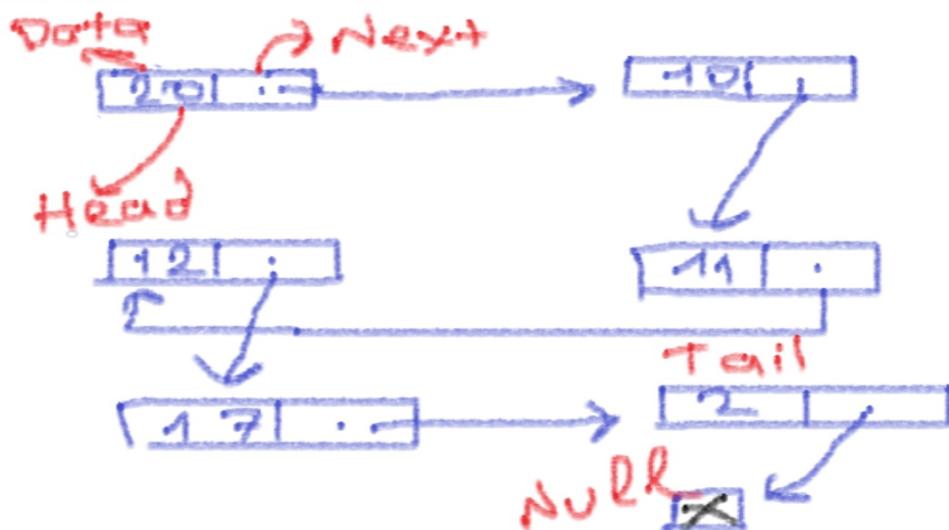


## Arrays

- Arrays (Diziler), anlam ifade etmesi için birden fazla nesneye ihtiyaç duyabilir. Mesela, Şu an karşısında olduğunuz bilgisayar örneğini inceleyelim. Masaüstü bilgisayarlar, klavye-mouse-monitör üçlüsünü bir araya getirince anlam ifade eder. Herhangi biri olmadan bir işlem yapmanız olasıdır ama zorludur.
- Array (Dizi), dezavantajlarından biri olan hafıza problemini inceleyelim. Bilgisayar örneğimizden devam edelim. Hali hazırda bir klavye, bir mouse ve bir monitörümüz var. Yeni bir monitör aldığımızda daha büyük bir masaya ihtiyacımız var. Aynı şekilde yeni bir klavye veya mouse aldığımızda da aynı durum geçerli. Bir yerden bir yere taşıırken zaman ve güç kaybına uğruyoruz.
- Dynamic Arrays (Dinamik diziler) ise yeni bir eleman için boşta yer tutmasından ötürü esnektiler. Örneğin, bazı mutfağın masaları açılan sürgülü bir yapıya sahiptir. Masanın küçük kaldığı durumlarda büyütmek için kullanılır. Dinamik dizilerde aynı mantık sahiptir. Yeni elemanlar için yer tutarlar.
- Dynamic Array (Dinamik dizinin) dezavantajlarından biri ise hafızada fazladan yer kaplaması, gerçekleşecek olan bir diğer olayı engelleyebilir. Nasıl mı, hemen örnek ile kavrayalım. Masa örneğinden bahsetmiştim. Misafirleriniz bir işi çıkması durumunda fazladan yer kapladık ve hareket kabiliyetimizi kaybettik.
- Hep dezavantajlarını konuşuyorsun, e yahu bunun avantajı yok mu? Tabii ki var. Array'lerin birbirine bağlı olması ulaşılabilirliğini kolaylaştırır. Klavye-Mouse-Monitör örneğini vermiştık. Hepsi bir masada bulununca ulaşılması kolaydır. (Masa = Array, Klavye-Mouse-Monitör = Array Elemanı)

## Linked-List

- Linked-List (Bağlı listeler), yan yana zorunluluğu olmadan veri tutmamızı sağlayan yapılardır. Yeni gelen eleman için hafızada yeni bir alan açmamız gerekmek. Array'dan farklı olarak evet elemanlar hafıza içerisinde dağınık olabilir, fakat son gelen eleman kendinden bir önceki elemana adresini bildirmek zorundadır.



- Yukarıdaki örnekte gördüğünüz üzere, her bir düğüm bir sonrakiniin adresini tutar. Her bir önceki eleman bir sonraki eleman ile bağlıdır.

