**Tek Yönlü Doğrusal Bağlı Liste**

-Dizi yok, düğümler vardır. Düğümler birbirini takip ederek listeyi oluşturur.

-Düğüm içinde data ve pointer vardır. Pointer, bir sonraki düğümdeki verinin adresini gösterir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

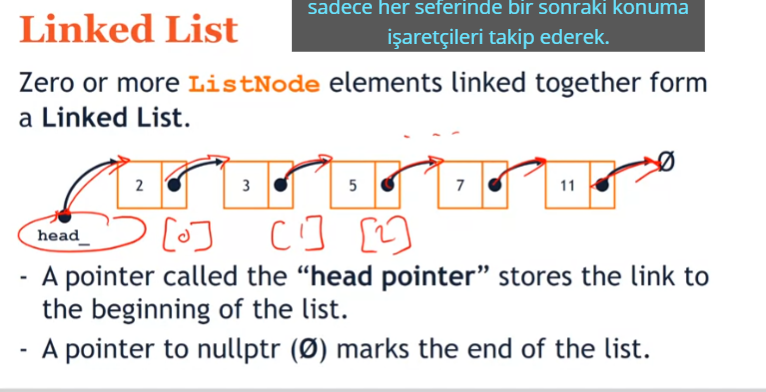
Açıklama otomatik olarak oluşturuldu

-Dizide adresler ardışıktır, araya bir şey giremez. Ama listede düğümler farklı yerlerde olabilir. Düğümler birbirini göstererek bir liste oluşturur ve son eleman null’ı gösterir. Null ile liste sonuna geldiğimizi anlarız.

metin, mobilya, tablo, çalışma masası içeren bir resim

Açıklama otomatik olarak oluşturuldu

-Tek yönlü bir hareket olduğu için tüm düğümlere ulaşabileceğimiz bir adres tutmalıyız. Bu yüzden listenin en başındaki elemanın adresini tutan bir pointer vardır (=head). Bu pointer gerçek listemizdeki eleman değildir sadece listenin başlangıcını kaybetmemek için tutarız.



-Yeni düğüm oluşturma işlemi heap’te yapılmalı. Yoksa fonksiyon kapatıldığı an bellekten silinir bu da stack bellek bölgesidir. Biz istediğimiz zaman listeyi silmeliyiz.

**NOT:**

**Tek Yönlü Dairesel Bağlı Liste**

-Tek fark son elemanın null’ı değil ilk düğümü göstermesidir.

-son düğüm head’i gösteremez çünkü head liste elemanlarına dahil değildir.

metin, beyaz tahta içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Çift Yönlü Doğrusal Bağlı Liste**

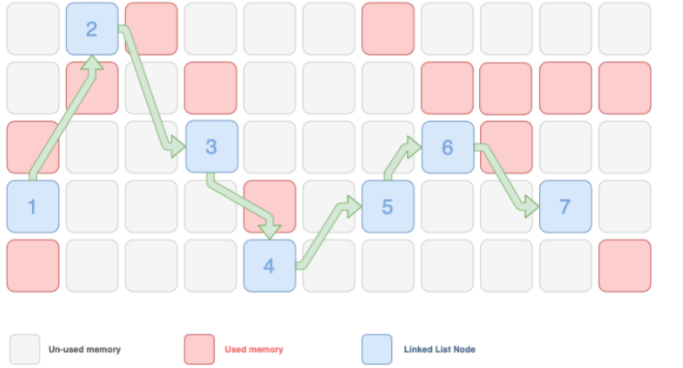
(ileride)

**Çift Yönlü dairesel Bağlı Liste**

(ileride)

**Array and List**

-Linked list bellekte farklı yerlerde bulunan düğümlerden oluşur. Diziler birbirini takip eden adreslerden oluşur.



-Linked list’de indexleme yoktur bu yüzden anlık erişim yoktur. Dizilerdeki elemanlar ise arka arkaya olan adreslerde saklandığından dolayı indexleme mantığıyla çalışır ve anlık erişim vardır. Dizi[10] dediğinde 10. indexe ulaşabilirsin. Ayrıca ilk elemanın adresi üzerinden (dizi adı ilk eleman adresini tutan pointerdır) istediğimiz elemana gitmek için (dizi+i) mantığını uygulayabilir. Bu mantık listelerde çalışmaz. Listelerde istenilen elemana ulaşmak için en baştan okumaya başlayıp istediğimiz elemana gideriz.

