GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ TELAFİ SINAVI

Adı – Soyadı – Numarası:

Soru 1: Aşağıda verilen kod parçasının ne iş yaptığını, ekran çıktısını ve algoritma karmaşıklığını bulunuz, nedenini kısaca açıklayınız.

```
for (int i = 0; i < n; i += 2) {
      System.out.println(i);
}
```

Bu kod parçası, 0'dan başlayarak n'e kadar olan çift sayıları ekrana yazdırır. Döngü, i değişkeni 0'dan başlayıp n'den küçük olduğu sürece devam eder ve her adımda i'nin değeri 2 artar.

Döngü, n / 2 + 1 kez çalışır (eğer n çift ise) veya (n - 1) / 2 + 1 kez çalışır (eğer n tek ise). Ancak, karmaşıklık analizinde sabit terimler ve katsayılar göz ardı edildiğinden, algoritmanın zaman karmaşıklığı O(n) olarak ifade edilir. Bu, kodun çalışma süresinin, n'in büyüklüğü ile doğrusal olarak arttığı anlamına gelir.

Soru 2: Aşağıda verilen kod parçasının ne iş yaptığını, ekran çıktısını ve algoritma karmaşıklığını bulunuz, nedenini kısaca açıklayınız.

```
int[] dizi = {5, 7, 3, 4, 2, 8};
for (int i = 1; i < dizi.length; i++) {
      int anahtar = dizi[i];
      int j = i - 1;
      while (j >= 0 && dizi[j] > anahtar) {
            dizi[j + 1] = dizi[j];
            j = j - 1;
      }
      dizi[j + 1] = anahtar;
      System.out.println(Arrays.toString(dizi));
}
```

Bu kod parçası, verilen diziyi Araya Ekleyerek Sıralama (Insertion Sort) algoritmasını kullanarak sıralar. Algoritma, diziyi sıralı ve sıralanmamış olmak üzere iki bölüme ayırır. Her adımda, sıralanmamış bölümden bir eleman alınır ve sıralı bölümdeki doğru yerine yerleştirilir.

Ekran Çıktısı:
[5, 7, 3, 4, 2, 8]
[3, 5, 7, 4, 2, 8]
[3, 4, 5, 7, 2, 8]
[2, 3, 4, 5, 7, 8]
[2, 3, 4, 5, 7, 8]

Algoritma Karmaşıklığı:
En iyi durum: Dizi zaten sıralıysa, içteki while döngüsü hiç çalışmaz ve karmaşıklık O(n)'dir.
En kötü durum: Dizi ters sıralıysa, içteki while döngüsü her adımda en fazla sayıda karşılaştırma yapar ve karmaşıklık O(n^2)'dir.
Ortalama durum: O(n^2) olarak kabul edilir.

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 05.05.2025

Bu kod parçasında, dıştaki for döngüsü n-1 kez çalışır (n: dizinin uzunluğu). İçteki while döngüsünün kaç kez çalışacağı, dizinin durumuna bağlıdır. En kötü durumda, içteki döngü de n kez çalışabilir. Bu nedenle, toplam karmaşıklık O(n^2) olur.

Soru 3: Doğrusal Arama (Linear Search) ve İkili Arama (Binary Search) algoritmalarını avantaj ve dezavantajlarına göre karşılaştırınız.

Doğrusal Arama (Linear Search): Bir dizideki elemanları baştan sona doğru tek tek kontrol ederek aranan elemanı bulmaya çalışır.
Avantajları: Kullanımı ve uygulaması basittir. Sıralı veya sırasız dizilerde çalışabilir.
Dezavantajları: En kötü durumda (aranan eleman dizinin sonunda veya dizide yoksa) zaman karmaşıklığı O(n)'dir, yani dizi boyutuna bağlı olarak performansı düşer. Büyük dizilerde verimsizdir.
İkili Arama (Binary Search): Sıralı bir dizide aranan elemanı bulmak için kullanılır. Dizi sürekli olarak ikiye bölünerek arama yapılır.
Avantajları: Sıralı dizilerde çok daha hızlıdır. En kötü durumda bile zaman karmaşıklığı O(log n)'dir. Bu, büyük dizilerde bile çok verimli olduğu anlamına gelir.
Dezavantajları: Sadece sıralı dizilerde uygulanabilir. Sıralama işlemi ek bir maliyet getirebilir (eğer dizi başlangıçta sıralı değilse).

