**O Notasyonu (Büyük O Notasyonu):**

* **Tanım:** Bir algoritmanın çalışma zamanının, asimptotik olarak

En kötü durumu temsil eder. Yani, algoritmanın maksimum çalışma zamanını ifade eder.

* **Örnek:** Bir algoritmanın zaman karmaşıklığı O(n^2) ise, bu algoritmanın çalışma zamanı n'nin karesi ile orantılıdır.

**Ω Notasyonu (Büyük Omega Notasyonu):**

* **Tanım:** Bir algoritmanın çalışma zamanının, asimptotik olarak

En iyi durumu temsil eder. Yani, algoritmanın minimum çalışma zamanını ifade eder.

* **Örnek:** Bir algoritmanın zaman karmaşıklığı Ω(n) ise, bu algoritmanın çalışma zamanı en azından n ile doğru orantılıdır.

**Θ Notasyonu (Theta Notasyonu):**

* **Tanım:** Bir algoritmanın çalışma zamanının, asimptotik olarak

En iyi ve en kötü durumları arasında bir sınırlama koyar. Yani, bir algoritmanın çalışma zamanının sabit bir sınırlar aralığında olduğunu ifade eder.

* **Örnek:** Bir algoritmanın zaman karmaşıklığı Θ(n) ise, bu algoritmanın çalışma zamanı n ile doğru orantılıdır ve bu orantı en iyi ve en kötü durumlar için de geçerlidir.

**Lineer Search(Doğrusal Arama):**

**Dizinin ilk elemanından başlayarak adım adım arar.**

Worst case = **O(n)**

Average case = **O(n)**

**Binary Search(İkili Arama):**

* Sıralı bir dizide orta eleman seçilir.
* Aranan eleman, orta elemandan küçükse, sol tarafı kontrol ederiz.
* Aranan eleman, orta elemandan büyükse, sağ tarafı kontrol ederiz.
* Aranan eleman, orta elemana eşitse, eleman bulunmuştur.
* Aranan eleman bulunana kadar ya da alt dizinin boyutu sıfır olana kadar bu adımları tekrarlarız.

Worst case = **O(logn)**

Average case = **O(logn)**

**Insertion(Ekleme) Sort Algoritması:**

* Dizinin ilk elemanı, sıralı kabul edilir.
* Dizi elemanları bir bir karşılaştırılarak sıralı konumlarına yerleştirilir.

Best case=**O(n)**

Worst case = **O(n^2)**

Average case = **O(n^2)**

[**Insertion Sort için Youtube link:**](https://www.youtube.com/watch?v=NnMSqZgvruM&list=PLh9ECzBB8tJPTWIUbZjHZMMGuZcpHUv5h&index=4)

**Merge(Birleştirme) Sort Algoritması:**

* Verilen diziyi iki eşit parçaya böler.
* Her iki parçayı ayrı ayrı sıralar.
* İki sıralı parçayı birleştirerek tam sıralı diziyi oluşturur.
* Parçalama ve birleştirme işlemleri rekürsif olarak devam eder.
* Parçalanan diziler, tek elemanlı hale gelene kadar bölünür.
* Ardından, birleştirme işlemi başlar.

Best case=**O(n logn)**

Worst case = **O(n logn)**

Average case=**O(n logn)**

[**Merge Sort için Youtube link:**](https://www.youtube.com/watch?v=f9CNp_uuNJg&list=PLh9ECzBB8tJPTWIUbZjHZMMGuZcpHUv5h&index=10)

**Heap(Yığın) Sort Algoritması:**

**Diziye “heapify” işlemi uygulanır. İsteğe göre en küçük veya en büyük eleman başa gelir ve diziden silinip yeni dizinin ilk elemanı olur. silindikten sonra başa gelen ikinci eleman olur ve bu böyle devam eder.**

[**Heap Sort için Youtube Link:**](https://www.youtube.com/watch?v=JS-5PA8hFA4&t=943s)

**Quick(Çabuk) Sort Algoritması:**

**Bir pivot seçilir ardından bu pivottan büyük olanlar pivotun sağ tarafına küçük olanlar ise sol tarafına yazılır. Sonra ayrılan bu sağ ve soldaki diziler kendi aralarında aynı şekilde parçalanarak sıralanır.**

Worst case = **O(n2 )** Average case **= O(n logn)**

[**Quick Sort için Youtube link:**](https://www.youtube.com/watch?v=aubOM9dOy6c&list=PLh9ECzBB8tJPTWIUbZjHZMMGuZcpHUv5h&index=12)

**Counting(Sayarak) Sort Algoritması:**

Her elamandan kaç tane olduğunu sayar ve başka bir diziye kaydeder. Sonra sıralanmış olur. Başka bir dizi kullanıldığı için bellekte ekstra yer kaplamakla beraber doğrusal bir yapıya sahiptir yani karmaşıklığı O(n) olur.