Array	Linked-List
Array'in herhangi bir elemanına ulaşmak aynı sürede gerçekleşir (Random Access).	Linked-List'de ulaşmak istediğimiz elemana gidebilmek için birbirine bağlı olan elemanları ziyaret etmemiz gerekiyor.
Array'ler, sadece eleman tuttuğu için hafızada daha az yer kaplarlar.	Linked-List'ler, eleman ile birlikte adres tuttuğundan dolayı hafızada daha fazla yer kaplarlar.
Daha çok statik (durağan) durumlarda daha fazla performans gösterir.	Ekleme, çıkarmadan fazla olduğu durumlarda linked-list daha fazla performans gösterir.

## Array

- Arrayin istediğimiz elemanına sabit sürede erişebiliyoruz
- Sadece elemanı tuttuğumuz için daha az yer kaplıyor
- Memory locality için iyi

## Linked List

- \* Eleman eklemek ve silmek arraylere göre daha kolay
- Elemanların bir blok olarak tutulması gerekmiyor, hafızada blok olarak yer yoksa da kullanılabilir

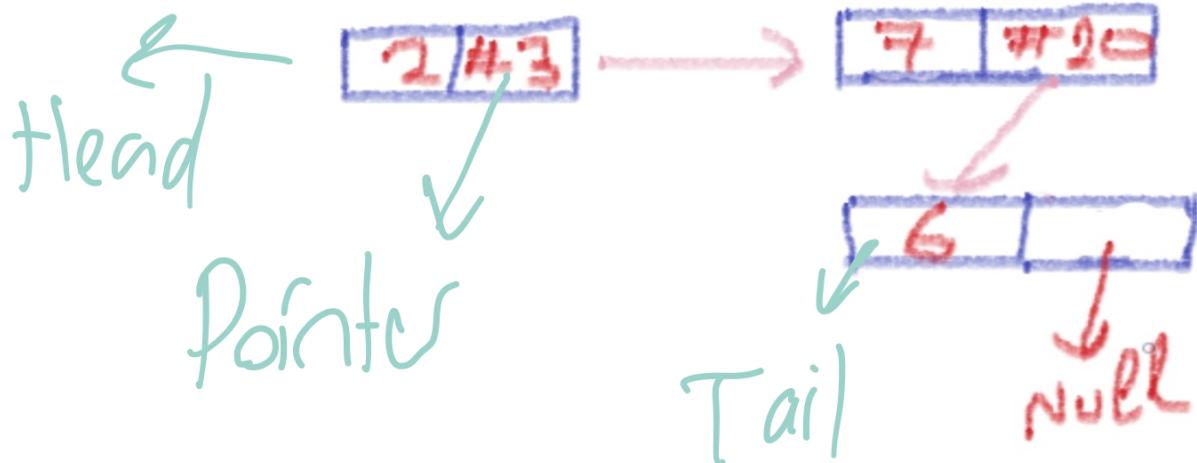
# Linked Liste Eleman Ekleme :

## Eleman Ekleme/Çıkarma

Gelin 3 elemanlı bir hücre oluşturalım.



2,7,6,

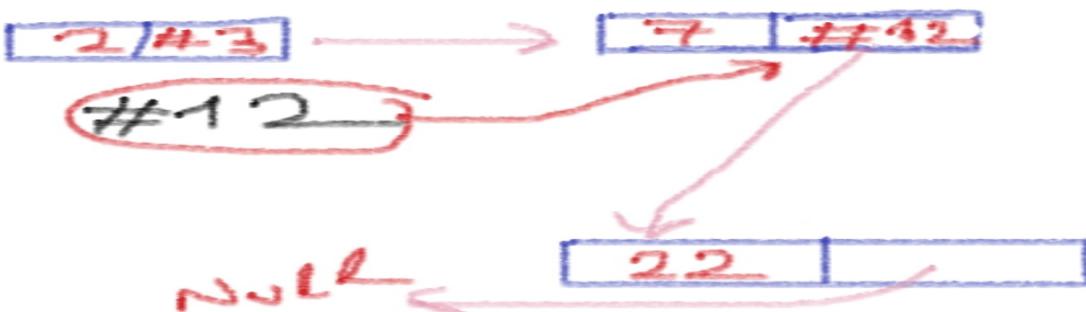


## Eleman Çıkarma



- Adresi #20 olan 6 numaralı hücreni çıkarmak istiyoruz. Linked-List'de bir önceki eleman adresini tutuyordu. Yani 7 numaralı hücrede bulunan 6'nın hücre adresini siliyoruz. Yerine 22 numaralı hücrenin adresini yazıyoruz.