Soru 4: Aşağıdaki metodun ne iş yaptığını, ekran çıktısını ve algoritma karmaşıklığını bulunuz, nedenini kısaca açıklayınız.

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

Bu metot, iç içe döngüler kullanarak bir hesaplama yapar ve sonucu sayac değişkeninde saklar. Dıştaki döngü i = 1'den başlar ve her adımda i'yi 2 ile çarparak n'e kadar (n hariç) devam eder. Yani, i değerleri 1, 2, 4, 8, ... şeklinde ilerler.
İçteki döngü ise 0'dan i'ye kadar (i hariç) döner ve her adımda sayac'ı 1 artırır. sayac değişkeni, içteki döngünün toplam kaç kez çalıştığını tutar.

Ekran çıktısı, sayac değişkeninin değerine bağlıdır. sayac'ın değeri ise n'e bağlı olarak değişir.
n = 8 ise, dış döngü 3 kez çalışır (i = 1, 2, 4). İç döngü sırasıyla 1, 2, ve 4 kez çalışır. sayac = 1 + 2 + 4 = 7 olur. Ekran çıktısı: 7
n = 16 ise, dış döngü 4 kez çalışır (i = 1, 2, 4, 8). İç döngü sırasıyla 1, 2, 4, ve 8 kez çalışır. sayac = 1 + 2 + 4 + 8 = 15 olur. Ekran çıktısı: 15
Genel olarak, n = 2^k ise sayac = 2^k - 1 = n - 1 olur.

Dıştaki döngü log(n) kez çalışır. İçteki döngü ise dıştaki döngünün her adımında farklı sayıda (i) kez çalışır. İç döngünün çalışma sayısı, dış döngü değişkeni i'nin değerine bağlıdır ve i'nin değerleri geometrik olarak

artar (1, 2, 4, 8, ...). Toplam adım sayısı yaklaşık olarak 1 + 2 + 4 + 8 + ... + 2^k-1 = 2^k - 1 ≈ n - 1 olur. Bu nedenle, algoritmanın zaman karmaşıklığı O(n)'dir.

Soru 5: Aşağıdaki metodun ne iş yaptığını, ekran çıktısını ve algoritma karmaşıklığını bulunuz, nedenini kısaca açıklayınız.

```
void fonksiyon(int n) {
      int sonuc = 0;
      for (int i = 0; i < n; i++) {
         for (int j = i; j < n; j++) {
             sonuc += 1;
         }
      }
      System.out.println(sonuc);
}
```

Bu metot, iç içe döngüler kullanarak bir hesaplama yapar ve sonucu sonuc değişkeninde saklar. Dıştaki döngü, i = 0'dan n'e kadar (n hariç) döner.
İçteki döngü, j = i'den n'e kadar (n hariç) döner. Yani, içteki döngünün başlangıç değeri, dıştaki döngünün o anki değerine bağlıdır. İçteki döngü her çalıştığında, sonuc değişkeni 1 artırılır.
Sonuç olarak, sonuc değişkeni, içteki döngünün toplam kaç kez çalıştığını tutar. Bu da aslında aşağıdaki toplamı hesaplar: n + (n-1) + (n-2) + ... + 1

Metodun ekran çıktısı, sonuc değişkeninin değerine bağlıdır. sonuc'un değeri ise n'e bağlı olarak değişir.
n = 4 ise, içteki döngü toplam 4 + 3 + 2 + 1 = 10 kez çalışır. Ekran çıktısı: 10
n = 5 ise, içteki döngü toplam 5 + 4 + 3 + 2 + 1 = 15 kez çalışır. Ekran çıktısı: 15
Genel olarak, 1'den n'e kadar olan sayıların toplamı n * (n + 1) / 2 formülüyle bulunur.

Dıştaki döngü n kez çalışır.
İçteki döngü, dıştaki döngünün her adımında farklı sayıda çalışır (n'den 1'e kadar).
Toplam adım sayısı n + (n-1) + (n-2) + ... + 1 = n * (n + 1) / 2 = (n^2 + n) / 2 olur.
Karmaşıklık analizinde sabit katsayılar ve düşük dereceli terimler göz ardı edildiğinden, algoritmanın zaman karmaşıklığı O(n^2)'dir.