Algoritma Analizi

Algoritma analizi

- Konular
  - Doğruluk - correctness
  - Zaman - time efficiency
  - Bellek - space efficiency
  - En iyi çözüm - optimality

- Yaklaşımlar
  - Teorik analiz - theoretical analysis
  - Kaba analiz - empirical analysis (sayaç/süre ölçmek gibi)

Teorik Analiz

Zaman verimliliği, girdi boyutunun bir fonksiyonu olarak temel işlemin tekrar sayısı belirlenerek analiz edilir.

Temel İşlem (Basic operation): Algoritmanın çalışma süresini en çok etkileyen işlemdir. Sıralamada ..... temel işlemdir. ??

Kaba Analiz

- Bir giriş verisi seçilir
- Zaman birimini seç (milisaniye)

  veya

  Yürütülen temel işlem adımlarının sayısı
- Deneysel veriler ile analiz yapılır

Problemler - Temel işlemler

Problem	***Giriş verisinin büyüklüğü***	***Temel işlemler***
n elemanlı bir dizide eleman arama	Dizinin eleman sayısı. *n*	Elemanları karşılaştırma
İki matrisin çarpımı	Matris boyutu veya toplam eleman sayısı	İki sayının çarpımı
İnteger bir sayının asal olup olmadığının kontrolü	N sayısının dijit sayısı (ikili gösterilim)	Bölme
Graf problemi	Düğüm / Kenar sayısı	Düğüm ve kenarların gezilmesi

Temel Kavramlar

Yürütme Zamanı (Running Time) T(n)

- Algoritmanın işlevini yerine getirebilmesi için kabul edilen işlemlerden kaç adet yürütülmesi gerektiğini gösteren matematiksel bir ifadedir
- İşlem olarak karşılaştırma sayısı, çevrim sayısı, aritmetik işlem sayısı kabul edilir
- T(n) hesaplanırken hangi işleme göre hesaplandığı bildirilmelidir

Alan Maliyeti (Space Cost)

- Algoritmanın işlevini yerine getirebilmesi için gerekli olan bellek ihtiyacıdır
- Maliyet programın kodu, veri yapısı alanları ve yığın bellek alanıdır
- Alan maliyeti algoritma rekürsif değilse ve veri yapısı alanı çok çok büyük değilse pek hesaplanmaz

Asimptotik İfade

- Bir problemin büyümesinin üst, alt ve ortalama sınırını belirleyen ifadedir
- Big O → O, Big Omega → Ω, Big Theta → θ

Alan Karmaşıklığı (Space Complexity)

- Eleman sayısı n'nin büyük değerleri için bellek alanı gereksiniminin artışını gösteren asimptotik bir ifadedir

Zaman Karmaşıklığı (Time Complexity)

- Algoritmanın yürütülürken geçecek zamanın artması hakkında bilgi veren asimptotik bir ifadedir
- O(g(n)), θ(g(n)), Ω(g(n)), o(g(n)) gösterilir
- Veri kümesinin büyümesi durumunda çalışma süresinin nasıl etkileneceği hakkında bilgi verir

Yürütme Zamanı-Running Time T(n)

$T(n) = 2n^2 - 2n + 5$ olabilir.

n, ifadenin bağımsız değişkenidir ve temel işlem sayısına bağlıdır.

$T(n) = 1$	sabit
$T(n) = \log n$	logaritmik
$T(n) = n$	lineer
$T(n) = n^2$	karesel

Örnek

```
float BulOrta (float A[], int n)
{
        float ortalama, toplam=0;
        int k;
        for (k=0; k<n; k++)
                toplam=toplam + A[k];
        ortalama = toplam/n;
return ortalama;
}
```

toplam=0 **1 işlem**

k=0 **1 kere**
k<n **(n+1) kere**
k++ **n kere**
1+(n+1)+n=2n+2

atama ve toplama 2 işlem
döngü içerisinde n kere
2n

Döngü dışında bölme ve atama **2** işlem

1 işlem

T(n)=1+2n+2+2n+2+1=4n+6
T(n) = **4n + 6**

Örnek

```
float BulEnkucuk (float A[ ])
{
        float enkucuk;
        int k;
        enkucuk=A[0];
        for (k=1; k<n; k++)
                if (A[k] < enkucuk)
                enkucuk=A[k];
        return enkucuk;
}
```

atama **1 işlem**

k=1 **1 kere**
k<n **n kere**
k++ **n-1 kere**

1+n+(n-1)=2n

karşılaştırma 1 işlem **n-1 kere**

1 işlem

Bu işlemin kaç kez yürütüleceği belli değil, en kötü durumda **n-1 kere**

$T(n)=1+2n+(n-1)+(n-1)+1=4n$

$T(n) = \mathbf{4n}$

```
public static int [] selectionsort(int [] A,int n)
  {
    int tmp;
    int min;

    for(int i=0; i < n-1; i++)   I
    {
      min=i;   II

      for(int j=i; j < n; j++)   III
      {
        if (A[j] < A[min]){   IV

          min=j;   V
        }

      }
        tmp=A[i];
        A[i]=A[min];   VI
        A[min]=tmp;
    }
    return A;   VII
  }
```

	İşlem	Tekrar	Toplam
I	1,1,1	1,n,n	2n+1
II	1	n	n
III	1,1,1	n, n(n+1)/2, n(n+1)/2	n+2n(n+1)/2
IV	1	n(n+1)/2	n(n+1)/2
V	1	n(n+1)/2	n(n+1)/2
VI	1,1,1	n,n,n	3n
VII	1	1	1
		4n(n+1)/2+7n +2	

$$T(n) = 2n^2 + 9n + 2$$

Matris Toplamı

```
for (i=0; i<n; i++)
        for (j=0; j<m; j++)
            C[i][j] = A[i][j] + B[i][j];
```

$1+(n+1)+n = 2n + 2$

$(1+(m+1) + m) * n = (2m + 2)n$

$(2 * m * n)$

$4mn + 4n + 2$

m ve n eşit alırsak $T(n) = 4n^2 + 4n + 2$

Fonksiyonların Büyümesi ve Asimptotik Notasyonlar

- Hem bilgisayar biliminde hem de matematikte, bir fonksiyonun ne kadar hızlı büyüdüğü sıklıkla incelenir

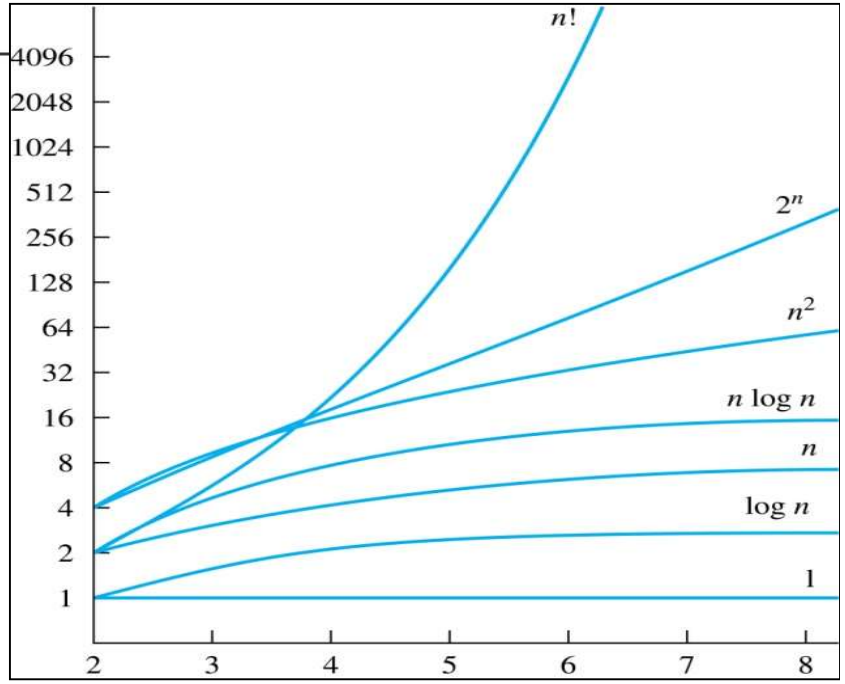
- Bilgisayar biliminde girdinin boyutu büyüdükçe, algoritmanın bir problemi ne kadar hızlı çözebileceğini bilmek isteriz
  - Aynı problemi çözmek için iki farklı algoritmanın verimliliğini karşılaştırabiliriz
  - Girdi büyüdükçe belirli bir algoritmayı kullanmanın pratik olup olmadığını da belirleyebiliriz