2,7,6,22 → #12



# Stack LIFO: Last in first out

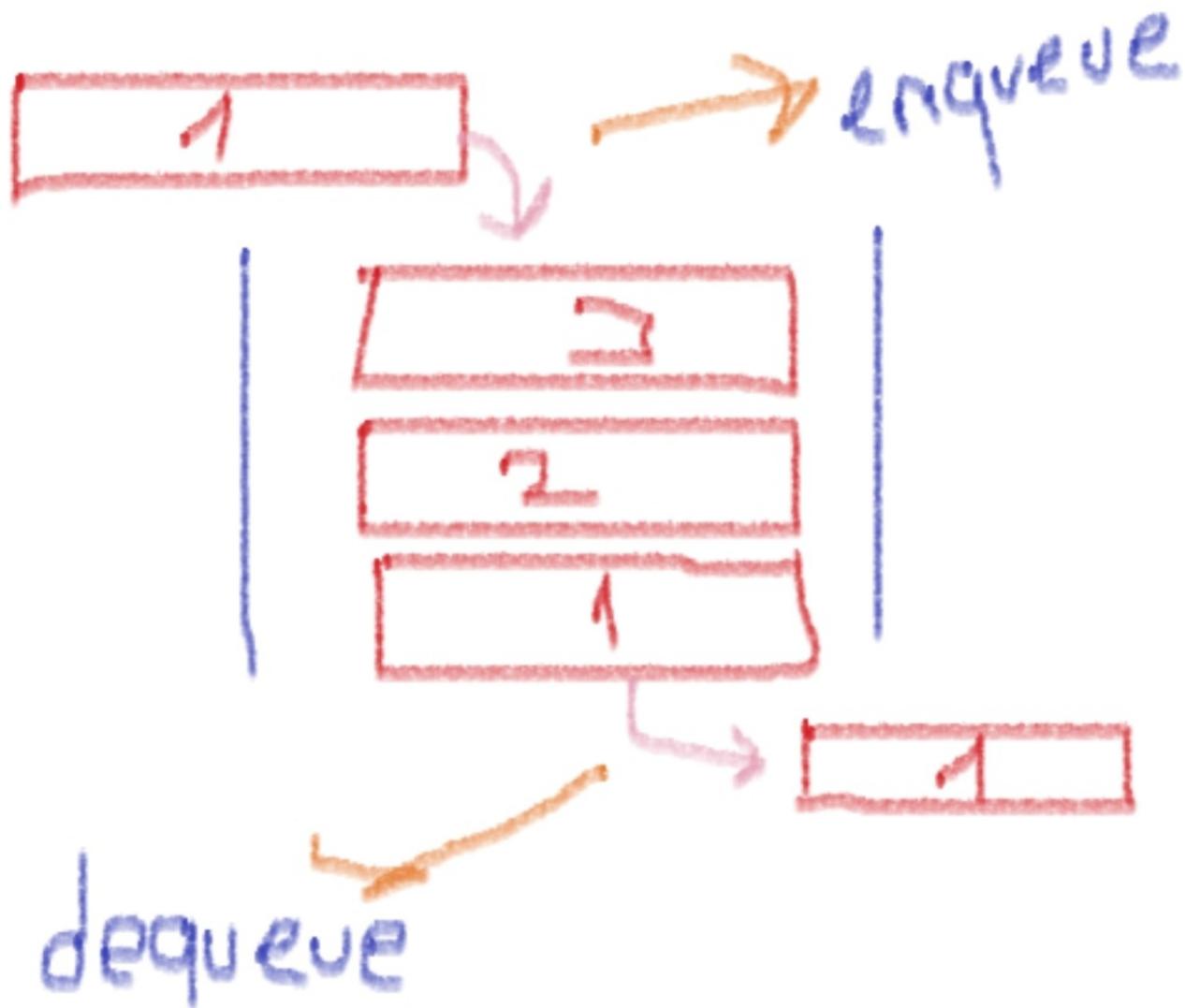
## Stack

- Stack, LIFO (Last in First out) (En son giren en önce çıkar) mantığına dayanan, elemanlar topluluğundan oluşan bir yapıdır. Gelin hemen örneğimize geçelim. Taşınırken topladığınız koli kutusu düşünün. İçerisinde kitaplar var ve en, boy olarak koliye tam olarak koyuluyor. Mantıken kolinin altı kapalı ve üst üste koyma gerekmektedir. Yeni taşındığınız yerde çıkartırken en üstekinden başlarsınız. İşte stack (Yığın) da aynı mantıkta çalışıyor.
- Yığınlara eleman eklerken veya çıkartırken bazı methodlar uygulanır. Bunlardan biri push, diğer ise pop. Push, yığının üzerine eleman eklemek için kullanılır (Koliye kitap koymak). Pop ise, yığından eleman çıkarmak için kullanılır.



