi = 472 / 10` oldu)*
2. Adım	?	?	?
3. Adım	?	?	?
4. Adım (Çıkış)	Yanlış	??	

Algoritma Analizi

BÖLÜM 1: Problem 1'in Değerlendirmesi (Rakam Toplama)

Aynı problemi (Rakam Toplama) iki farklı yöntemle çözdünüz ve 472 girdisi için ikisinin de aynı sonucu (13) verdiğini gördünüz. Peki, *hangisi daha iyi bir çözümdü?*

Değerlendirme

Soru 1: Hız açısından, iki yöntemde ne kadar adım atıldı? Yöntemlerin birinde her adımda *yeni bir fonksiyon çağırmanın* (sistemin stack'e bir şeyler eklemesi/çıkarması) ek bir maliyeti (overhead) var mıdır? Bu durumda, hız açısından hangi çözüm *biraz* daha verimlidir?"

Cevap:

Soru 2 : İteratif Çözüm hangi değişkenleri için hafızada yer tuttu?

Cevap:

Soru 3 : Rekürsif Çözümünüzde kaç farklı fonksiyon hafızaya alındı ve bekletildi.

Cevap:

Soru 4: Peki, girdi boyutu `n = 987654321` (9 basamaklı) olsaydı, rekürsif çözüm hafızada kaç fonksiyonu bekletmek zorunda kalırdı?"

Cevap:

Soru 5: Girdi boyutu `n` büyüdükçe, hangi çözümün hafıza kullanımı **sabit** kalır?

Cevap:

Çıkarım: Problem 1'den ne öğrendiniz? Hangi çözümün hangi avantajı/dezavantajı olduğunu bir cümleyle özetleyin.

Problem 2 - Fibonacci Serisi

Rakam Toplama probleminde, rekürsif çözüm *biraz* verimsizdi. Şimdi göreceğimiz Fibonacci probleminde ise, 'kötü' tasarlanmış bir rekürsif çözümün nasıl *felaket düzeyde* yavaş olabileceğini bizzat test edeceğiz.

Problem Tanımı

Fibonacci Serisi: Her sayının, kendinden önceki iki sayının toplamı olduğu seridir. (0, 1, 1, 2, 3, 5, 8, 13...)