Karmaşıklık

- Bir algoritmanın, üzerinde işlem yapacağı veri kümesinin eleman sayısının artması durumunda yürütme zaman maliyetinin nasıl değişeceğini gösteren asimptotik bir ifadedir
  - Yürütme maliyetine, zaman karmaşıklığı
  - Bellek kullanım maliyetine, alan karmaşıklığı
- Karmaşıklıkta ki amaç gerçek zaman ve bellek büyüklüğü bilgisi değil, veri kümesi büyüdüğünde maliyet bilgisinin değişimidir

$n \to \infty$ bazı önemli fonksiyonların büyüme hızı

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		



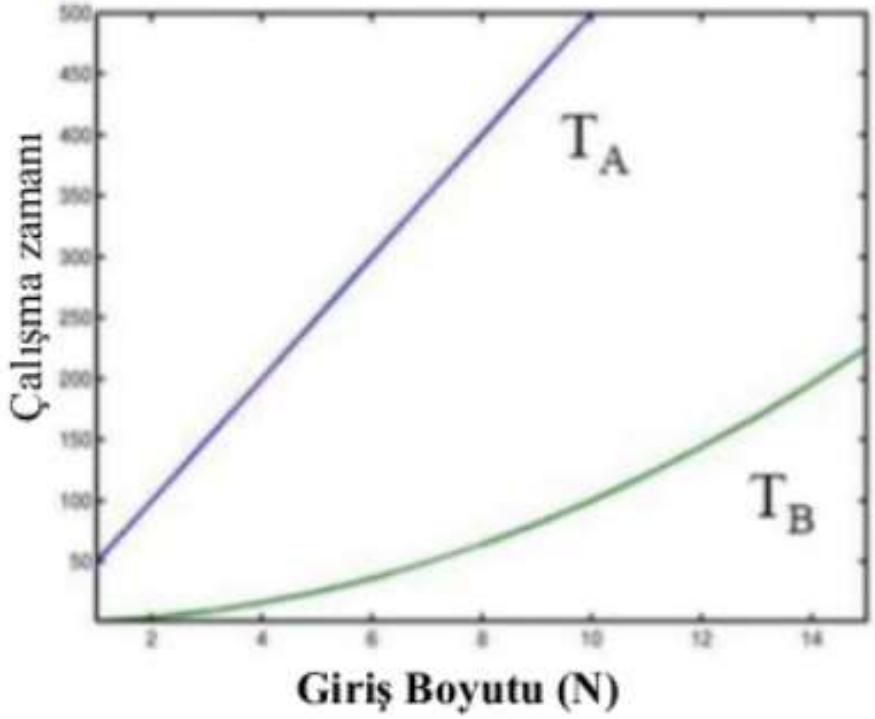
Algoritma Analiz Türleri

- **Worst case (en kötü):** Algoritma çalışmasının en fazla sürede gerçekleştiği analiz türüdür. En kötü durum, çalışma zamanında bir üst sınırdır ve o algoritma için verilen durumdan *"daha uzun sürmeyeceği"* **garantisi** verir. Bazı algoritmalar için en kötü durum ***oldukça sık rastlanır***. Arama algoritmasında, *aranan öğe* genellikle **dizide olmaz** dolayısıyla **döngü N kez çalışır**.
- **Best case (en iyi):** Algoritmanın en kısa sürede ve en az adımda çalıştığı giriş durumu olan analiz türüdür. Çalışma zamanında bir alt sınırdır.
- **Average case (ortalama):** Algoritmanın ortalama sürede ve ortalama adımda *çalıştığı* giriş durumu olan analiz türüdür.

https://www.slideshare.net/DenizKILIN/yzm-2116-blm-2-algoritma-analizi

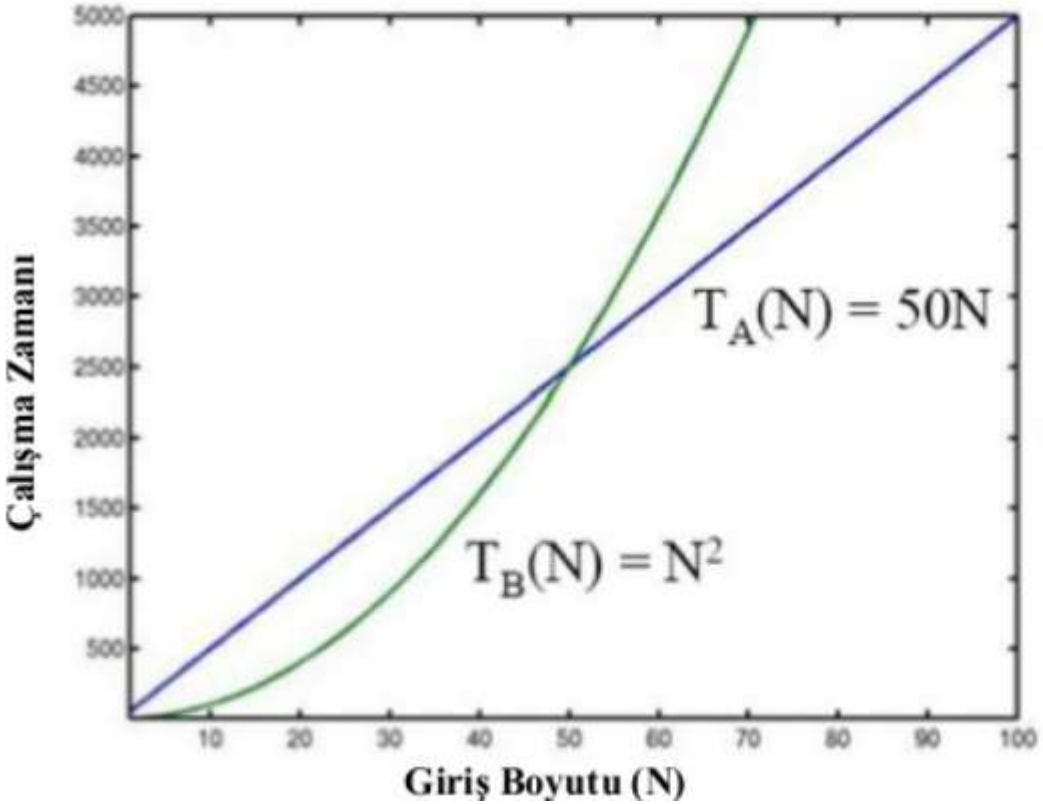
- Bir problemi çözmek için **A ve B** şeklinde iki algoritma verildiğini düşünelim.
- Giriş boyutu N için aşağıda A ve B algoritmalarının çalışma zamanı T_A **ve** T_B **fonksiyonları** verilmiştir.

Hangi algoritmayı seçersiniz?



https://www.slideshare.net/DenizKILIN/yzm-2116-blm-2-algoritma-analizi

- N büyüdüğü zaman A ve B nin çalışma zamanı:



Şimdi hangi algoritmayı seçersiniz?

https://www.slideshare.net/DenizKILIN/yzm-2116-blm-2-algoritma-analizi

- Asimptotik notasyon, **eleman sayısı n'nin sonsuza gitmesi durumunda** algoritmanın, benzer işi yapan algoritmalarla karşılaştırmak için kullanılır.
- Eleman sayısının *küçük olduğu durumlar* mümkün olabilir fakat bu birçok uygulama için geçerli değildir.
- Verilen iki algoritmanın çalışma zamanını T1(N) ve T2(N) fonksiyonları şeklinde gösterilir. Hangisinin **daha iyi** olduğunu belirlemek için bir *yol belirlememiz* gerekiyor.
  - Big-O (Big O): Asimptotik üst sınır
  - Big Ω (Big Omega): Asimptotik alt sınır
  - Big Θ (Big Teta): Asimptotik alt ve üst sınır

f(x), bir algoritmanın fonksiyon şeklindeki gösterimi ise karmaşıklık O(f(x)), Ω(f(x)), ... şeklinde gösterilir.

https://www.slideshare.net/DenizKILIN/yzm-2116-blm-2-algoritma-analizi

	n=1	n=2	n=4	n=8	n=16	n=32
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
n!	1	2	24	40320	20.9T	Don't ask!