# Queue: Fifo: first in first out.

- Queue (Kuyruk), FIFO (First in First out) (İlk giren ilk çıkar) prensibine dayanan, girişlerde ve çıkışlarda belirli bir kurala göre çalışan yapıdır. Stack de verdigimiz örneği kuyruğa göre uyarlayalım. Biz örnekte altı kapalı bir koli kutusunu düşünmüşük. Şimdi o koli kutusunun altı yırtılmış. Sonuç olarak ne oluyor? İlk giren ilk çıkmış oluyor.
- Queue (Kuyruk)'da eleman eklemesi yaparken enqueue methodunu kullanıyoruz. Eleman silerken ise dequeue methodunu kullanıyoruz.

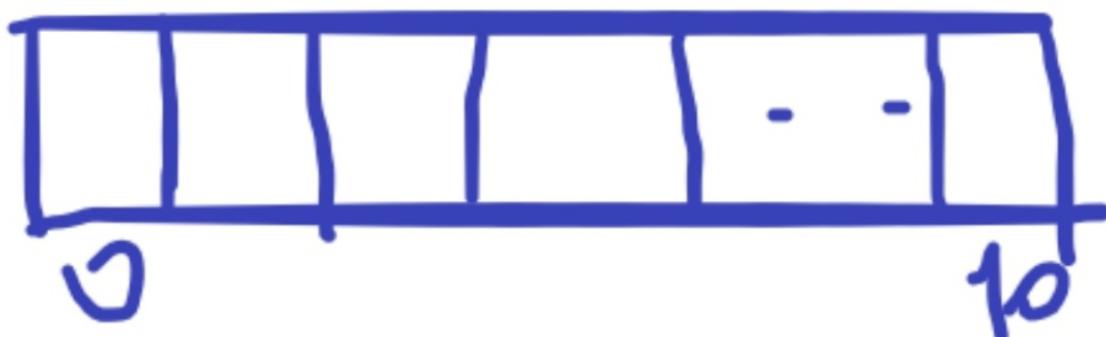


# Hash Table: Key value prensibi

## Hash Function/ Hash Table

Hash Table, key value prensibine dayanan bir array kümeleridir. Key olarak çağrırdığınız elemanın değerini (value) yansıtır.

- Hash Table yerine dizileri kullanabilirdik. Fakat her ürünü ve fiyatını tek tek aramak istemediğimiz için hash table kullanıyoruz. Peki bu süreç nasıl işliyor? Hemen bir örnek yapalım. Örneğimiz bir kuru yemiş dükkanından gelecek.



