GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ VİZE SINAVI

Adı – Soyadı – Numarası:

Soru 1: Aşağıda verilen kod parçasının zaman karmaşıklığı nedir? Kısaca açıklayınız.

```
void yerDegistir(int[] dizi, int i, int j) {
      int gecici = dizi[i]; // 1 işlem (atama)
      dizi[i] = dizi[j];  // 1 işlem (atama)
      dizi[j] = gecici;  // 1 işlem (atama)
}
```

Metot, bir dizideki iki elemanın yerini değiştirir. İçerisinde sabit sayıda işlem (3 atama işlemi) bulunmaktadır. Girdi boyutu (n) ne olursa olsun, işlem sayısı değişmez. Metodun çalışması dizinin boyutuna (n) bağlı değildir. Her çağrıldığında sabit sayıda işlem (3 işlem) yapar. Bu nedenle zaman karmaşıklığı O(1) olur.

Soru 2: Bir algoritmanın en iyi, en kötü ve ortalama durum analizleri ne anlama gelir? Örnek ile açıklayınız.

En iyi durum: Algoritmanın en hızlı çalıştığı senaryodur. Girdi verisi algoritma için en uygun şekilde düzenlenmiştir, böylece en az işlem yapılır. Kabarcık sıralama algoritmasında, dizi zaten sıralıysa, sadece bir geçiş yapılır ve O(n) karmaşıklığında çalışır.

En kötü durum: Algoritmanın en yavaş çalıştığı senaryodur. Girdi verisi algoritma için en kötü şekilde düzenlenmiştir, en fazla işlem yapılır. Kabarcık sıralama algoritmasında, dizi tersten sıralıysa, her elemanın her geçişte yer değiştirmesi gerekir. Bu durumda $O(n^2)$ karmaşıklık ortaya çıkar.

Ortalama durum: Algoritmanın rastgele sıralanmış verilere uygulandığında ortalama çalışma süresidir. Gerçek dünyada en sık karşılaşılan durumdur. Doğrusal arama algoritmasında, aranan eleman dizinin ortalarında bulunuyorsa, ortalama O(n/2) ≈ O(n) zaman alır.

Soru 3: Aşağıda verilen kod parçasının yaptığı işi açıklayınız. Algoritma karmaşıklığını bulunuz. Dizi [7, 3, 9, 4, 2] olsun, fonksiyon çalıştırıldığında oluşacak ekran çıktısını yazınız.

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

Bu kod, Selection Sort (Seçmeli Sıralama) algoritmasını kullanarak bir diziyi küçükten büyüğe sıralar.

Dış döngü (i), n-1 kez çalışır. İç döngü (j), n-1'den başlayarak her seferinde bir azalır. Her döngüde en küçük elemanı bulmak O(n) sürer. Toplam karmaşıklık: $O(n^2)$

Ekran çıktısı:

[2, 3, 9, 4, 7]

[2, 3, 9, 4, 7]

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 07.04.2025

[2, 3, 4, 9, 7]

[2, 3, 4, 7, 9]

Soru 4: Hash tablosunda arama algoritmasının en iyi durum ve en kötü durum karmaşıklığı nedir?

Hash tabloları (Hash Tables), veriyi anahtar-değer (key-value) çiftleri halinde saklayan ve hızlı erişim sağlayan veri yapılarıdır. Hash fonksiyonu, anahtarları belirli bir dizine (hash değeri) dönüştürerek doğrudan erişim sağlar.

En iyi durumda, hash fonksiyonu mükemmel çalışır ve hiçbir çakışma (collision) oluşmaz. Anahtarın hash değeri doğrudan doğru dizine yönlendirilir. Tek adımda erişim sağlanır. Bu nedenle karmaşıklık O(1) olur.

Hash fonksiyonu kötü çalışırsa veya çok fazla çakışma (collision) oluşursa, en kötü durumda arama işlemi O(n) zaman alabilir. Diziye zincirleme (chaining) yöntemiyle eklenen tüm elemanlar aynı indekse düşerse, bağlı liste gibi sırayla arama yapmak gerekir. Bu durumda lineer arama yapılacağı için karmaşıklık O(n) olur.

Soru 5: Dizinin elemanlarını birbiriyle karşılaştırmadan sıralayan bir algoritma öneriniz.

Counting Sort, belirli bir aralıktaki elemanlar için eleman sayısını sayarak sıralama yapan bir algoritmadır. Bu algoritma, karşılaştırma yapmadan sıralama işlemi gerçekleştirir. Dizi elemanlarını saymak için bir sayım dizisi oluşturulur. Elemanlar, sayım dizisi kullanılarak sıralanır. Zaman Karmaşıklığı: En İyi, En Kötü ve Ortalama Durum: O(n + k) (n: dizi eleman sayısı, k: en büyük elemanın değeri)

Radix Sort, sayıları basamaklarına ayırarak sıralama yapar ve her basamağa göre sıralama işlemi gerçekleştirir. Bu algoritma da karşılaştırma yapmadan sıralama sağlar. Dizi elemanları basamağa göre sıralanır. Genellikle Counting Sort gibi daha küçük sıralama algoritmasıyla yapılır. Sayının en düşük basamağından en yüksek basamağına kadar sıralama yapılır. Zaman Karmaşıklığı: En İyi, En Kötü ve Ortalama Durum: O(nk) (n: dizi eleman sayısı, k: maksimum basamak sayısı)

Soru 6: Aşağıdaki kod parçasının algoritma karmaşıklığını bulunuz, nedenini kısaca açıklayınız.

```
for (int i = 0; i < dizi.length - 1; i++) {
      for (int j = 0; j < dizi.length - 1 - i; j++) {
            if (dizi[j] > dizi[j + 1]) {
                  yerDegistir(dizi, j, j + 1);
            }
      }
}
```

Bu kod parçası, Bubble Sort (Kabarcık Sıralama) algoritmasını gerçekleştirerek verilen diziyi küçükten büyüğe sıralar.

Dış döngü, i değişkeniyle dizi.length - 1 kadar döner. İç döngü, her dış döngü turunda, dizi.length - 1 - i kadar çalışır. Yer değiştirme (swap) işlemi, sadece soldaki eleman sağdaki elemandan büyük ise gerçekleşir. Bu işlemin karmaşıklığı sabittir (O(1)).

Toplam adım sayısı=(n−1)+(n−2)+(n−3)+...+1 = O(n2)

Soru 7: Aşağıda verilen kod parçasının yaptığı işi açıklayınız. Algoritma karmaşıklığını bulunuz. Dizi: [2, 3, 4, 7, 8, 9, 10, 21, 22, 31] olsun. Aranan 40 olduğunda oluşacak ekran çıktısını yazınız.

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 07.04.2025

```
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;
	}
}
```

Bu kod parçası, ikili arama (binary search) algoritmasını kullanarak sıralı bir dizide verilen elemanı arar.

Binary search algoritması, her adımda diziyi yarıya böldüğü için karmaşıklığı O(log n)'dir. n: Dizi eleman sayısı.

Ekran çıktısı:

8 21 22 31