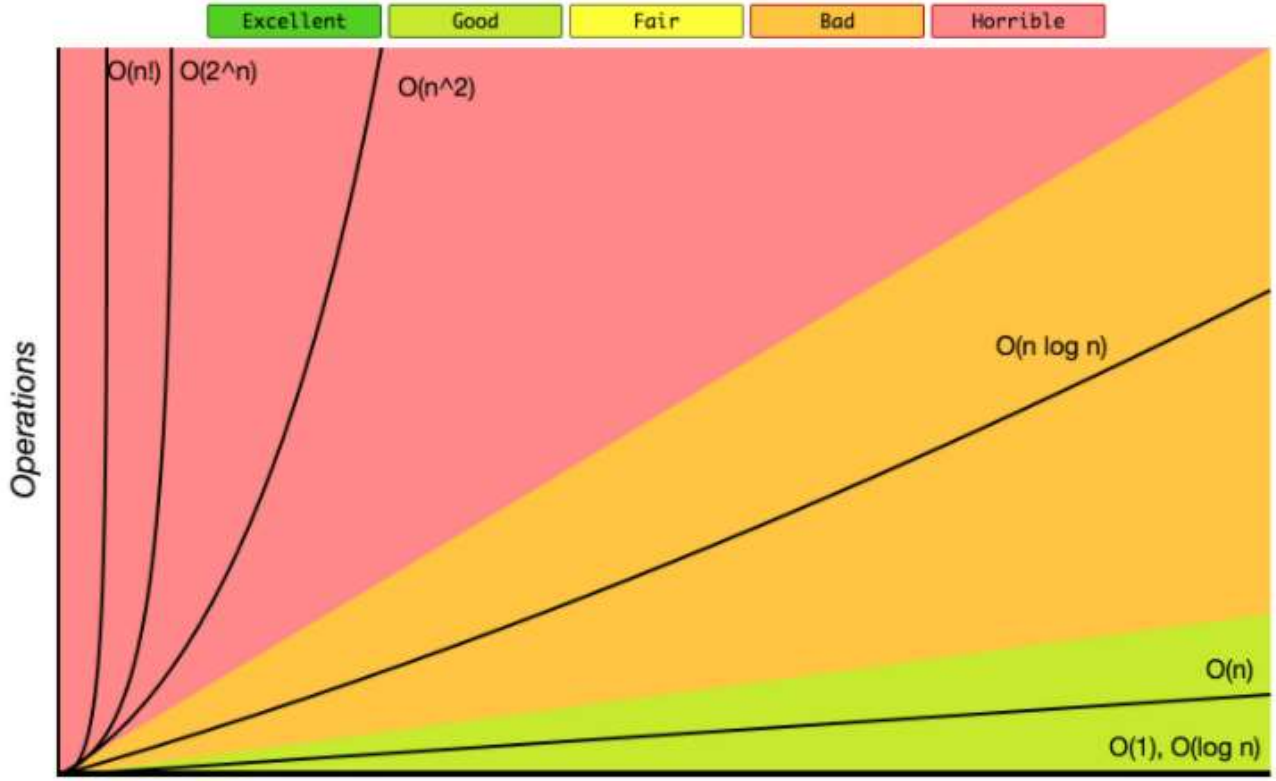


TABLE 1 Commonly Used Terminology for the Complexity of Algorithms.

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

TABLE 2 **The Computer Time Used by Algorithms.**

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-11} s	10^{-10} s	3×10^{-10} s	10^{-9} s	10^{-8} s	3×10^{-7} s
10^2	7×10^{-11} s	10^{-9} s	7×10^{-9} s	10^{-7} s	4×10^{11} yr	*
10^3	1.0×10^{-10} s	10^{-8} s	1×10^{-7} s	10^{-5} s	*	*
10^4	1.3×10^{-10} s	10^{-7} s	1×10^{-6} s	10^{-3} s	*	*
10^5	1.7×10^{-10} s	10^{-6} s	2×10^{-5} s	0.1 s	*	*
10^6	2×10^{-10} s	10^{-5} s	2×10^{-4} s	0.17 min	*	*

Büyük- O (Big-*O*) Gösterimi

Tanım: ***f*** ve ***g,*** tamsayı kümesinden veya reel sayı kümesinden reel sayılara tanımlanmış olsun. $\mathbb{Z}+ \rightarrow \mathbb{R}$

Eğer, $x > k$ olduğunda $|f(x)| \leq C|g(x)|$ oluyorsa ve bu eşitsizliği sağlayan C ve k gibi sabit sayılar varsa f(x)=O(g(x)) olmaktadır.

Örnek

Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$

Since when $x > 1$, $x < x^2$ and $1 < x^2$

$$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$$

Can take $C = 4$ and $k = 1$ as witnesses to show that

$f(x)$ is $O(x^2)$

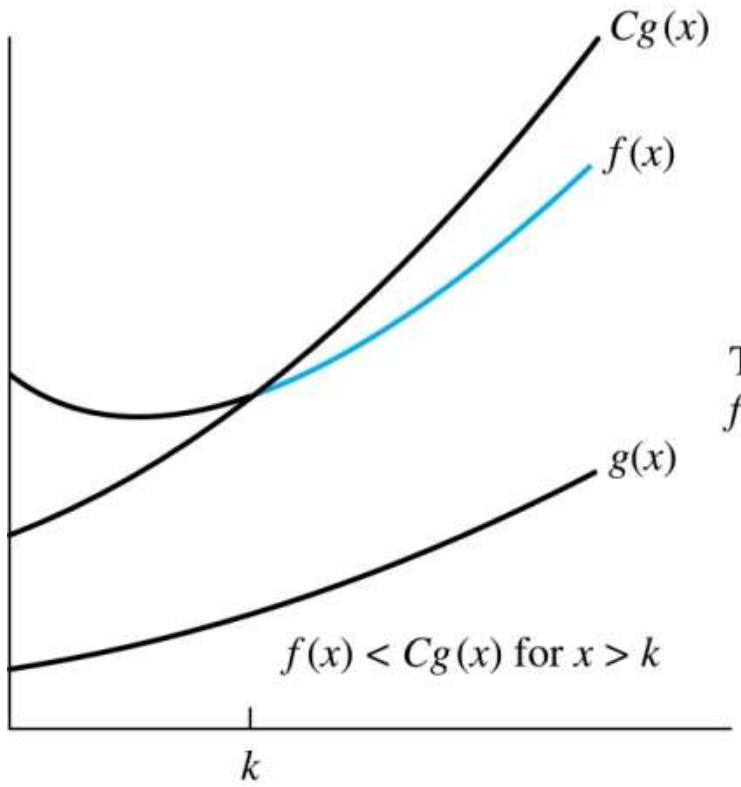
Alternatively, when $x > 2$, we have $2x \leq x^2$ and $1 < x^2$.

Hence, $0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2$

when $x > 2$.

- Can take $C = 3$ and $k = 2$ as witnesses instead.

$f(x)$ is $O(g(x))$



The part of the graph of $f(x)$ that satisfies $f(x) < Cg(x)$ is shown in color.