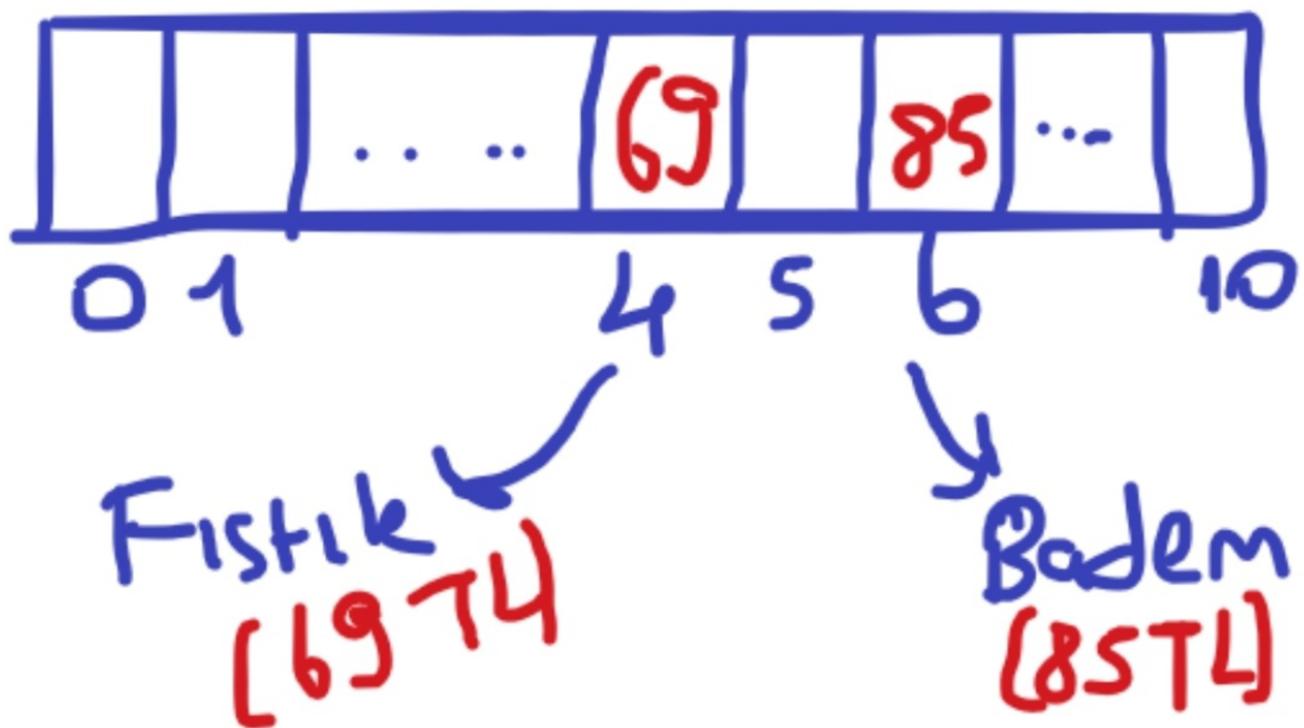
"Fıstık" →  → 4

"Badem" →  → 6

! HashTable = Hashfunction + array

Bu kısımda ilk olarak bulunan ürün sayımız kadar değeri olan bir Array oluşturduk.

Daha sonra hash fonksiyonundan ürünlerini geçirerek index değerlerine ulaştık.



Hash function aynı keyler için aynı olmalı ve farklı keyler için farklı output vermelidir.

• Hash function'ın output range'si array'in boyutunda olmalıdır.

Collision: farklı çıktılar ama aynı sonuçlar.

Yeni farklı inputlar aynı outputu verebilir.

## Hash Collision

Hash Function farklı iki değerden aynı sayı üretilirse bu duruma Collision (çarpışma) denir. Bu olay istediğimiz bir durum değildir.

- Hash Function'lar bazen farklı durumlar için farklı sonuçlar üretmemeyebilir. Örnek olarak araçları bir hash function dan geçirelim. Bu fonksiyonumuz son harflerine göre bir değer atıyor. Örneğin, motor ve tır için aynı değerleri ataması collision'a neden oluyor.
- Collision sorunuyla az karşılaşabilmek için kaliteli bir hash function olmalı. Bu sayede verimli bir Hash Table elde etmiş oluyoruz.
- Çarpışma sayısı arttıkça aradığımız şeyi bulma hızı azalır.

## Hashtable

↳ Hashfunction → Collision 97  
olmali ve  
hic olmasal.



# Algoritmalar Analizi :

Yukarıda bir önceki (rate of growth) ile yapılır.  $\rightarrow$  Ceng 185

We use Ram Model.

