Ad-Soyadı: **No:**

Sivas Cumhuriyet Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü
2024-2025 Eğitim Öğretim Yılı Bahar Algoritma Analizi ve Tasarımı Course AAD321 Vize Sınavı 4

1- Bir kütüphane sisteminde yeni gelen kitaplar teker teker kaydedilmektedir.

Her kitap bir ISBN numarasına sahiptir ve kütüphane görevlisi bu numarayı, önceden sıralı olan bir listeye, listenin sıralı yapısını bozmadan uygun konuma ekler.

Liste, her eklemede baştan sona yeniden sıralanmaz. Ayrıca, her 10 kitapta bir, sistemdeki tüm ISBN numaraları arasından ortanca değer hesaplanarak raporlanır.

Bu senaryo için en uygun algoritma türü nedir? Açıklayınız. ISBN numaralarını sıralı şekilde listeleyen bir algoritma yazınız. Yazdığınız algoritmanın zaman verimliliğini t(n) olarak ifade ediniz ve bu fonksiyon için Big-O notasyonunu belirtiniz.

```
Liste_median(A[0, ..., n-1])
// Girdi : A uzunluğunda bir A dizisi
// Çıktı : Elemanları küçükten büyüye sıralanmış A
for i = 1 to n - 1
    v = A[i]
    j = i - 1
    while j >= 0 and A[j] > v
        A[j + 1] = A[j]
        j = j - 1
    A[j + 1] = v
mid_indis=⌊n/2⌋
return A[mid_indis]
```

i=0 olduğunda iç for loop n-1 kez çalışır.

i=1 olduğunda iç for loop n-2 kez çalışır.

….

i=n-1 olduğunda iç for loop 1 kez çalışır.

Böylece iç for loop'un toplam çalışma sayısı $1+2+\ldots+n-1=\frac{(n-1)(n)}{2}=\frac{n^2-n}{2}$, buda t(n) 'i verir.

Son olarak $t(n)=\frac{n^2-n}{2}$ 'in ait olduğu O kümesini bulalım.

$$t(n)=\frac{n^2-n}{2}=\frac{n^2}{2}-\frac{n}{2}\leq n^2+n^2=2n^2$$

yazabiliriz. Yani c sabitini 2 alırsak, 1'den büyük her n için t(n) fonksiyonunu $2n^2$'den küçük kalır. Böylece

$$t(n)\in O(n^2)$$

2- 53 ve 26 olmak üzere iki tam sayı veriliyor. Bu iki sayının çarpımını, çarpma (*) operatörünü kullanmadan, sadece toplama, bölme ve ikiyle çarpma işlemleri kullanarak hesaplayınız.

Bu işlem için uygun ve sistematik bir yöntem belirleyiniz. Seçtiğiniz yönteme göre gerekli tabloyu oluşturunuz. Daha sonra bu yöntemi uygulayan sözde kodu (pseudo code) yazınız.

n	m	c
53	26	0
26	52	+26
13	104	0
6	208	+104
3	416	0
1	832	+416
-	832+416+104+26 =2952	

```
RusKoyulusuCarpim (m, n)
//Girdi: m ve n tam sayıları
//Çıktı: m*n
toplam ← 0
While m ≠ 1 :
    if m mod 2 = 0
        m ← m / 2
    else
        m ← (m - 1) / 2
        toplam ← toplam + n
            n ← 2 * n
return toplam + n
```

3- Bir e-ticaret sitesinde ürün fiyatları kullanıcıya her zaman artan sırada gösterilmelidir. Ancak veritabanından ürün fiyatları karışık şekilde çekilmektedir. Sistem yöneticisi, bu fiyat listesini verimli bir şekilde sıralamak için bir algoritma kullanmak istemektedir.

Bu senaryoda Merge Sort algoritmasının neden uygun bir yöntem olacağını açıklayınız.

42	15	23	8	16	4

Öncelikle listeyi nasıl parçaladığınızı gösteriniz.

Son olarak Merge Sort algoritmasının zaman karmaşıklığını t(n) ile ifade ediniz ve büyük O gösterimini belirtiniz.