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$

```
//Searches for a given value in a given array by sequential search
//Input: An array A[0..n − 1] and a search key K
//Output: The index of the first element of A that matches K
//        or −1 if there are no matching elements
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1
if i < n return i
else return −1
```

- Worst case O(n)
- Best case O(1)
- Average case $\frac{1+2+3+\ldots+n}{n} = \frac{n(n+1)}{2n} = \frac{(n+1)}{2}$

Zaman karmaşıklığı	Açıklama	Örnek
O(1)	Sabit: Veri giriş boyutundan bağımsız gerçekleşen işlemler.	Bağlı listeye ilk eleman olarak ekleme yapma
O(log N)	Logaritmik: Problemi küçük veri parçalarına bölen algoritmalarda görülür.	Binary search tree veri yapısı üzerinde arama
O(N)	Lineer – doğrusal: Veri giriş boyutuna bağlı doğrusal artan.	Sıralı olmayan bir dizide bir eleman arama
O(N log N)	Doğrusal çarpanlı logaritmik: Problemi küçük veri parçalarına bölen ve daha sonra bu parçalar üzerinde işlem yapan.	N elemanı böl-parçala-yönet yöntemiyle sıralama. Quick Sort.
$O(N^2)$	Karesel	Bir grafikte iki düğüm arasındaki en kısa yolu bulma veya Buble Sort.
$O(N^3)$	Kübik	Ardarda gerçekleştirilen lineer denklemler
$O(2^N)$	İki tabanında üssel	Hanoi'nin Kuleleri problemi

Büyük- Ω(Big-*Omega)* Gösterimi

Tanım: ***f*** ve ***g,*** tamsayı kümesinden veya reel sayı kümesinden reel sayılara tanımlanmış olsun. $\mathbb{Z}+ \rightarrow \mathbb{R}$

Eğer, $x > k$ olduğunda $|f(x)| \geq C|g(x)|$ oluyorsa ve bu eşitsizliği sağlayan C ve k gibi sabit sayılar varsa f(x)=Ω(g(x)) olmaktadır.

Big-O ile Big- Ω arasında sıkı bir ilişki vardır.
Ancak ve ancak g(x), O(f(x)) olduğunda f(x), Ω(g(x)) olacaktır.

Example: Show that $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(g(x))$ where $g(x) = x^3$.

Solution: $f(x) = 8x^3 + 5x^2 + 7 \geq 8x^3$ for all positive real numbers x.

- Is it also the case that $g(x) = x^3$ is $O(8x^3 + 5x^2 + 7)$?

Büyük- θ(Big-*Theta)* Gösterimi

Tanım: ***f*** ve ***g,*** tamsayı kümesinden veya reel sayı kümesinden reel sayılara tanımlanmış olsun. $\mathbb{Z}+ \rightarrow \mathbb{R}$

Eğer, f(x), O(g(x)) ve f(x), Ω(g(x)) ise f(x), θ(g(x)) deriz.

Eğer x>k olduğunda $C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$ oluyorsa ve bu eşitsizliği sağlayan pozitif $\boldsymbol{C_1}$ ve $\boldsymbol{C_2}$ reel sayıları ve bir pozitif ***k*** reel sayısı bulunabiliyorsa bu durumda f(x)'in θ(g(x)) olduğunu gösterebiliriz.

* Ex:

$f(x) = 4x^3 + 3x^2$, $g(x) = x^4$

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \frac{4x^3+3x^2}{x^4} = \lim_{x\to\infty} \frac{4x^3}{x^4} + \lim_{x\to\infty} \frac{3x^2}{x^4}$$

$$\underbrace{\lim_{x\to\infty} \frac{4}{x}}_{\emptyset} + \underbrace{\lim_{x\to\infty} \frac{3}{x^2}}_{\emptyset} = \emptyset$$

$L = \emptyset$ çıktığı için,

$$4x^3 + 3x^2 \in O(x^4)$$

veya

$4x^3 + 3x^2 \leq C.(x^4)$ sağlayan

$C = 7$ gibi bir sayı vardır. $\therefore$ $O(x^4)$ olur.

Ex| $n>6$ ve n bir tamsayı ıçın $3n<n!$ ispatlayınız.

Tümevarım olabilir veya Big O kullanılır.

1) $n=\{7,8,9...\}$ kümesinden;

$n=7$ için

$3.7<7!$ doğru

2) $n=k$ doğru kabul edilir.

$3k<k!$ doğru kabul edilir.

3) $n=k+1$

$3(k+1) \overset{?}{<} (k+1)!$ doğruluğu gösterilir.

$3k+3<(k+1).k!$

$3k+3<k.k!+k!$ $\quad k!>3k$ $k>6$ ıçın doğru kabul edildi

Burada $k.k!>3$ ispat edilmeli $k>6$ ıçın bu doğrudur

$\therefore$ $3n<n!$ doğrudur.

Arama Algoritmaları

Searching

Algorithm	Data Structure	Time Complexity		Space Complexity
		Average	Worst	Worst
Depth First Search (DFS)	Graph of |V| vertices and |E| edges	-	O(|E| + |V|)	O(|V|)
Breadth First Search (BFS)	Graph of |V| vertices and |E| edges	-	O(|E| + |V|)	O(|V|)
Binary search	Sorted array of n elements	O(log(n))	O(log(n))	O(1)
Linear (Brute Force)	Array	O(n)	O(n)	O(1)
Shortest path by Dijkstra, using a Min-heap as priority queue	Graph with |V| vertices and |E| edges	O((|V| + |E|) log |V|)	O((|V| + |E|) log |V|)	O(|V|)
Shortest path by Dijkstra, using an unsorted array as priority queue	Graph with |V| vertices and |E| edges	O(|V|^2)	O(|V|^2)	O(|V|)
Shortest path by Bellman-Ford	Graph with |V| vertices and |E| edges	O(|V||E|)	O(|V||E|)	O(|V|)

Sıralama Algoritmaları

Sorting

Algorithm	Data Structure	Time Complexity			Worst Case Auxiliary Space Complexity
		Best	Average	Worst	Worst
Quicksort	Array	O(n log(n))	O(n log(n))	O(n^2)	O(n)
Mergesort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Heapsort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Select Sort	Array	O(n^2)	O(n^2)	O(n^2)	O(1)
Bucket Sort	Array	O(n+k)	O(n+k)	O(n^2)	O(nk)
Radix Sort	Array	O(nk)	O(nk)	O(nk)	O(n+k)

Heap ve Graf Algoritmaları

Heaps

Heaps	Time Complexity						
	Heapify	Find Max	Extract Max	Increase Key	Insert	Delete	Merge
Linked List (sorted)	-	O(1)	O(1)	O(n)	O(n)	O(1)	O(m+n)
Linked List (unsorted)	-	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)
Binary Heap	O(n)	O(1)	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(m+n)
Binomial Heap	-	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Fibonacci Heap	-	O(1)	O(log(n))*	O(1)*	O(1)	O(log(n))*	O(1)

Graphs

Node / Edge Management	Storage	Add Vertex	Add Edge	Remove Vertex	Remove Edge	Query
Adjacency list	O(|V|+|E|)	O(1)	O(1)	O(|V| + |E|)	O(|E|)	O(|V|)
Incidence list	O(|V|+|E|)	O(1)	O(1)	O(|E|)	O(|E|)	O(|E|)
Adjacency matrix	O(|V|^2)	O(|V|^2)	O(1)	O(|V|^2)	O(1)	O(1)
Incidence matrix	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|E|)

Birçok algoritma birden fazla alt programdan oluşabilir.

$$\left.\begin{array}{l} f1 \rightarrow O(n^c) \\ f2 \rightarrow O(n^d) \end{array}\right\} \max(O(n^c), O(n^d)) \rightarrow O(n^d)$$

$1<c<d$ ise

$$\left.\begin{array}{l} f1 \rightarrow O(\log n) \\ f2 \rightarrow O(n) \end{array}\right\} \max(O(\log n), O(n)) \rightarrow O(n)$$

$$\left.\begin{array}{l} f1 \rightarrow O(2^n) \\ f2 \rightarrow O(n) \end{array}\right\} \max(O(2^n), O(n)) \rightarrow O(2^n)$$

$$\left.\begin{array}{l} f1 \rightarrow O(n^2) \\ f2 \rightarrow O(n^2) \end{array}\right\} \rightarrow O(n^2)$$

$f(n) = 3n \log(n!) + (n^2+3) \log n$

n, pozitif bir tamsayı olmak üzere Big-O ?

