Öğrenci No		Bölüm	Bilgisayar Mühendisliği		
Adı Soyadı		Yıl / Yarıyıl	2025-2026	☐ Güz	☒ Bahar
İmza		Tarih / Saat / Süre	30/03/2026	13.00	60 dakika
Dersin Kodu / Adı	BILM-202 ALGORİTMALAR	Sınav Türü	☒Ara Sınav	☐Final	☐Bütünleme
Dersin Sorumlusu	Dr. Öğr. Üyesi Sercan KÜLCÜ		☐Mazeret	☐Diğer:	

Öğrenme Çıktıları	ÖÇ1	ÖÇ2	ÖÇ3	Toplam
Sorular	**1-3**	**4-5**	**6-7**	7 Soru
Puanlar	40	30	30	100 Puan
Alınan Puan				

Sınava İlişkin Açıklamalar

- Sınavın ilk 30 dakikasında salondan çıkmak yasaktır.
- Sınavda cep telefonu, akıllı saat vb. kullanmak; kalem, silgi alışverişi yapmak yasaktır.
- Cevaplarınız adım adım, anlaşılır ve düzenli olmalıdır. Anlaşılamayan düzensiz cevaplar puanlanmayacaktır.
- Sınav kurallarına mutlaka uyunuz. Sınav kurallarına uymayanlar hakkında disiplin soruşturması açılacaktır.

SORULAR

Soru 1: Aşağıda verilen fonksiyonun zaman karmaşıklığını bulunuz ve kısaca açıklayınız. (10 puan)

```
public void foo(int n) {
    for (int i = 1; i < n; i = i * 2) {
        for (int j = 0; j < n; j++) {
            System.out.println(i + " " + j);
        }
    }
}
```

Bu algoritmanın zaman karmaşıklığı O(n log n)'dir. Dış döngüde i değişkeni her adımda iki katına çıkar (1, 2, 4, 8, …), yani iterasyon sayısı $\log_2 n$'dir. İç döngü ise her dış döngü adımında n kez çalışır. Bu nedenle toplam işlem sayısı yaklaşık olarak n × log n olur.

Toplam işlem sayısı: $T(n)=n\cdot\log_2 n$ şeklinde ifade edilebilir. Burada dikkat edilmesi gereken önemli nokta, büyüme hızının yalnızca iterasyon sayısıyla değil, iterasyonların artış biçimiyle belirlendiğidir. Lineer artış yerine üstel artış söz konusu olduğunda, toplam iterasyon sayısı dramatik şekilde azalır.

Bu tür yapılar, özellikle böl ve yönet (divide-and-conquer) algoritmalarında sıkça karşımıza çıkar. Logaritmik faktör, problemin her adımda küçültülmesinden kaynaklanır. Bu nedenle bu algoritma, saf lineer algoritmalara kıyasla daha ölçeklenebilir bir yapı sunar.

Soru 2: Aşağıda verilen fonksiyonun zaman karmaşıklığını bulunuz ve kısaca açıklayınız. (15 puan)

```
public void foo(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            System.out.println(i + " " + j);
        }
    }
}
```

Bu algoritmanın zaman karmaşıklığı $O(n^2)$'dir. Dış döngü n kez çalışırken, iç döngü her iterasyonda i'den başlayarak n'e kadar ilerler. Bu durum toplamda: $n+(n-1)+(n-2)+\cdots+1$ şeklinde bir işlem sayısı oluşturur. Bu toplamın kapalı formu: $n\,(n+1)\,/\,2$ olup, asimptotik olarak $O(n^2)$'ye karşılık gelir.

Bu yapı, klasik bir üçgensel (triangular) toplam örneğidir. Her ne kadar her adımda yapılan iş azalıyor gibi görünse de, toplam maliyet hâlâ karesel büyümektedir. Bu durum, algoritma analizinde önemli bir kavrayışı ortaya koyar: azalan iş yükü her zaman daha iyi karmaşıklık anlamına gelmez.

Bu tip algoritmalar, özellikle sıralama ve karşılaştırma tabanlı problemlerde karşımıza çıkar ve büyük veri

setlerinde performans darboğazı oluşturabilir.

Soru 3: Aşağıdaki kod parçasının yaptığı işi açıklayınız ve zaman karmaşıklığını bulunuz. (15 puan)

```
int count = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        count++;
    }
}
System.out.println(count);
```

Verilen kod parçası count değişkenini belirli bir sayıda artırmaktadır. Dış döngü i değerini 0'dan n-1'e kadar artırırken, iç döngü her i değeri için 0'dan i-1'e kadar çalışır. Bu durumda toplam işlem sayısı 0 + 1 + 2 + ... + (n-1) şeklinde olur. Bu toplam n(n-1)/2'ye eşittir. Bu ifade asimptotik olarak $O(n^2)$ büyür. Dolayısıyla algoritmanın zaman karmaşıklığı $O(n^2)$'dir. Bu kod aslında üçgensel bir artışla toplam işlem sayısını hesaplamaktadır. Bu tür yapılar, genellikle kümülatif maliyet analizinde karşımıza çıkar. Her iterasyonda yapılan iş artıyor olsa da toplam maliyet karesel büyür.

Soru 4: Hızlı sıralama (QuickSort) algoritmasında pivot seçiminin performansa etkisini açıklayınız. En iyi durum, En kötü durum, Ortalama durum kavramlarını bu algoritma üzerinden yorumlayınız. (15 puan)

QuickSort algoritmasında pivot seçimi, algoritmanın performansını doğrudan belirleyen kritik bir faktördür. Pivot, diziyi iki alt parçaya böler ve bu bölünmenin dengesi toplam çalışma süresini belirler.

