1.Bir liste nasıl bir gurup posta kutusuna benzer.

Cevap = 1- Bir liste, bir grup posta kutusuyla karşılaştırılabilir çünkü her ikisi de

öğeleri depolamak ve düzenlemek için bir araç görevi görür. Ayrıca,

içeriklerine kolay erişim ve erişim sağlayan ve içeriklerindeki değişiklikleri

karşılamak için gerektiğinde değiştirilebilen (eklenebilen, kaldırılabilen,

yeniden düzenlenebilen) benzersiz tanımlayıcılara (adresler veya dizinler)

sahiptirler.

2.Bir listenin bir öğesine erişmek nasıl olur da bir posta kutusunun içeriğini

almak gibi DEĞİLDİR?

Cevap = Bir listenin bir öğesine erişmek, bir posta kutusunun içeriğini almaya

benzemez çünkü bir listenin bir öğesine eriştiğinizde, yalnızca listedeki

konumuna atıfta bulunursunuz, oysa bir posta kutusunun içeriğini almak,

onu fiziksel olarak açmayı içerir. kutuyu ve içindekileri çıkarmak.

Ek olarak, bir listenin bir öğesine eriştiğinizde, dilimleme veya diğer

teknikleri kullanarak birden çok öğeyi aynı anda alabilirsiniz, oysa bir posta

kutusunun içeriğini almak aynı anda yalnızca bir öğeye erişmenizi sağlar.

Son olarak, bir posta kutusunun içeriğini aldığınızda, içindekiler kutusu

açılır ve yalnızca bir kez erişilebilir, oysa bir listenin bir öğesine

eriştiğinizde, öğe listede kalır ve tekrar tekrar erişilebilir veya değiştirilebilir.

3.Python kullanarak bir bilgisayarda bir işlemi tamamlamak için gereken süreyi

nasıl hesaplayabilirsiniz?

Cevap = Python'da bir işlemi tamamlamak için gereken süreyi hesaplamak için, zamanı

ölçmek için işlevler sağlayan time modülünü kullanabilirsiniz. İşte basit adımlar:

İşlemin başında, zaman fonksiyonunu kullanarak mevcut zamanı kaydedin.

Zamanlamak istediğiniz işlemi gerçekleştirin.

İşlemin sonunda, zaman fonksiyonunu kullanarak şimdiki zamanı tekrar kaydedin.

İşlem için geçen süreyi elde etmek için başlangıç saatini bitiş zamanından çıkarın.

Yüksek hassasiyetli zamanlama için time.perf\_counter() işlevini veya işlem

tarafından kullanılan CPU süresini ölçmek için time.process\_time() işlevini

kullanabilirsiniz. Geçen süreyi ölçtüğünüzde, sonucu ileride kullanmak üzere

yazdırabilir veya saklayabilirsiniz. Bu teknik, kodu optimize etmek veya farklı

algoritmaların verimliliğini karşılaştırmak için yararlıdır

4.Hesaplama karmaşıklığı açısından hangisi daha iyidir, O(n2) olan bir algoritma mı yoksa O(2n) olan bir algoritma mı?

Cevap = O(n^2) olan bir algoritma, O(2^n) olan bir algoritmadan daha iyidir çünkü

ilki, girdi boyutuyla yönetilebilir bir oranda büyüyen bir polinom zaman

karmaşıklığına sahipken, ikincisi üstel bir zaman karmaşıklığına sahiptir.

ılımlı giriş boyutları için bile pratik olmayabilir.

5. Bir algoritmanın O(n2) olmasının ne anlama geldiğini İngilizce olarak açıklayınız.

Cevap = O(n^2) olan bir algoritma, zaman karmaşıklığının girdi boyutunun (n)

karesiyle orantılı bir oranda büyüdüğü anlamına gelir. Bu, girdi boyutu

arttıkça, sorunu çözmek için gereken işlem sayısının n^2 ile orantılı bir

oranda arttığı anlamına gelir.

Örneğin, n elemanlı bir diziyi sıralamak için zaman karmaşıklığı O(n^2)

olan bir algoritmamız varsa, dizinin boyutu iki katına çıktıkça diziyi

sıralamak için gereken işlem sayısı bir artacaktır. 4 çarpanı (2^2).

Pratik açıdan, O(n^2) zaman karmaşıklığına sahip bir algoritmanın büyük

girdi boyutlarına sahip sorunları çözmesi uzun zaman alabilir ve belirli türde

gerçek dünya uygulamaları için uygun olmayabilir

6.Tümevarım yoluyla bir ispat yaparken, ispatın hangi iki bölümü vardır?