$3n \log(n!) \rightarrow 3n$ ve $\log(n!)$

$3n \rightarrow O(n)$

$\log(n!) \rightarrow O(\log n^n)$

$O(\log n^n) = O(n \log n)$

$O(n) * O(n \log n) = O(n^2 \log n)$

$(n^2+3) \log n$

$(n^2+3) \rightarrow n^2 \log n$

$\log n \rightarrow 3 \log n$

$n^2 \log n + 3 \log n \rightarrow O(n^2 \log n)$

$O(n^2 \log n)$ max $O(n^2 \log n)$

$F(x) = (x+1)\log(x^2+1) + 3x^2$ Big-O ?

$$\left.\begin{array}{l} O(x+1) \rightarrow O(x) \\ O(\log(x^2+1)) \rightarrow O(\log x^2) \rightarrow O(2\log x) \rightarrow O(\log x) \end{array}\right\} O(x\log x)$$

$$3x^2 \rightarrow O(3x^2) \rightarrow O(x^2)$$

$$\max(O(x\log x), O(x^2)) \rightarrow O(x^2)$$

İteratif ve Özyinelemeli Algoritmaların Analizi

İteratif (nonrecursive) algoritmaların analizi

Genel Adımlar:

- *n* girdi boyutu *(input size)* belirlenir
- Algoritmanın temel operasyonu saptanır *(basic operation)*
- Durum analizleri (worst, average ve best cases for input of size) n değerine göre belirlenir
- Temel operasyonun kaç kez işletileceğini hesaplamak için toplama işlemi yapılır ve *n* değerine bağlı bir çalışma zamanı fonksiyonu *T(n)* elde edilir
- Toplama sonucu elde edilen *n*'e bağlı fonksiyon *T(n)*, asimptotik notasyonlara göre ifade edilir

Insertion Sort (Sokma Sıralaması)

A *pseudocode* for insertion sort (INSERTION SORT)

```
INSERTION-SORT(A)
    for j ← 2 to length [A]
        do key ← A[ j]
          Insert A[j] into the sortted sequence A[1,..., j-1].
        i ← j – 1
        while i > 0 and A[i] > key
            do A[i+1] ← A[i]
                i ← i – 1
      A[i +1] ← key
```

11	7	15	3	16	13
0	1	2	3	4	5

(key indisi = 1)

11	7	15	3	16	13
0	1	2	3	4	5

7	11	15	3	16	13
0	1	2	3	4	5

7	11	15	3	16	13
0	1	2	3	4	5

(key indisi = 2)

7	11	15	3	16	13
0	1	2	3	4	5

7	11	15	3	16	13
0	1	2	3	4	5

(key indisi = 3)

7	11	15	3	16	13
0	1	2	3	4	5

3	7	11	15	16	13
0	1	2	3	4	5

3	7	11	15	16	13
0	1	2	3	4	5

(key indisi = 4)

3	7	11	15	16	13
0	1	2	3	4	5

3	7	11	15	16	13
0	1	2	3	4	5

(key indisi = 5)

3	7	11	15	16	13
0	1	2	3	4	5

3	7	11	13	15	16
0	1	2	3	4	5

3	7	11	13	15	16

INSERTION-SORT(A)	cost	times
for $j \leftarrow 2$ **to** $length[A]$	c_1	n
do $key \leftarrow A[j]$	c_2	$n-1$
∇ Insert $A[j]$ into the sorted sequence $A[1 \cdot\cdot j-1]$	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ *and* $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

Toplam Çalışma süresi

$$T(n) = c_1 + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1)$$

$$+ c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1).$$

Best-case : O(n)

Dizi başlangıçta sıralı ise. Yer değiştirme yapılmaz

Average-case : $O(n^2)$

Worst case : $O(n^2)$

Başlangıçta dizi büyükten küçüğe sıralı ise her eleman için **en başa** kadar karşılaştırma yapılacaktır.

Selection Sort (Seçmeli Sıralama)

```
procedure selection sort
   list  : array of items
   n     : size of list

   for i = 1 to n - 1
   /* set current element as minimum*/
      min = i

      /* check the element to be minimum */

      for j = i+1 to n
         if list[j] < list[min] then
            min = j;
         end if
      end for

      /* swap the minimum element with the current element*/
      if indexMin != i  then
         swap list[min] and list[i]
      end if
   end for

end procedure
```

17	33	14	27	18
0	1	2	3	4

Swap Yapılır. (**i = 0**)

17	33	14	27	18
İ =0	1	2	3	4

14	33	17	27	18
İ =0	1	2	3	4

Swap Yapılır. (**i = 1**)

14	33	17	27	18

14	17	33	27	18

Swap Yapılır. (**i = 2**)

14	17	33	27	18
0	1	2	3	4

14	17	18	27	33
0	1	2	3	4

Swap Yapılır. (**i = 3**)

14	17	18	27	33
0	1	2	3	4

```
void selection(int A[], int N)
{ int i, j, temp;
  for (j = 0; j < N-1; j++)
  { int min_idx = j;
    for (i = j+1; i < N; i++)
      if (A[i] < A[min_idx])
            min_idx = i;
    temp = A[min_idx];
    A[min_idx] = A[j];
    A[j] = temp;
  }
}
```

iterations → (N-1)

iterations → (N-1-j) (depends on j)

j	Iterations of inner loop = N-1-j
0	N-1
1	N-2
2	N-3
...	...
N-3	2
N-2	1

Total instructions (all iterations of inner loop for all values of j)

$T(N) = (N-1) + (N-2) + \ldots 2 + 1 =$

$= [N * (N-1)]/2 \rightarrow N^2$ order of magnitude

Note that the came from the summation NOT because 'there is an N in the inner loop' (NOT because N * N).

22

worst-case : $O(n^2)$

average-case : $O(n^2)$

best_case : $O(n^2)$

Öz yinelemeli (Recursive) Algoritmaların Analizi

- *n* girdi boyutu (*input size)* belirlenir
- Algoritmanın temel operasyonu saptanır (*basic operation)*
- Temel işlemin gerçekleştirilme sayısının aynı boyuttaki farklı girişlere göre değişiklik gösterip göstermediği kontrol edilir (Eğer değişiklik gösterirse, durum analizi ayrı ayrı the worst, average ve best cases yapılmalıdır)
- Temel işlemin kaç kez yürütüldüğünü ifade eden uygun bir başlangıç koşulu ile bir özyineleme bağıntısı (recurrence relation) kurulur
- Recurrence relation uygun bir yöntemle çözülür. Çözüm sonucu elde edilen *n*'e bağlı fonksiyon *T(n),* asimptotik notasyonlara göre ifade edilir

Öz yinelemeli (Recursive) bağıntılarının çözümü

- *n* bağımsız değişkene göre ifade edilen ve *n* değerinin önceki değerlerini içinde bulunduran fonksiyonlardır

$$T(n) = T(n-1) + 1$$

- Çözüm yöntemleri
  - Forward Subsitition
  - Backward Subsitition
  - Recursive Tree Method
  - Master Theorem
  - Karakteristik Denklem Yöntemi (2. veya yüksek dereceden bağıntılar için)

Forward Subsitition

- Recurrence: $T(n) = T(n-1) + 1$, with initial condition $t(1) = 2$
- Look for a pattern:
  - $T(1) = 2$, Initial condition
  - $T(2) = T(1) + 1 = 2+1 = 3$
  - $T(3) = T(2) + 1 = 3+1 = 4$
  - $T(4) = T(3) + 1 = 4+1 = 5$
  - $T(5) = T(4) + 1 = 5+1 = 6$
- Guess:
  - $T(n) = n + 1$

- Informal Check:
  - $T(n) = T(n-1)+1 = [(n-1)+1] + 1 = n+1$
  - $T(1) = 1+1 = 2$

Örnek : 1'den 5'e kadar olan sayıları toplayalım.

$$1 + 2 + 3 + \cdots . + (n-2) + (n-1) + n = \frac{n(n+1)}{2}$$

Gauss

1+	2+	3+	...+	(n-2)+	(n-1)+	n
n+	(n-1)+	(n-2)+	..+	3+	2+	1

f(n)=n + f(n-1) F(1)=1 ilk değer

f(5) = 5 + f(4) 10

15

f(4) = 4 + f(3) 6

f(3) = 3 + f(2) 3

f(2) = 2 + f(1) 1

Örnek : n sayısının faktöriyelini hesaplayalım

factorial(n):
if n is 0
return 1
return n * factorial(n-1)

$T(n) = T(n — 1) + 3$
$T(0) = 1$

- factorial(0) sadece 1 karşılaştırma (1 unit of time)
- factorial(n) (1 karşılaştırma, 1 multiplication, 1 subtraction) (n-1) kez

```
T(n) = T(n-1) + 3
     = T(n-2) + 6
     = T(n-3) + 9
     = T(n-4) + 12
     = ...
     = T(n-k) + 3k


as we know T(0) = 1
we need to find the value of k for which n - k = 0, k = n


T(n) = T(0) + 3n , k = n
     = 1 + 3n


that gives us a time complexity of O(n)
```

Rekürans Bağıntıları (Özyineli Bağıntılar)

Bir dizinin içinde kendisinden bir parça bulunuyorsa o diziye rekürans bağıntısı (özyineli dizi) denir.