**[Counting Sort için Youtube Link:](https://www.youtube.com/watch?v=fkX5Zy1qL7Y&t=97s)**

**Radix(Taban) Sort Algoritması:**

1. **En Düşük Basamaktan Başlayarak Sıralama:**

Elemanlar, en düşük basamaktan başlayarak sıralanır. Örneğin, bir sayının birler basamağından başlanarak sıralama yapılabilir.

1. **Her Basamak İçin Sıralama ve Gruplandırma:**

Elemanlar, her bir basamak için sıralanır ve gruplandırılır. Önce birler basamağına göre sıralama yapılır, ardından onlar basamağına göre sıralama yapılır, bu işlem en yüksek basamağa kadar devam eder.

1. **Gruplandırılan Elemanları Birleştirme:**

Her basamaktaki sıralama işlemi tamamlandıktan sonra, elemanlar birleştirilir ve tam sıralanmış dizi elde edilir.

Örnek olarak, aşağıdaki sayıları ele alalım:

Dizi: [170, 45, 75, 90, 802, 24, 2, 66]

1. **Birler Basamağına Göre Sıralama:**
   * [170, 90, 802, 2, 24, 45, 75, 66]
2. **Onlar Basamağına Göre Sıralama:**
   * [802, 2, 24, 45, 66, 170, 75, 90]
3. **Yüzler Basamağına Göre Sıralama:**
   * [2, 24, 45, 66, 75, 90, 170, 802]

[Radix Sort için Youtube Link:](https://www.youtube.com/watch?v=M1hABszC9gU&t=427s)

**Bucket(Kova) Sort Algoritması:**

1. **Elemanların Kovalara Dağıtılması:**

Elemanlar, belirli kural veya fonksiyonlar kullanılarak kovalara yerleştirilir.

Elemanlar, kova sıralamasının etkin çalışabilmesi için uniform bir şekilde dağılmış olmalıdır.

1. **Her Kovanın İçindeki Elemanların Sıralanması:**

Her bir kova içindeki elemanlar, genellikle başka bir sıralama algoritması kullanılarak sıralanır.

Örneğin, her kova içindeki elemanlar için bir sıralama algoritması uygulanabilir.

1. **Kovaların Birleştirilmesi:**

Sıralanmış kovalar, birleştirilerek tam sıralı dizi elde edilir.

Worst case = **O(n2 )** Average case **= O(n)**

[Bucket Sort için Youtube Link:](https://www.youtube.com/watch?v=04FK4njWrBc)

**The Master Method (Ana Metod):**

*T*(*n*)=*a*⋅*T*(*b/n*​)+*f*(*n*) ana formülünde *a*≥1 ve *b*>1 sabitleri, *f*(*n*) ise bir fonksiyondur.

Burada bulunan a=alt problem sayısını, b=alt problem boyutunu f(n)=doğrusal zamanda birleşmeyi temsil eder.

Burada üç durum vardır;

1. **F(n)< *nlogba ise T(n) = nlogba  olur,***

**Örnek 1:**

**T(n)=9T(n/3)+n bu soruda a=9,b=3 ve f(n)=n dir. Buna göre çözümleme yapalım.**

**nlogba=nlog39 olur yani log3^9 2 olduğu için toplam çıkan sonuç n^2 olur. N^2 > f(n) olduğu için T(n) = n^2 olur.**

**2.F(n)= *nlogba ise T(n)= F(n).logn olur,***

**Örnek 2:**

**T(n)=2T(n/2)+n bu soruda a=2,b=2 ve f(n)=n dir. Buna göre çözümleme yapalım.**

**nlogba=nlog22 olur yani log2^2 1 olduğu için toplam çıkan sonuç n olur. N = f(n) olduğu için T(n) = f(n).logn yani nlogn olur.**

**3.F(n)> *nlogba ise T(n)=F(n ) olur.***

**Örnek 3:**

**T(n)=3T(n/4)+nlogn bu soruda a=3,b=4 ve f(n)=nlogn dir. Buna göre çözümleme yapalım.**

**nlogba=nlog43 olur yani log4^3 < f(n) olduğu için T(n) = nlogn olur.**

[**Master Teorem için Youtube link:**](https://www.youtube.com/watch?v=yfIbsFfOTt8&list=PLh9ECzBB8tJPTWIUbZjHZMMGuZcpHUv5h&index=13)

**Hashing:**

Bir verinin belli bir fonksiyona göre başka bir diziye kaydedilmesi. Yani şifreleme gibidir. Çakışma durumları için bazı metotlar vardır;

* **Ayrık zincirleme (Separate Chaining):**

Çakışmaların yönetimi için kullanılan bir hashing yöntemidir. Bu yöntemde, aynı hash değerine sahip elemanlar ayrı bağlı listelerde saklanır.