Cevap = genel ifade (genel durum) ve özel durum (özel ifade) bölümleri vardır.

Genel ifade, tüm olası durumlar için geçerli olan bir ifadedir. Özel durum ise, belirli bir örnek için doğruluğu ispatlanan bir ifadedir.

7.İlk seferde n adım, ikinci seferde n - 2 adım, sonraki seferde n - 4 adım

yürüten ve son seferde 2 adım yürütene kadar tekrar eden bir döngüye sahip

bir algoritmanız olsaydı, bu döngünün karmaşıklık ölçüsü ne olurdu?

Cevap = Bu döngüde toplam adım sayısı, n + (n-2) + (n-4) + ... + 2 olacaktır.

Bu toplam, aritmetik serinin toplamına benzer bir şekilde hesaplanabilir:

n + (n-2) + (n-4) + ... + 2 = [n + 2 + (n-2) + 4 + ... + n] / 2

= [n/2 \* (n/2 + 1) + 2/2 \* (2/2 + 1)] / 2

= [n^2/4 + n/2 + 1] / 2

= O(n^2)

Bu nedenle, bu döngünün karmaşıklık ölçüsü O(n^2)'dir.

8.Elinizde n boyutunda bir veri kümesi ve bu veri kümesini aynı şekilde işleyen

iki algoritma olduğunu varsayalım. Algoritma A, veri setindeki her bir öğeyi

işlemek için 10 adım attı. Algoritma B ise her bir öğeyi 100 adımda işledi. Bu

iki algoritmanın karmaşıklığı ne olurdu?

Cevap = Her iki algoritma da n öğeyi işlediğinden, Algoritma A'nın toplam adım sayısı 10n, Algoritma B'nin toplam adım sayısı ise 100n'dir. Bu nedenle, algoritma A'nın karmaşıklığı O(n), algoritma B'nin karmaşıklığı ise O(n) olarak ifade edilir. Ancak, algoritma B'nin her bir öğeyi işlemek için Algoritma A'ya göre 10 kat daha fazla adım attığı göz önüne alındığında, pratikte algoritma A, algoritma B'ye göre daha verimlidir.

9.Bir liste üzerinde append işleminin neden + operatöründen daha verimli olduğunu

açıklayınız.

Cevap = Append() metodu, tek bir eleman eklemek için kullanılabilir. Bu nedenle, eleman eklemek istediğimiz listenin sonuna tek bir eleman eklemek için, + operatörüne göre daha hızlıdır.

Append() metodu, listeyi değiştirirken, + operatörü, iki listenin kopyalarını oluşturup birleştirir. Bu nedenle, büyük listelerle çalışırken, + operatörü çok fazla bellek kullanır ve zaman alır.

10. Bir listede belirli bir değeri bulmak için bir algoritma tanımlayın. Daha sonra

Bu algoritmanın hesaplama karmaşıklığı. İstediğiniz varsayımları

yapabilirsiniz, ancak varsayımlarınızı algoritmanızla birlikte belirtmelisiniz.

Cevap = Belirli bir değeri bulmak için bir algoritma aşağıdaki gibi tanımlanabilir:

Listenin ilk elemanından başlayarak, her elemanın hedef değere eşit olup olmadığını kontrol edin.

Eğer eleman hedef değere eşitse, o elemanı döndürün ve işlemi sonlandırın.

Eğer eleman hedef değere eşit değilse, bir sonraki elemana geçin ve adımları 1 ve 2'ye tekrarlayın.

Tüm elemanlar kontrol edildiğinde, hedef değer listeye dahil edilmediyse, None (veya -1 gibi bir işaretleyici değer) döndürün.

Varsayımlar:

.Liste önceden sıralanmamıştır.

.Liste elemanları sayısal veya karakter değerlerinden oluşabilir.

.Listenin n elemanı olduğunu varsayalım.

.Her elemanın hedef değere eşit olma olasılığı eşit kabul edilebilir.

Bu algoritmanın hesaplama karmaşıklığı, en kötü durumda O(n) olacaktır. Bu, listedeki tüm elemanların hedef değere eşit olmadığı ve son elemana kadar tüm elemanların kontrol edildiği durumu ifade eder. Ancak, en iyi durumda, hedef değer ilk eleman olabilir ve bu durumda hesaplama karmaşıklığı O(1) olacaktır. Ortalama durumda, hedef değer listenin ortalarında bulunabilir ve hesaplama karmaşıklığı O(n/2) olarak hesaplanabilir.