$a_n = 5n+1$ (dizi) (sequence) $\longrightarrow a_{20} = 101$

$\boxed{a_n = 2a_{n-1} + 3}$ (a_{n-1} kendisinden parça) rekürans bağıntısı

$a_0 = 3 \qquad n \geq 1$

$a_n = a_{n-1} - 2a_{n-2} + 5^n$

$a_n = 3a_{n-1} + 2a_{n-2}$

$\longrightarrow a_1 = 3,\ a_0 = 5,\ n \geq 2$

rekürans bağıntısı

Ör: $a_n = 2a_{n-1} - a_{n-2} + 2^{n-1}$, $a_0 = 1$, $a_1 = 2$, $n \geq 2$ olarak veriliyor. Buna göre, a_5 nedir?

$n=2 \Rightarrow a_2 = 2a_1 - a_0 + 2 \Rightarrow a_2 = 4 - 1 + 2 \Rightarrow \boxed{a_2 = 5}$

$n=3 \Rightarrow a_3 = 2a_2 - a_1 + 4 \Rightarrow a_3 = 10 - 2 + 4 \Rightarrow \boxed{a_3 = 12}$

$n=4 \Rightarrow a_4 = 2a_3 - a_2 + 8 \Rightarrow a_4 = 24 - 5 + 8 \Rightarrow \boxed{a_4 = 27}$

$n=5 \Rightarrow a_5 = 2a_4 - a_3 + 16 \Rightarrow a_5 = 54 - 12 + 16 \Rightarrow a_5 = 58$ //

Ayrık matematikteki hedefimiz

Rekürans Bağıntısı

Rekürans bağıntısını çözmek

$a_n=3a_{n-1} + a_{n-2}$

$a_n=2^n - n + 1$

İç bileşenlerden kurtaracak şekilde yazmalıyız

Sözel bir problemi rekürans bağıntısına çevirme

Rekürans Bağıntılarının Sınıflandırılması

a_n'nin kendisinden sonraki kaç terime bağlı olması rekürans bağıntısının derecesini verir

1) Derece (Order)

$a_n = 5a_{n-1} + n^2$ → 1 first order

$a_n = 3a_{n-1} - 5a_{n-2}$ → 2 second order

$a_n = 3a_{n-1} + a_{n-3} + 3^n$ → 3 third order

$a_n = 2a_{n-4} + 1$ → 4. derece

2) Homojen – homojen değil (homogeneous – nonhomogeneous)

$a_n = 3a_{n-1}$ ✓

$a_n = a_{n-1} - 2a_{n-2}$ ✓

$a_n = 3a_{n-1} + n \cdot a_{n-2} + 3^n$ ✗

$a_n = a_{n-1} + 2a_{n-3} - 4$ ✗

$a_n = a_{n-2} + n^3 + n^2$ ✗

3) Lineer, lineer olmayan (Linear, nonlinear)

a_n'li terimlerin üslerinin alınması veya birbirleri ile çarpılması lineer'liği bozar.

$$a_n = 5a_{n-2} + n.a_{n-3} + n^2 \quad ✓$$

$$a_n = (a_{n-1})^2 + 3n \quad ✗$$

$$(a_n)^3 = a_{n-1} + a_{n-2} \quad ✗$$

$$a_n = (a_{n-1}).(a_{n-2}) - 3 \quad ✗$$

4) Sabit Katsayılı olan ve olmayan (constant coefficient)

a_n'li ifadeler n'li birşey ile çarpım durumunda olursa sabit katsayılı değildir.

$$a_n = 3a_{n-1} + 5a_{n-2} + n - 7 \quad ✓$$

$$a_n = n.a_{n-1} + 3a_{n-2} + 5^n \quad ✗$$

$$a_n = 7a_{n-1} + 10a_{n-2} \quad ✓$$

$$a_n = 3^n.a_{n-2} + 1 \quad ✗$$

Sabit Katsayılı Homojen Rekürans Bağıntılarının Çözülmesi (Karakteristik Kök Tekniği)

Karakteristik Kök Tekniği

- İkinci derece ve daha büyük dereceli rekürans bağıntılarının çözümünde kullanılır.
- Sabit ve homojen rekürans bağıntılarında kullanılır.
- Homojen olmayan rekürans bağıntısında ise homojen hale getirmek için kullanılır.

$$a_n = 3a_{n-1} + 4a_{n-2}$$

Sabit Katsayılı Homojen 2. derece bir rekürans bağıntısı

Homojen olduğundan

$$a_n = a_n^{(h)}$$

homojen olmadığında

$$a_n = a_n^{(h)} + a_n^{(ö)}$$

$$a_{n-2} = 1$$

$$a_{n-1} = r$$

$$a_n = r^2$$

$$r^2 = 3r + 4$$

$$r^2 - 3r - 4 = 0$$

karakteristik denklem

2 tane farklı kökü varsa

Eşit 2 tane kökü varsa

Karakteristik Denklemin Köklerine Göre Homojen Çözümün Yazımı

2. derece

1) $r_1 \neq r_2$ ise $\Longrightarrow$ $\boxed{a_n^{(h)} = c_1 \cdot (r_1)^n + c_2 \cdot (r_2)^n}$

2) $r_1 = r_2$ ise $\Longrightarrow$ $\boxed{a_n^{(h)} = c_1 \cdot (r_1)^n + c_2 \cdot (r_2)^n \cdot n}$

$r = 4 \quad r = -1$

$(r-4)(r+1) = 0$

$\boxed{r^2 - 3r - 4 = 0}$

$$a_n = c_1 \cdot 4^n + c_2 \cdot (-1)^n$$

3. derece

1) $r_1 \neq r_2 \neq r_3 \Rightarrow a_n^{(h)} = c_1 \cdot (r_1)^n + c_2 \cdot (r_2)^n + c_3 \cdot (r_3)^n$

2) $r_1 = r_2 = r_3 \Rightarrow a_n^{(h)} = c_1 \cdot (r_1)^n + c_2 \cdot (r_2)^n \cdot n + c_3 \cdot (r_3)^n \cdot n^2$

3) $r_1 = r_2 \neq r_3 \Rightarrow a_n^{(h)} = c_1 \cdot (r_1)^n + c_2 \cdot (r_2)^n \cdot n + c_3 \cdot (r_3)^n$

Soru: $a_n = 3a_{n-1} + 10a_{n-2}$, $a_0 = 5$ ve $a_1 = 8$ olan başlangıç değerleri verilmiş rekurans bağıntısını çözünüz.