## Time Complexity:

Worst Case - Best Case

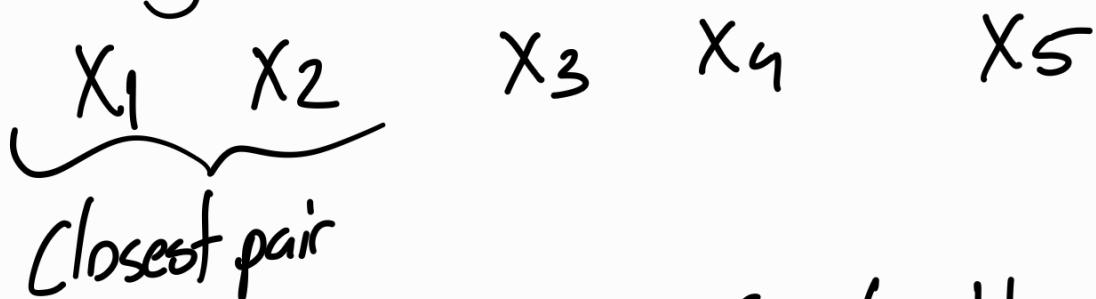
Average  
Case

↓  
ideal

Big O Notation "ex"  $O(n)$   $O(\log n)$   
 $x=1 : O(n)$   $n^2+2 : O(n^2)$

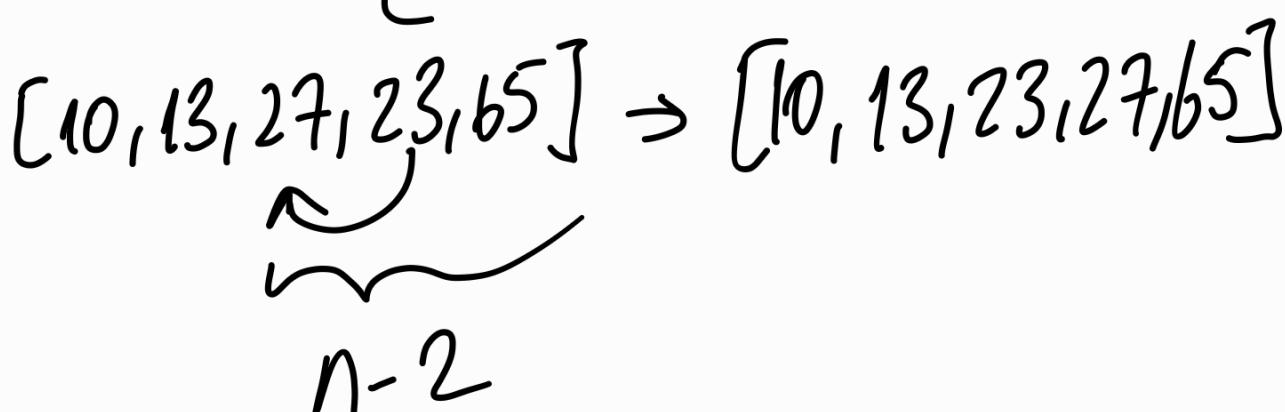
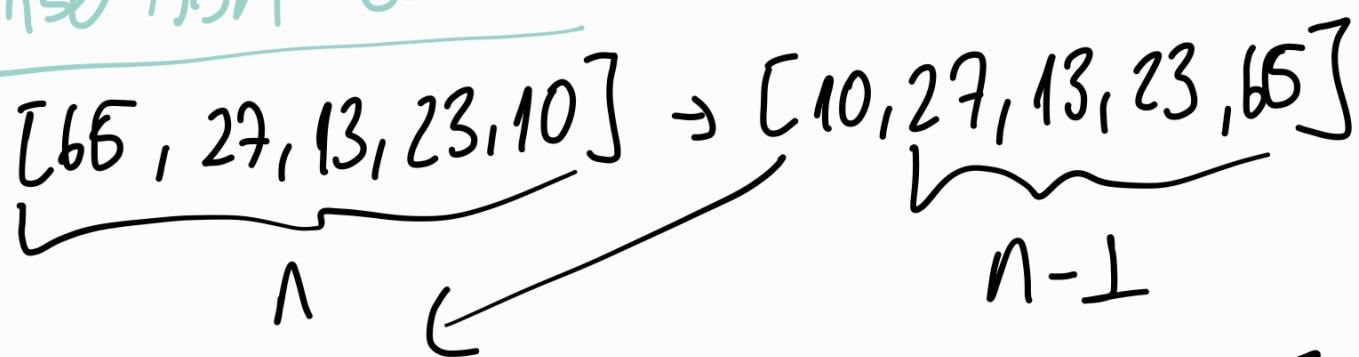
# SORTING (sıralama) Algoritmalar

Sorting :: Elemların en basiton sıralanmak



Aynı eleman kontrolü : Sıradıighter sonra  
aynı elemlar yan yana düşer.