En iyi durum (Best Case): Pivot diziyi iki eşit parçaya böler

→ Zaman karmaşıklığı: O(n log n)

En kötü durum (Worst Case): Pivot sürekli en küçük veya en büyük eleman seçilir

→ Zaman karmaşıklığı: $O(n^2)$

Ortalama durum (Average Case): Rastgele veya dengeli bölünmeler

→ Zaman karmaşıklığı: O(n log n)

Daha teorik açıdan, QuickSort'un beklenen karmaşıklığı: E[T(n)]=O(nlogn) olarak ifade edilir.

Bu durum, algoritma analizinde önemli bir kavramı ortaya koyar: deterministik yapı + rastgelelik = ortalama performans optimizasyonu. Bu nedenle pratikte pivot seçimi genellikle rastgele veya "median-of-three" gibi yöntemlerle yapılır.

Soru 5: Aşağıda verilen kod parçasının yaptığı işi açıklayınız. Zaman karmaşıklığını bulunuz. Dizi: [5, 2, 8, 1] için her adımda oluşan çıktıyı yazınız. (15 puan)

```
for (int i = 0; i < dizi.length - 1; i++) {
    for (int j = 0; j < dizi.length - i - 1; j++) {
        if (dizi[j] > dizi[j + 1]) {
            int tmp = dizi[j];
            dizi[j] = dizi[j + 1];
            dizi[j + 1] = tmp;
        }
    }
    System.out.println(Arrays.toString(dizi));
}
```

Verilen kod parçası Bubble Sort (Kabarcık Sıralama) algoritmasını gerçekleştirmektedir. Algoritma, komşu elemanları karşılaştırarak büyük olanı sağa doğru iteratif olarak taşır. Her dış döngü iterasyonunda, dizinin en büyük kalan elemanı sona yerleşir.

Verilen dizi için adımlar:

tur: [2, 5, 1, 8]

tur: [2, 1, 5, 8]

tur: [1, 2, 5, 8]

Zaman karmaşıklığı $O(n^2)$'dir çünkü iki iç içe döngü bulunmaktadır.

Bubble Sort'un temel problemi adaptif olmamasıdır (optimizasyon yapılmadığı sürece). Yani veri zaten büyük ölçüde sıralı olsa bile gereksiz karşılaştırmalar yapılır. Ancak erken durdurma (early termination) gibi iyileştirmelerle en iyi durum O(n)'e indirilebilir. Bu algoritma pratikte verimsiz olsa da, lokal swap tabanlı optimizasyonların anlaşılması açısından değeri yüksektir.

Soru 6: İkili arama (Binary Search) algoritmasını açıklayınız. Zaman karmaşıklığı nedir? Hangi koşul sağlanmazsa algoritma çalışmaz? Neden hızlıdır? (15 puan)

Binary Search (İkili Arama), sıralı veri yapılarında çalışan ve her adımda arama uzayını ikiye bölen bir algoritmadır. Her iterasyonda orta eleman kontrol edilir ve arama alanı yarıya indirilir.

Zaman karmaşıklığı: O(logn)

Bu performans, rekürsif bağıntı ile de ifade edilebilir: T(n)=T(n/2)+O(1)

Algoritmanın çalışabilmesi için temel koşul, veri yapısının sıralı olmasıdır. Aksi halde, orta elemana bakarak yön tayin etmek mümkün değildir ve algoritma anlamını yitirir.

Binary Search'ün hızının temelinde, her adımda yapılan işlem sayısından ziyade, problemin boyutunun üstel olarak küçültülmesi yatar. Bu nedenle büyük veri setlerinde lineer aramaya göre dramatik performans avantajı sağlar.

Soru 7: Aşağıda verilen kod parçasının yaptığı işi açıklayınız. Algoritma karmaşıklığını bulunuz. Dizi: [2, 3, 4, 7, 8, 9, 10, 21, 22, 31, 33, 37, 38, 39, 40, 42, 45] olsun. Aranan 40 olduğunda oluşacak ekran çıktısını yazınız. (15 puan)

```
int baslangic = 0;
int bitis = dizi.length - 1;
while (baslangic <= bitis) {
	int orta = baslangic + (bitis - baslangic) / 2;
	System.out.println(dizi[orta]);
	if (dizi[orta] == aranan) {
		return orta;
	} else if (dizi[orta] > aranan) {
		bitis = orta - 1;
	} else {
		baslangic = orta + 1;
	}
}
```

Verilen kod parçası Binary Search (İkili Arama) algoritmasını uygulamaktadır. Sıralı bir dizide aranan değeri bulmak için her adımda ortadaki elemanı kontrol eder ve arama alanını ikiye böler. Verilen dizi [2, 3, 4, 7, 8, 9, 10, 21, 22, 31, 33, 37, 38, 39, 40, 42, 45] ve aranan değer 40 için işlem adımları şu şekildedir:
İlk adımda orta eleman 22'dir ve ekrana yazdırılır.
İkinci adımda orta eleman 38'dir ve ekrana yazdırılır.
Üçüncü adımda orta eleman 40'tır ve ekrana yazdırılır ve aranan değer bulunur.
Ekran çıktısı sırasıyla 22, 38 ve 40 şeklindedir. Bulunan indeks 14'tür. Algoritmanın zaman karmaşıklığı O(log n)'dir çünkü her adımda arama alanı yarıya indirilmektedir. Bu algoritma, her adımda arama uzayını yarıya indirerek ilerler. Daha formel olarak: T(n)=T(n/2)+O(1)