1.adım $\quad a_n = a_n^{(h)}$

2.adım $\quad r^2 = 3r + 10$ (karakteristik denklem)

2.derece

$$r^2 - 3r - 10 = 0$$

$$(r-5)(r+2) = 0$$

$$5 \qquad -2$$

$$\boxed{a_n^{(h)} = c_1(5)^n + c_2(-2)^n}$$

3.adım

$$a_n = c_1 5^n + c_2(-2)^n$$

$$a_0 = 5 \Rightarrow \boxed{c_1 + c_2 = 5} \;/2$$

$$a_1 = 8 \Rightarrow \boxed{5c_1 - 2c_2 = 8}$$

$$7c_1 = 18$$

$$\boxed{c_2 = \frac{17}{7}}$$

$$\boxed{a_n = \frac{18}{7} \cdot 5^n + \frac{17}{7} \cdot (-2)^n}$$

Soru: $a_n = -4a_{n-1} - 4a_{n-2}$, $a_0 = 3$, $a_1 = 5$ başlangıç koşulları ile verilen rekürans bağıntısının çözümünü bulunuz.

1. adım $a_n = a_n^{(h)}$

2 adım $a_{n-2} = 1$

2. derece $a_{n-1} = r$ $\Rightarrow$ $r^2 = -4r - 4 \Rightarrow r^2 + 4r + 4 = 0$

$a_n = r^2$

$(r+2)(r+2) = 0$

$r_1 = -2$ $r_2 = -2$

$$a_n^{(h)} = c_1 . (-2)^n + c_2 (-2)^n . n$$

3. adım $a_n = c_1(-2)^n + c_2(-2)^n .$

$$\boxed{a_n = 3.(-2)^n - \frac{11}{2}(-2)^n . n}$$

$a_0 = 3 \Rightarrow \boxed{c_1 = 3}$

$a_1 = 5 \Rightarrow -2c_1 - 2c_2 = 5 \Rightarrow -2c_2 = 11 \Rightarrow \boxed{c_2 = -\frac{11}{2}}$

Sabit Katsayılı Homojen Olmayan Rekürans Bağıntılarının Çözülmesi

$$a_n = 3a_{n-1} + 4a_{n-2} + f(n)$$

genel özel

$$a_n^{(g)} = a_n^{(h)} + a_n^{(ö)}$$

polinom olursa	üstel fonksiyon olursa
1	3.2^n
3	5^n
$n+2$	7^{-n}
n^2	

Özel çözüm bulunurken $a_n^{(ö)}$ tahmin edilir ve çözüm bu şekilde elde edilir.

Polinom Durumu

Ör: $a_n = 2a_{n-1} + 3a_{n-2} + 5$, $a_0 = 4$, $a_1 = 10$ başlangıç koşulları ile verilen rekürans bağıntısını çözünüz.

1. adım $a_n^{(g)} = a_n^{(h)} + a_n^{(ö)}$

2. adım $a_n^{(h)}$ bulunur: $\underset{r^2}{a_n} = 2\underset{r}{a_{n-1}} + 3\underset{1}{a_{n-2}} \Rightarrow r^2 = 2r + 3$

$$r^2 - 2r - 3 = 0$$

$$(r-3)(r+1) = 0$$

$$r = 3 \quad r = -1$$

$$\boxed{a_n^{(h)} = c_1 . 3^n + c_2 . (-1)^n}$$

$$a_n = 2a_{n-1} + 3a_{n-2} + 5$$

not: polinomun derecesi ne ise a_n nin de dereceside aynı olur
f(n) sabit olduğundan tüm terimler sabite eşit olur

3. adım $a_n^{(ö)}$ tahmin edilir.

$a_n^{(ö)} = A$

$a_{n-1} = A$

$a_{n-2} = A$

ardından a_{n-1} ve a_{n-2} elde edilip eşitlikte yerine konulur, ama c tahmindeki katsayıyı bulmaktır.

$$A = 2A + 3A + 5 \Rightarrow -4A = 5 \Rightarrow \boxed{A = -\frac{5}{4}}$$

$$\boxed{a_n^{(ö)} = -\frac{5}{4}}$$

4. adım :

$$\boxed{a_n = c_1 \cdot 3^n + c_2 \cdot (-1)^n - \frac{5}{4}}$$

$a_0 = 4$

$a_1 = 10$

$$c_1 + c_2 - \frac{5}{4} = 4 \Rightarrow c_1 + c_2 = \frac{21}{4}$$

$$3c_1 - c_2 - \frac{5}{4} = 10 \Rightarrow 3c_1 - c_2 = \frac{45}{4}$$

$$\Rightarrow 4c_1 = \frac{\cancel{66}}{\cancel{4}} \; \frac{33}{2}$$

$$\boxed{c_1 = \frac{33}{8}} \quad \boxed{c_2 = \frac{9}{8}}$$

$$\boxed{a_n = \frac{33}{8} \cdot 3^n + \frac{9}{8}(-1)^n - \frac{5}{4}}$$

Ör: $a_n = a_{n-1} + 2a_{n-2} \boxed{+3n+4}$, $a_0 = 5$, $a_1 = 7$ başlangıç koşulları ile verilen rekürans bağıntısını çözünüz.

<u>1. adım</u> $a_n^{(g)} = a_n^{(h)} + a_n^{(ö)}$

<u>2. adım</u> $a_n^{(h)} \longrightarrow \overset{r^2}{a_n} = \overset{r}{a_{n-1}} + 2\overset{1}{a_{n-2}} \Rightarrow r^2 = r + 2$

$r^2 - r - 2 = 0$

$(r-2)(r+1) = 0$

$2 \quad -1$

$$\boxed{a_n^{(h)} = c_1 . 2^n + c_2 . (-1)^n}$$

1. derece polinom

$a_n = a_{n-1} + 2a_{n-2} \boxed{+3n+4}$

<u>3. adım</u>

$$\left.\begin{aligned} a_n^{(ö)} &= An + B \\ a_{n-1} &= An - A + B \\ a_{n-2} &= An - 2A + B \end{aligned}\right\}$$

$\cancel{An} + \cancel{B} = \cancel{An} - A + \cancel{B} + 2An - 4A + 2B + 3n + 4$

$-2An + 5A - 2B = 3n + 4$

$\boxed{A = -\frac{3}{2}}$

$5A - 2B = 4 \Rightarrow -\frac{15}{2} - 2B = 4$

$-\frac{23}{2} = 2B \Rightarrow \boxed{B = -\frac{23}{4}}$

$$\boxed{a_n^{(ö)} = -\frac{3}{2}n - \frac{23}{4}}$$

4. adım

$$\boxed{a_n = c_1 \cdot 2^n + c_2 \cdot (-1)^n - \frac{3}{2}n - \frac{23}{4}}$$

$a_0 = 5$

$a_1 = 7$

c_1 ve c_2 bulunup en son a_n yazılır.

Üstel Fonksiyon Durumu

Ör: $a_n = -4a_{n-1} - 3a_{n-2} + 2.5^n$, $a_0 = 6$, $a_1 = 10$ başlangıç koşulları ile verilen rekürans bağıntısını çözünüz.

5^n

$a_n^{(h)}$ içinde var mı?

YOK

1. adım: $a_n^{(g)} = a_n^{(h)} + a_n^{(ö)}$

2. adım $a_n^{(h)} \Rightarrow a_n = -4a_{n-1} - 3a_{n-2}$ (r^2, r, 1) $\Rightarrow r^2 = -4r - 3$

$r^2 + 4r + 3 = 0$

$(r+3)(r+1) = 0$

$r = -3 \quad r = -1$

$$\boxed{a_n^{(h)} = c_1(-3)^n + c_2.(-1)^n}$$

Eğer varsa

1 tane varsa $a_n^{ö} \to$ 1 tane n çarpanı gelir.

2 tane varsa $a_n^{(ö)} \to$ 2 tane n^2 çarpanı gelir.

3. adım

$$\left.\begin{aligned} a_n^{(ö)} &= A.5^n \\ a_{n-1} &= A.5^{n-1} = \frac{A}{5}.5^n \\ a_{n-2} &= A.5^{n-2} = \frac{A}{25}.5^n \end{aligned}\right\} \Rightarrow$$

$A.5^n = -\frac{4A}{5}5^n - \frac{3A}{25}.5^n + 2.5^n$

$A = -\frac{23A}{25} + 2 \Rightarrow \frac{48A}{25} = 2$

$A = \frac{25}{24}$

$a_n^{(ö)} = \frac{25}{24}.5^n$

4. adım: $a_n = c_1.(-3)^n + c_2.(-1)^n + \frac{25}{24}.5^n$

$a_0 = 6$

$a_1 = 10$

$c_1 + c_2 + \frac{25}{24} = 6$

$-3c_1 - c_2 + \frac{125}{24} = 10$

$\Rightarrow c_1 = \cdots$

$c_2 = \cdots$

$a_n = \cdots$

Ör: $a_n = 2a_{n-1} + 3a_{n-2} + 3^n$, $a_0 = 7$, $a_1 = 5$ ..... çözünüz.