## Insertion Sort

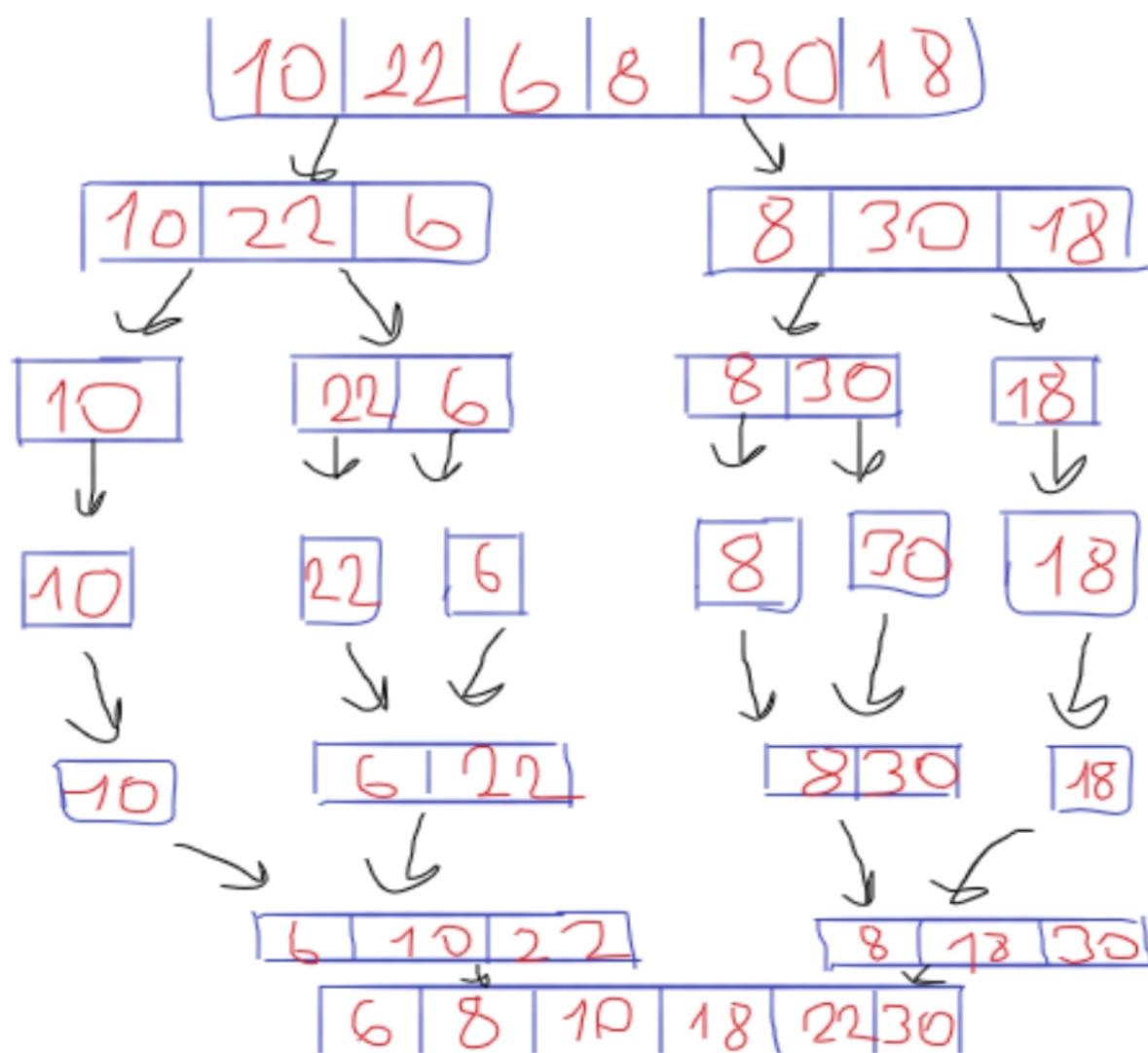


Toplam :  $1 + 2 + 3 + \dots + n$

$$\begin{aligned}
 &= \frac{n \cdot n + 1}{2} = \frac{n^2 + n}{2} : \text{Big(O)} = n^2
 \end{aligned}$$

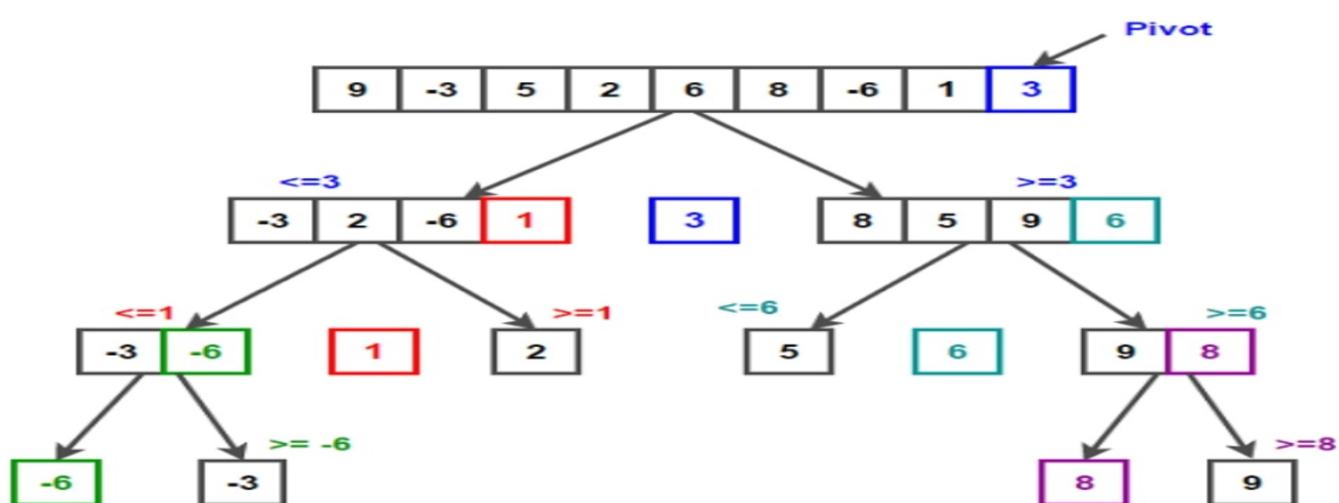
# Merge Sort:

$O(n \log n)$



Quick Sort merge sort'dan daha hızlı.

Hızlı sıralama günümüzde çok yaygın olarak kullanılan bir sıralama algoritmasıdır. N tane sayıyı average case e göre big-o  $n \log n$ , worst case e göre big-o  $n^2$  karmaşıklığı ile sıralanır.