-Dizilerde elemana erişim hızlı O(1), listelerde yavaş O(n)

-Dizilerde ekleme, silme yavaş çalışır. Listelerde bu işlemler daha hızlıdır

-Diziler sabit boyutludur ama liste dinamiktir.

-Dizilerde araya eleman eklemek/çıkarmak çok maliyetlidir. Elemanları kaydırmamız gerekir, yeterli alan yoksa diziyi yeniden boyutlandırmamız, elemanları kopyalamamız gerekir. Sürekli bu işlem yapılacaksa liste tercih edilmelidir. Elemanların listede kaymasına gerek yok sadece göstilen yer (next) değişir

-Listeler de diziler gibi tek tip veri içerir bu açıdan benzerdir. Neden? Çünkü pointerı belli bir türü tutacak şekilde belirliyoruz.

**Run Time Analysis**

-Dizide bütün elemanlara erişme süresi aynıdır. Birinci elemana da erişmeye çalışsak on milyonuncuya da erişmeye çalışsak aynı süre geçer çünkü indexleme var -> anlık erişim. Listede ise en baştan başlayarak isteğimiz elemana erişene kadar dolaşmamız gerekir.

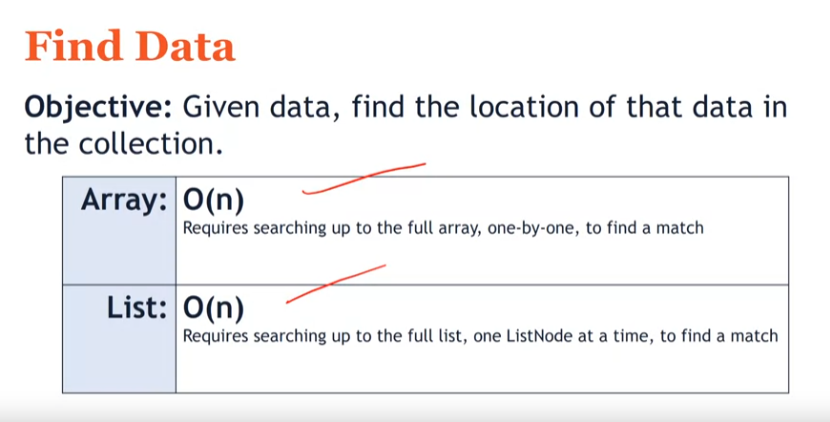
-n eleman için n. elemana erişme süresi Dizide: O(1), Listede: O(n)

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Array and List Operations**

**Find Data**



Veriyi bulmak ikisinde de O(n) notasyonundadır çünkü eleman sayısı kadar gider.

Peki daha kısa bir arama/eleman bulma yapabilir miyiz?

**Array**

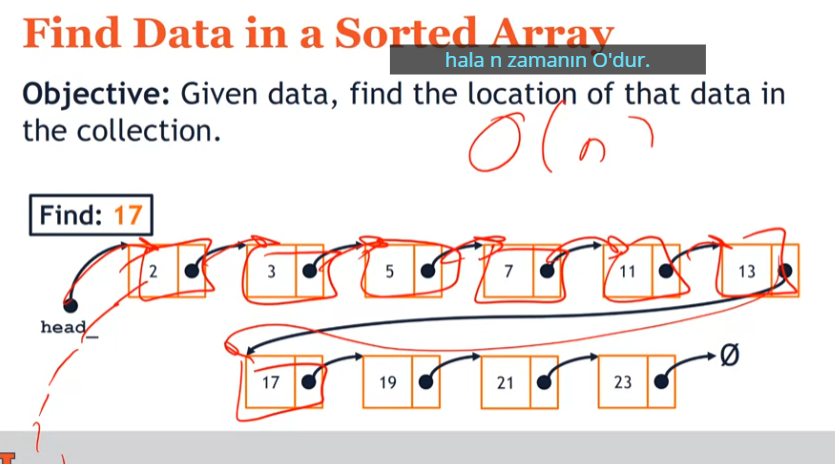
metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Sıralı dizide eleman bulmak işlemleri daha da hızlandırır. Dizide ikili arama yapısı ile her seferinde elemanları yarıya indiriyoruz. O(n) arama süresi birden O(logn)’e iner