Merge Sort, her durumda (en iyi, en kötü, ortalama) zaman karmaşıklığı O(n log n) ile garantilenmiştir. Yani büyük veri setlerinde bile performansı stabil ve öngörülebilirdir.

```
Adım 1: Listeyi ikiye böl
Sol parça: [42, 15, 23]
Sağ parça: [8, 16, 4]
Adım 2: Sol parçayı tekrar böl
Sol-sol: [42]
Sol-sağ: [15, 23]
Adım 3: [15, 23] listesini böl
[15]
[23]
Adım 4: Sağ parçayı tekrar böl
Sağ-sol: [8]
Sağ-sağ: [16, 4]
Adım 5: [16, 4] listesini böl
[16]
[4]
3. Parçaları Birleştirerek Sıralama
[15] ve [23] birleştirilerek [15, 23] olur.
[42] ve [15, 23] birleştirilerek [15, 23, 42] olur.
[16] ve [4] birleştirilerek [4, 16] olur.
[8] ve [4, 16] birleştirilerek [4, 8, 16] olur.
Son olarak [15, 23, 42] ve [4, 8, 16] birleştirilerek tam sıralı liste elde
edilir:
[4, 8, 15, 16, 23, 42]
```

$$T(n) = 2 \cdot T(n/2) + n$$

$$T(n/2) = 2 \cdot T(n/4) + n/2$$

$$T(n) = 2 \cdot (2 \cdot T(n/4) + n/2) + n = 4 \cdot T(n/4) + 2n$$

$T(n/4) = 2 \cdot T(n/8) + n/4$ olduğundan

$$T(n) = 4 \cdot (2 \cdot T(n/8) + n/4) + 2n = 8 \cdot T(n/8) + 3n$$

$$T(n) = 2^i \cdot T(n/2^i) + i \cdot n$$

$$T(n) = 2^{\log_2 n} \cdot T(1) + n \cdot \log_2 n = n + n \cdot \log_2 n$$

$$\boldsymbol{T(n) \in O(n \log n)}$$

4- Böl ve Yönet (Divide and Conquer) stratejisini kullanarak, verilen bir dizideki en büyük elemanı bulan bir algoritma tasarlayınız.

(Yine sadece Böl ve Yönet yöntemi kullanarak ve özyinelemeli şekilde yapınız.)

Not 1: Başka yöntem kullanmayacaksınız.

Not 2: Yazacağınız algoritmalar tamamen özyinelemeli (recursive) olmalıdır.

```
maxBl_Bol_Yonet(A[0], ..., A[n-1])
// birden uzunluğunda bir A dizisi
// çıktı: A'nın en büyük elemanı
n ← A'nın uzunluğu
if n == 1
    return A[0]
birinci_yarı ← A[0], ..., A[⌊n/2⌋ - 1]
ikinci_yarı ← A[⌊n/2⌋], ..., A[n-1]
ilk_max ← maxBl_Bol_Yonet(birinci_yarı)
ikinci_max ← maxBl_Bol_Yonet(ikinci_yarı)
if ilk_max > ikinci_max
    return ilk_max
else
    return ikinci_max
```

5- Sıralı bir tamsayı A dizisi veriliyor. Bu dizi artan sırada (küçükten büyüğe) sıralanmıştır.Yapman gereken:Dizi yerinde (in-place) olacak şekilde, yani ek bir dizi kullanmadan, tekrar eden elemanları silmek.Elemanların sıralaması ilk görü-ndüğü haliyle aynı kalmalı.Fonksiyon, tekrarlar kaldırıldıktan sonra dizide kalan benzersiz elemanların sayısını (k) döndürmeli.

İşlem tamamlandıktan sonra A dizisinin ilk k elemanı yalnızca benzersiz elemanları içermeli.k'-den sonraki elemanların değeri önemli değil ve dikkate alınmayabilir.