Kural: `Fib(n) = Fib(n-1) + Fib(n-2)`

Temel Durumlar (Durma Noktaları): `Fib(0) = 0` ve `Fib(1) = 1`

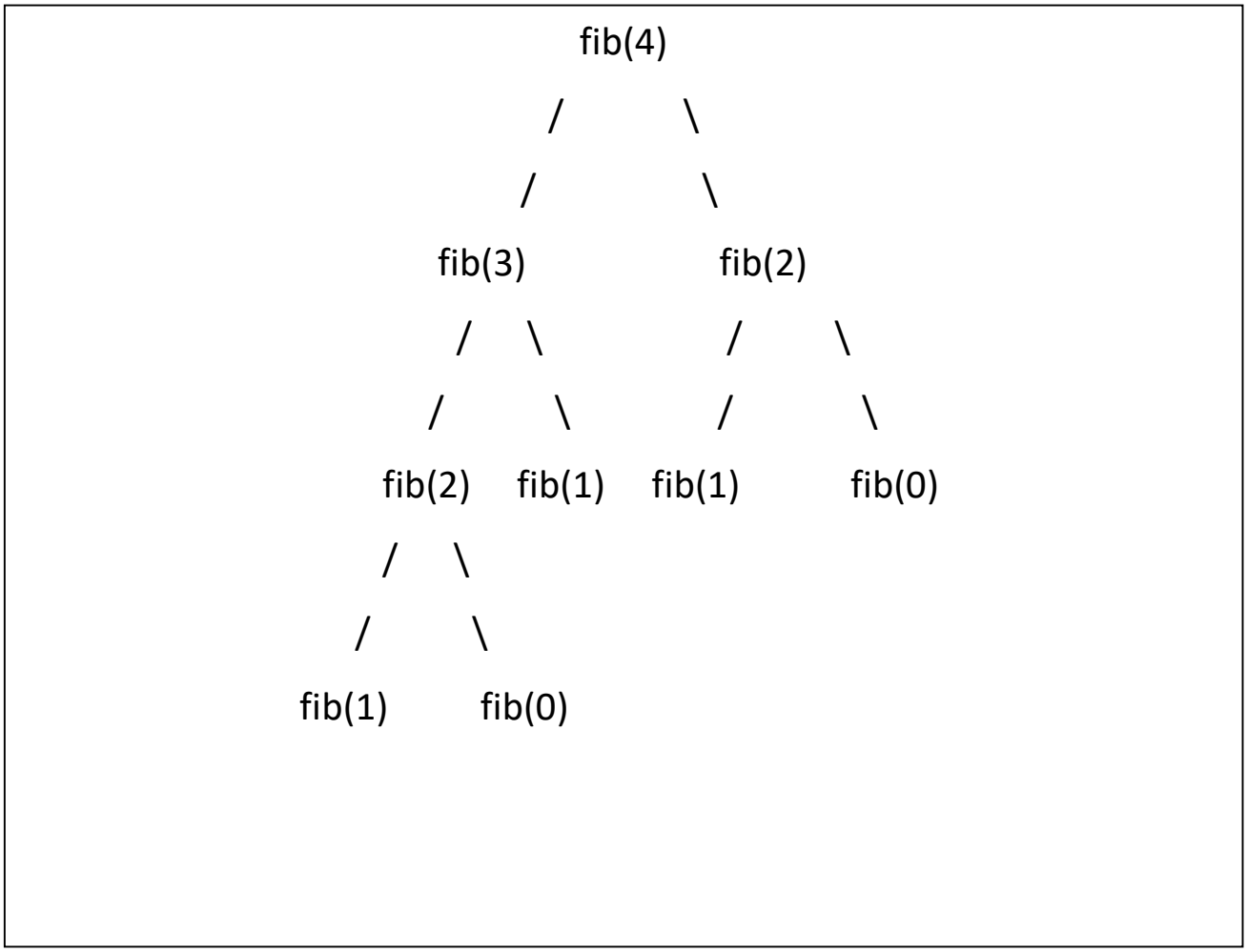
YÖNTEM 1: REKÜRSİF (ÖZYİNELİ) ÇÖZÜM (Naive – Verimsiz Yöntem)

Aşağıda, Fibonacci kuralının doğrudan Java'ya aktarılmış halini (Naive Çözüm) görüyorsunuz. Bu sefer kodu biz veriyoruz, sizin göreviniz bu kodu *analiz etmek.*

```
public int fibRekursif(int n)

{

 // Temel Durum 1

if (n == 0) { return 0; }

 // Temel Durum 2

if (n == 1) { return 1; }

// Rekürsif Adım (Problemin başladığı yer)

else { return fibRekursif(n - 1) + fibRekursif(n - 2); } }
```

Görev 5: Adım Adım Çalıştırma (Verimsizliğin Net Hali)

Bu kodun neden 'kötü' olduğunu anlamak için, `fibRekursif(4)` çağrısının 'Çağrı Ağacını' (Call Tree) birlikte çizelim. Bu, `fib(4)`'ü çözmek için bilgisayarın hangi fonksiyonları çağırdığını gösterir.



Görev 6: Akıl Yürütme (Çıktı Tespiti ve Analiz)

Yukarıdaki ağaca göre, fib(2) fonksiyonu toplamda **kaç kez** çağrıldı (hesaplandı)?

Cevap: _____

Bilgisayar fib(2)'nin sonucunu (yani 1) bir kez bulduktan sonra, onu tekrar tekrar (farklı dallarda) hesaplaması verimli midir, yoksa gereksiz bir iş tekrarı mıdır?

Cevap: _____

BÖLÜM 3: Problem 2'nin Düzeltilmesi (İteratif Fibonacci)

Az önce fibRekursif çözümünün fib(2) gibi aynı değerleri defalarca hesaplayarak sistemi nasıl yorduğunu (Zaman Verimsizliği) gördük.

Şimdi sizin göreviniz, bu problemi **İteratif (Döngülü)** yöntemle çözen 'iyi' algoritmayı tasarlamak. Bu çözüm, her Fibonacci sayısını *sadece bir kez* hesaplayacak.