Örnek: Ayrık Zincirleme ile Hashing;

Dizi=[12, 35, 27, 40, 55, 20, 44, 31]

Hash Fonksiyonu: **hash(x) = x % 10 bu dizinin kaydedilen boyuna göre değişebilir.**

Bu örnekte dizideki sayıları tek tek mod10 işlemine tabi tutup sonuca göre diziye ekliyoruz.

Sonuç;

Hash Değeri 0: [40,20] Hash Değeri 1: [31]

Hash Değeri 2: [12] Hash Değeri 3: []

Hash Değeri 4: [44] Hash Değeri 5: [35,55]

Hash Değeri 6: [] Hash Değeri 7: [27]

Hash Değeri 8: [] Hash Değeri 9: []

**Sonuçlara göre 40, 20 ve 35, 55 aynı hash değerini veriyor bu sebeple zincir gibi birbirine bağlanıyor. Bağlı liste yapısı benzeri bir yapısı vardır. Avantajı listenin boyutuna bağlı kalmayız.**

* **Linear Probing:**

Çakışmaların (collision) yönetimi için kullanılan bir açık adresleme (open addressing) tekniğidir. Bu yöntemde, bir anahtarın hash değeri çakışma yarattığında, bir sonraki boş hücreye doğrudan gidilir. Yani, çakışma durumunda bir sonraki boş hücreye gidilerek çözümlemeye çalışılır.

**Nasıl Çalışır:**

1. Bir anahtarın hash değeri hesaplanır. H(x)=(h(x)+i)%S buradaki “i” indistir ve çakışma olduğunda artış gösterir. Ve kullanıldıktan sonra yine sıfır olur. Buradaki “S” ise dizinin boyutudur çünkü i arttıktan sonra belli bir yerde dizinin boyutunu aşabilir ve S bunu engeller.
2. Eğer hesaplanan hücre boşsa, anahtar bu hücreye yerleştirilir.
3. Eğer hücre doluysa, bir sonraki hücreye geçilir ve bu hücre boş olana kadar devam edilir.
4. Eğer sona gelinirse, başa dönülerek arama devam edebilir (dairesel bir yapı).

Örnek: Dizi=[12, 33, 26, 75, 55, 20, 44, 31]

Hash Değeri 0: [20] Hash Değeri 1: [31]

Hash Değeri 2: [12] Hash Değeri 3: [33]

Hash Değeri 4: [44] Hash Değeri 5: [55]

Hash Değeri 6: [26] Hash Değeri 7: [75]

Hash Değeri 8: [] Hash Değeri 9: []

Burada eksik olan sayı 75 tir. Çünkü mod10 işlem sonucu 5 tir ve 5 doludur. Bu sebeple “i” = 1 yaparız ve sonucumuz artık 6 olur, 6 da dolu olduğu için “i” = 2 olur ve 7 sonucunu elde ederiz sonra 7 ye yerleştirip işlemi sonlandırırız. İşlemden sonra i yeniden 0 olur.

* **Quadratic Probing:**

Linear Probing yapısına çok benzer ayrıldıkları nokta

H(x) = (H(x)+i)%S formülünde i değil i^2 oluyor yani indeksin karesi alınır. **H(x) = (H(x)+i^2)%S formülüdür.**

**Çift kıyım (Double Hashing):**

çakışmaların yönetimi için kullanılan bir açık adresleme (open addressing) tekniğidir.

**Nasıl Çalışır:**

1. İki adet hash fonksiyonu tanımlanır. İlk hash fonksiyonu anahtarın doğrudan hedef indeksini belirler, ikinci hash fonksiyonu ise çakışma durumunda kullanılır.
2. İlk hash fonksiyonuyla hesaplanan indekse gidilir. Eğer bu indeks doluysa, ikinci hash fonksiyonu da devreye girer.
3. İkinci hash fonksiyonu, çakışmanın çözümü için kullanılır. Yani, ikinci bir indeks hesaplanır ve bu indekse gidilir. Eğer bu indeks de dolu ise, ikinci hash fonksiyon tekrar uygulanarak başka bir indeks bulunana kadar devam edilir.
4. Boş bir indeks bulunduğunda, anahtar bu indekse yerleştirilir.

[**Hashing için Youtube Link:**](https://www.youtube.com/watch?v=k2bsAEkB5l4)