Diyelim ki 17 yi bulmak istiyoruz. En ortaya bakarız: 13. 13<17 o zaman sağa bakacağız. Oranın da en ortasına bakarız: 19. 19>17 o zaman solda. Ve 17 yi bulduk

**List**



Yine sıralanmış bir dizimiz var. Ama listenin ortasına atlayabileceğimiz bir yapı yok. Elimmizde sadece head var o sa en başı gösteriyor. Bu yüzden doğrusal olarak tek tek listeyi gezmeliyiz. Listede hala veri bulma O(n) düzeyde olur

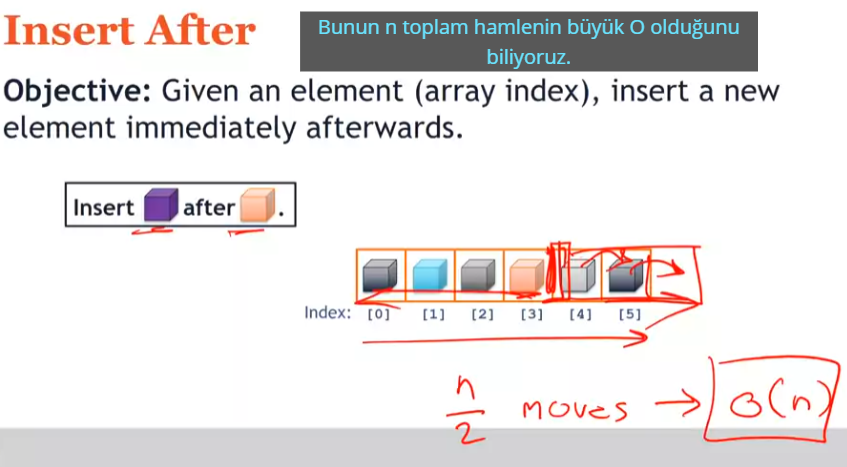
Sonuç olarak:

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

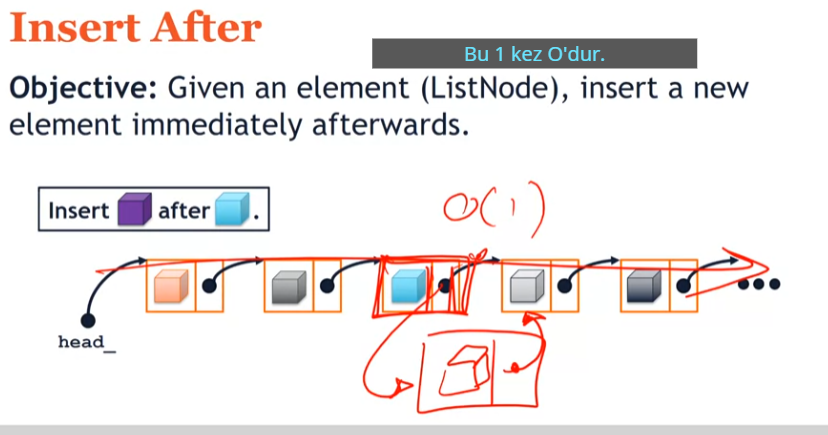
**Insert After**

**Array**



Elemanları kaydırmamız gerekir. Belki de dizişe bos yer yoktur ve yeniden boyutlandırma yapmamız gerekir. Gereken süre O(n) dir.