Görev 7: Metinsel Algoritma (İteratif Fibonacci) (İpucu: 0 ve 1 temel durumdur. Bir döngü kullanarak, `a=0` ve `b=1` ile başlayıp, bu sayıları `temp = a + b` gibi bir formülle kaydırarak ilerlemeniz gerekecek.)

Cevap:

Görev 8: Sözde Kod (İteratif Fibonacci) (İpucu: Bir `FOR` döngüsü 2'den `n`'ye kadar gidebilir. Döngü içinde `a`, `b` ve `temp` değişkenlerini güncelleyin. `n=0` ve `n=1` durumlarını en başta kontrol etmeyi unutmayın.)

Cevap:

Görev 9: Akış Şeması (İteratif Fibonacci) (Dijital olarak çizin veya fotoğraf ekleyin.)

Cevap:

Görev 10: Java Kodu (Boşluk Doldurma) (İpucu: `n=0` ve `n=1` durumlarını `if` ile kontrol edin. Sonra döngüyü kurun.)

Cevap:

BÖLÜM 4: TEST İki Yöntemi Karşılaştırma

Yazdığınız fibIteratif kodu ile föyünüzdeki fibRekursif kodunu bilgisayarınızda veya laboratuvar bilgisayarınızda (IDE'nizde) çalıştırın ve sonuçları gözlemleyin.

Görev 11: Çıktı Tespiti ve Zamanlama

- `fibRekursif(40)` fonksiyonunu çalıştırın. Sonucun ekrana gelmesi ne kadar sürdü? (Not: Bilgisayarınızın gücüne göre bu işlem *dakikalarca* sürebilir veya program kilitlenebilir.)
Gözlem: _____

- `fibIteratif(40)` fonksiyonunu çalıştırın. Sonucun ekrana gelmesi ne kadar sürdü?
Gözlem: _____

BÖLÜM 5: Ne Öğrendik?

Bu föyde, 'rekürsif' çözümün her zaman 'kötü' veya 'iteratif' çözümün her zaman 'iyi' olmadığını gördük. Her şey, *probleme* ve *tasarımınıza* bağlıdır.

> Bir yazılım mühendisi olarak, bir problemi çözmek için İteratif ve Rekürsif yöntemler arasında seçim yaparken dikkat edeceğiniz **iki ana faktör** nedir? *(İpucu: Hız / Gereksiz tekrarlar / Zaman Verimliliği) /Hafıza kullanımı / StackOverflow riski / Alan Verimliliği)*

1. Faktör 1: _____
2. Faktör 2: _____

TESLİMAT

- Bu 4 sayfalık Word belgesindeki **tüm görevleri** (boşluk doldurmalar, metinsel algoritmalar, sözde kodlar, akıl yürütme cevapları ve akış şeması çizimleri) eksiksiz olarak tamamlayın.

- **Akış Şemalarını (Görev 1.3, 2.3 ve 9)** dijital bir araçla (draw.io, vb.) çizebilir ve ekran görüntüsünü ilgili boşluğa yapıştırabilir veya temiz bir kağıda elle çizip okunaklı bir fotoğrafını ekleyebilirsiniz.

- **ÖNEMLİ NOT:** Föydeki cevap alanları (boşluklar) birer tahmindir. Cevaplarınız (özellikle metinsel algoritma veya kod kısımları) daha uzunsa, **sayfa düzenini bozmadan boşlukları kendinize göre ayarlayabilirsiniz.**

- Tamamladığınız bu Word dosyasını (`.docx`) adınızı, soyadınızı ve numaranızı içerecek şekilde yeniden adlandırın.

- Dosyanızı **Google Classroom**'daki ilgili "Hafta 7 Föyü" ödevine yükleyin.

- **SÜRE:** Bu föy, laboratuvar ders saati içinde tamamlanmak üzere tasarlanmıştır. Eğer ders süresi içinde bitiremezseniz, **Google Classroom'da belirtilen son teslim tarihine kadar** tamamlayıp gönderebilirsiniz.

Arş. Gör. Z. Beyza Metin

Doç. Dr. MUHAMMET BAYKARA
Arş. Gör. Z. BEYZA METİN