Örnek 1:

Girdi:A= [1, 1, 2]

Beklenen Çıktı:k = 2, A= [1, 2, _]

Örnek 2:

Girdi:A= [0, 0, 1, 1, 1, 2, 2, 3, 3, 4]

Beklenen Çıktı:k = 5, A= [0, 1, 2, 3, 4, _, _, _, _, _]

Not: k'den sonraki elemanların ne olduğunun önemi yoktur ve kontrol edilmez.

Not 2: Azalt ve Yönet (Decrease and Conquer) ya da Kaba Kuvvet (Brute Force) yöntemleri dışında başka hiçbir yöntem kullanamazsınız.

```
RemoveDuplicates(A[0, ..., n-1]):
    if n == 0:
        return 0
    k ← 1
    for i = 1 to n-1:
        if A[i] ≠ A[k - 1]:
            A[k] ← A[i]
            k ← k + 1
    return k
```

6- Bir Armstrong sayısı, basamaklarının küplerinin toplamı kendisine eşit olan sayıdır (3 basamaklı sayılar için klasik tanım).

Örneğin: 153 → $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$

Not: Kaba Kuvvet (Brute Force) yöntemi dışında başka hiçbir yöntem kullanamazsınız.

```
IsArmstrongNumber(n):
//Girdi:n Bir pozitif tamsayı. Bu sayı, Armstrong sayısı olup olmadığını kontrol
etmek istediğiniz sayıdır.
//Çıktı:Eğer n bir Armstrong sayısı ise (yani, her basamağının küpünün toplamı
kendisine eşitse), True döndürülür.
//Eğer n bir Armstrong sayısı değilse, False döndürülür.
    original_number ← n
    sum_of_cubes ← 0
    while n > 0:
        digit ← n mod 10
        sum_of_cubes ← sum_of_cubes + digit^3
        n ← n / 10  // Son basamağı çıkar
    if sum_of_cubes == original_number:
        return True
    else:
        return False  // Sayı Armstrong sayısı değil
```

7- Asimptot nedir, Big O, Big Ω ve Big Θ'yı anlamlarını açıklayınız, bu notasyonların birbirlerinden farkları nelerdir?

Asimptot: Bir fonksiyon çok büyük değerler aldığında ($n \to \infty$), nasıl davrandığını gösteren sınırdır. Algoritmanın büyüme hızını anlamamıza yarar.

Big O Notasyonu – $O(g(n))$

Anlamı: En fazla ne kadar büyüyebileceğini gösterir (üst sınır).

Yorum: "t(n) en kötü durumda g(n) kadar büyür."

Formal Tanım:

$t(n) \in O(g(n)) \Leftrightarrow \exists\ c > 0$ ve $\exists\ n_0 > 0$ öyle ki,

$t(n) \leq c \cdot g(n)$ her $n \geq n_0$ için geçerlidir.

Örnek: $t(n) = 3n^2 + 2n + 1 \to O(n^2)$

Big Omega Notasyonu – $\Omega(g(n))$

Anlamı: En az ne kadar büyüyebileceğini gösterir (alt sınır).

Yorum: "t(n) en iyi durumda en az g(n) kadar büyür."

Formal Tanım:

$t(n) \in \Omega(g(n)) \Leftrightarrow \exists\ c > 0$ ve $\exists\ n_0 > 0$ öyle ki,

$t(n) \geq c \cdot g(n)$ her $n \geq n_0$ için geçerlidir.

Örnek: $t(n) = 3n^2 + 2n + 1 \to \Omega(n^2)$

Big Theta Notasyonu – $\Theta(g(n))$

Anlamı: Fonksiyonun hem alt hem de üst sınırını gösterir.

Yorum: "t(n) hem en kötü hem de en iyi durumda g(n) kadar büyür."

Formal Tanım:

$t(n) \in \Theta(g(n)) \Leftrightarrow \exists\ c_1, c_2 > 0$ ve $\exists\ n_0 > 0$ öyle ki,

$c_1 \cdot g(n) \leq t(n) \leq c_2 \cdot g(n)$ her $n \geq n_0$ için geçerlidir.

Örnek: $t(n) = 3n^2 + 2n + 1 \to \Theta(n^2)$

Süre 75dk Başarılar dilerim...

Eng: Abdulrahman Hamdi