**List**



Elemanları kaydırmamıza ya da kopyalamamıza gerek yok. O(1)

Sonuç olarak:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Delete After**

(silme ve ekleme işlemleri benzer işlemlerdir)

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Liste Avantajları**

-Az memory israfı (Liste dinamik olduğundan gereksiz bellek kullanımının önüne geçer. İstediğimiz zaman eleman ekleriz, istediğimiz zaman belleği serbest bırakırız)

-Dimanik boyutlu kullanılması

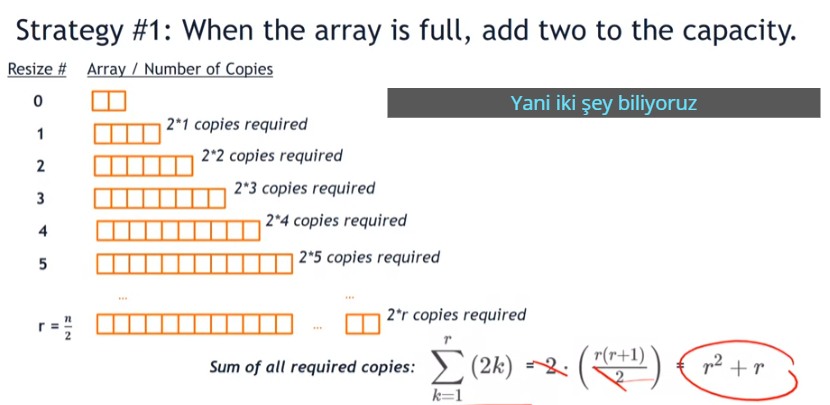
-Ekleme/Silme işlemleri

**Liste Dezavantajları**

-Anlık erişim yok (random Access e izin vermez.)

-Her eleman sonraki düğüme gösteren pointer’a sahiptir bu yüzden fazladan bellek kullanır. Diziler zaten sıralı tutulduğundan buna gerek yok.

**NOT:** Diziye eleman eklerken farklı yaklaşımlar kullanabilirsin. Kullanılan yaklaşımlar çalışma performansı etkiler

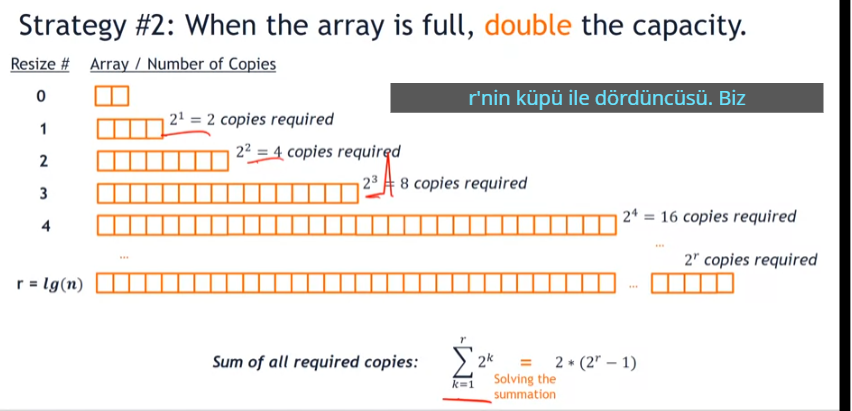


2 boyutlu bir dizi var, doldu 2 elemanımız var. Yeniden boyutlandırmak istediğimizde 4 boyutlu dizi tanımlıcaz ve 2 elemanın da kopyasını alacağız. Her seferinde boyutlandırma 2 artacak şekilde yapılır. Bu şekilde büyücek ve gerekli kopyaların tümü r^2 + r kadar olacak. (N veri sayısı, r kaçıncı basamakta olduğu)

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

R umurumuzda değil. Big o notasyonunda toplam zaman, elamanın karesi kadar olacak. (en büyük terimi aldığımız için kalan yerleri görmezden geldik)



Bu strateji de ise her seferinde kapasiteyi iki katına çıkarıyoruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kapasiteyi iki katına çıkardığımızda gereken süre O(n) olur. Yani algoritma tasarımının programın çalışma süresinde büyük etkisi vardır.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

!!!! bu yüzden diziyi yeniden boyutlandırırken kapasite sürekli x2 yapılır. amorti analizine göre en iyi sonuç bu şekilde alınır

**NOT:**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**NOT:** Bazen head dışında listenin tail adresi de tutulur. Bu durumda son elemana ulaşma O(n) zaman yerine O(1) zaman alır. Liste başındaki elemana ulaşmak her zaman O(1) dir. Ama aradaki elemanlaırn yerlerini bilemeyeceğimiz için her zaman O(n) zaman alır.