# SEARCHING ALGORITHMS:

## Sıralama

Searching : istedigim elementi arayip bulup getirme.

### Linear Search:

#### Linear Search

Linear search, tek tek elemanları dolandıktan sonra istediğim elemanın olup olmadığına bakmaktadır.

- Örneğin, [20,25,46,48] veri setini ele alalım. Benim aradığım eleman 25. İlk elemana gidiyorum ve değeri 20 sen değilsin diyorum. İkinci elemana gidiyorum ve değeri 25 evet sensin diyorum. Linear search algoritmam burada bitmiş oluyor.
- Big-o ya göre incelediğimizde bizim worst case'ımız neydi? Elemanın dizinin sonunda bulunmasıydı. Bu sebepten ötürü n elemanımız varsa big-o notasyonumuz otomatik olarak n oluyor.

Worst case :  $O(n)$  Best case :  $O(1)$

Binary Search : Once listeyi sıralamak gereklidir.

kitti aranır

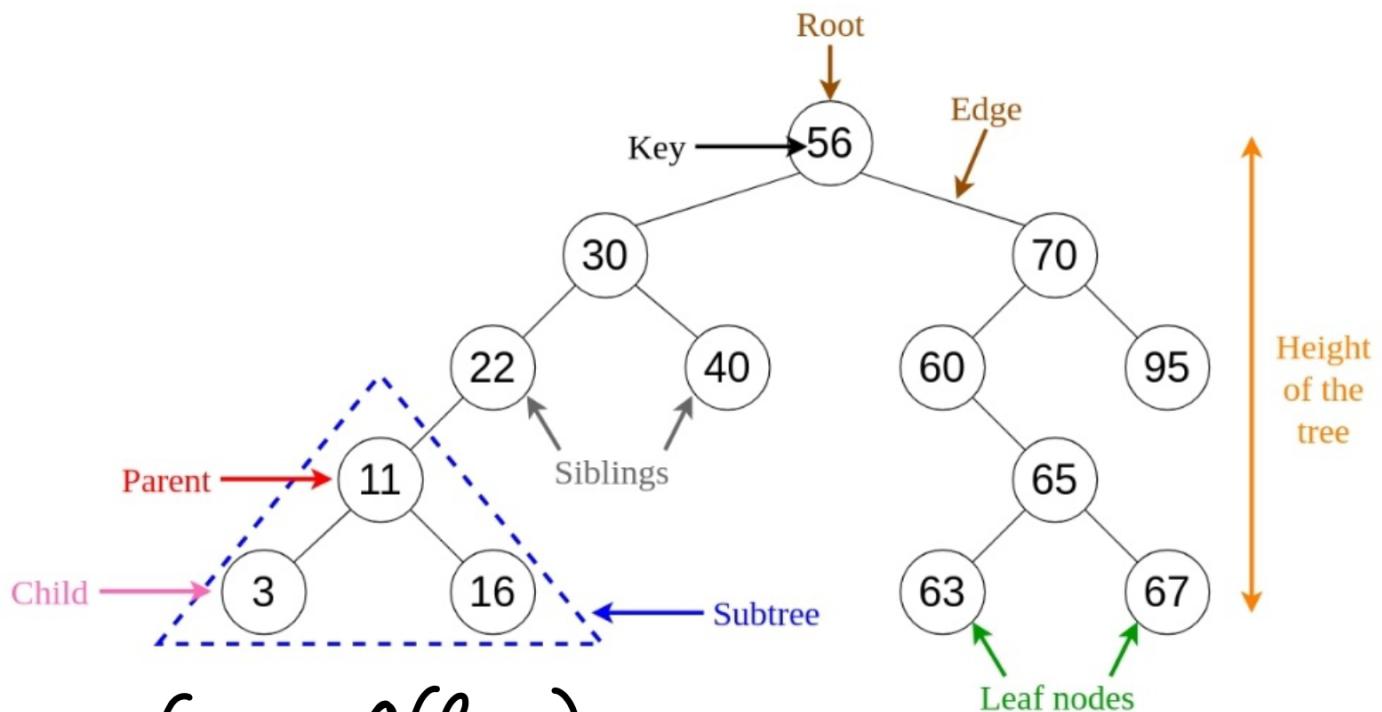
10, 20, 30, 40, 50

15 16 17 18 19

$$n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8$$

$$2^k = n \rightarrow O(n) = O(\log n)$$

# Binary Search Tree :



Average Case:  $O(\log n)$

Worst case:  $O(n)$

Random access yok.

Reference: Patika.dev