1 tane var.

1.adım $a_n^{(g)} = a_n^{(h)} + a_n^{(ö)}$

2.adım $a_n^{(h)} \Longrightarrow \overset{r^2}{a_n} = 2\overset{r}{a_{n-1}} + 3\overset{1}{a_{n-2}} \Rightarrow r^2 - 2r - 3 = 0$

$(r-3)(r+1) = 0$

$r = 3 \quad r = -1$

$$\boxed{a_n^{(h)} = c_1 3^n + c_2 . (-1)^n}$$

3.adım $a_n^{(ö)} = A.3^n.n$

$a_{n-1} = A.3^{n-1}.(n-1) = \frac{A}{3}.3^n.n - \frac{A}{3}.3^n$

$a_{n-2} = A.3^{n-2}.(n-2) = \frac{A}{9}.3^n.n - \frac{2A}{9}.3^n$

$$\cancel{A.3^n.n} = \underbrace{\cancel{\frac{2A}{3}.3^n.n} - \frac{2A}{3}.3^n + \cancel{\frac{A}{3}.3^n.n}}_{A.3^n.n} - \frac{2A}{3}.3^n + 3^n$$

$\frac{4A}{3}.3^n = 3^n \qquad \frac{4A}{3} = 1 \qquad \boxed{A = \frac{3}{4}} \qquad \boxed{a_n^{(ö)} = \frac{3}{4}.3^n.n}$

Sayı Dizilerini Rekürans Bağıntısına Dönüştürme

Bizlere bazen bir sayı dizisi bazen de sözel bir ifade verilerek ondan bir sayı dizi formunda olayı yazmamız beklenir.

$\underline{\underline{4}}, 7, 13, 25, 49, 97, \ldots\ldots$ Sayı dizisi

Rekürans bağıntısı : $\boxed{a_0 = 4 \quad , \quad a_n = 2a_{n-1} - 1 \quad , \quad n \geq 1}$

Ör: $1, 3, 7, 17, 41, 99, 239, \ldots$ sayı dizisinin rekürans bağıntısını olarak yazınız.

$\boxed{\begin{array}{l} a_0 = 1 \\ a_1 = 3 \end{array}}$ $\boxed{a_n = 2a_{n-1} + a_{n-2} \quad , \quad n \geq 2}$

Homojen olmayan rekürans bağıntısı için bir örnek yazalım

Ör: 1, 3, 6, 13, 27, 56, 115, ------ sayı dizisini rekürans bağıntısı haline getiriniz.

$1+3+2$

$3+6+4$

$6+13+8$

$$\boxed{\begin{aligned} a_0 &= 1 \\ a_1 &= 3 \end{aligned}}$$

$$\boxed{a_n = a_{n-1} + a_{n-2} + 2^{n-1}, \quad n \geq 2}$$

$a_2 = 2a_1 + a_0 \qquad a_2 = 7$

$a_3 = 2a_2 + a_1 \qquad a_3 = 17$

Soru: Sadece 0 ve 1 'lerden oluşan n birim uzunluğunda sayı dizileri oluşturuluyor. Buna göre, bu sayı dizilerinden n uzunluğunda olup "00" içerenlerin sayısını veren rekürans bağıntısını başlangıç koşullarını da belirleyerek yazınız.

$n=0$ birim $\longrightarrow$ dizi yok $\qquad a_0 = 0$

$n=1$ " $\longrightarrow$ 0, 1 $\qquad a_1 = 0$

$n=2$ " $\longrightarrow$ 00, 01, 10, 11 $\qquad a_2 = 1$

$n=3$ " $\longrightarrow$ 000, 001, 010, 100, 011, 101, 110, 111 $\qquad a_3 = 3$

$n=4$ " $\longrightarrow$

0000 - 0011 - 0111
0001 - 0101 1011
0010 - 1001 - 1101
0100 - 0110 1110
1000 - 1010 1111
1100 -

$a_4 = 8$

0, 0, 1, 3, 8, ...

$$a_n = 2.a_{n-1} + a_{n-2} + 1 \qquad n \geq 2$$

$$a_1 = 0$$

$$a_0 = 0$$

Problemlerin Karmaşıklığı

Çözülebilir Problemler (Tractable)

- Polinomsal en kötü durum karmaşıklığına sahip bir algoritma kullanarak çözülebilen bir problem **çözülebilir** (tractable) olarak adlandırılmaktadır.
- Çünkü algoritmanın, nispeten kısa bir zaman içinde makul büyüklükte veriye sahip bir probleme çözüm getireceği beklenmektedir.
- Bununla birlikte, büyük–*O* tahminindeki polinomlar yüksek dereceye sahipse (100. derece gibi) veya katsayılar aşırı büyükse, algoritmanın problemi çözmesi oldukça uzun zaman alabilir.
- Sonuç olarak, polinomsal en kötü durum zaman karmaşıklığına sahip bir algoritma kullanarak çözülebilen bir problemin nispeten küçük veri değerlerinde bile makul zamanda çözülebilmesi garanti edilemez.
- Pratikte bu tür tahminlerdeki polinomların derece ve katsayıları genellikle küçüktür.

Çözülemez Problemler (Intractable)

- En kötü durum polinomsal zaman karmaşıklığına sahip bir algoritma kullanılarak çözülemeyen problemler **çözülemez** (intractable) olarak adlandırılmaktadır.
- En kötü duruma sahip bir problemin çözümü, çok küçük girdi değerleriyle bile her zaman olmasa da genellikle aşırı miktarda zamana ihtiyaç duyabilmektedir.
- Bununla birlikte, pratikte bazı en kötü durum zaman karmaşıklığına sahip algoritmaların bir problemi, birçok durumda kendisine özgü en kötü durumundan çok daha hızlı çözebildiği durumlar vardır.
- Muhtemelen çok az sayıdaki durumların makul zamanda çözülememesine izin vermeye istekli olduğumuzda, ortalama durum zaman karmaşıklığı, bir algoritmanın bir problemi ne kadar uzun zamanda çözeceğinin en iyi ölçüsüdür.
- Endüstride önemli olan birçok problem çözülemez olarak düşünülmektedir ancak uygulamada, aslında günlük yaşamın getirdiği tüm girdi durumları için çözülebilmektedir.
- Pratik uygulamada karşılaşılan çözülemez problemlerin üstesinden gelmenin bir diğer yolu da problemin kesin çözümünü bulmak yerine yaklaşık çözümler aramaktır.
- Bu tür yaklaşık çözümler bulmakta hızlı algoritmaların varlığı işe yaramakta ve hatta kesin çözümden çok farklı olmama olasılığı bulunmaktadır.

Çözümü Bulunmayan Alg. (Unsolvable)

- Hiçbir algoritmanın onları çözemediği bazı problemler mevcuttur.
- Bu tür problemler **çözümü bulunamayan (unsolvable)** olarak adlandırılmaktadır
- Algoritma kullanarak **çözülebilen** problemlerin zıttı olarak ifade edilebilir.
- Çözümü bulunamayan problemlerin varlığına ilk kanıt, sonlanma (halting) probleminin çözülemez olduğunu gösteren büyük İngiliz matematikçi ve bilgisayar bilimcisi Alan Turing tarafından sağlanmıştır.

P ve NP Problem Sınıfları

- Çözülebilir problemlerin **P sınıfına (Class P)** ait olduğu söylenir.
- Polinomsal zamanda kontrol edilebilen bir çözüm için problemlerin **NP sınıfına (Class NP)** ait olduğu söylenmektedir.
- NP kısaltması, *belirli olmayan polinomsal zamanlı (nondeterministic polynominal time)* anlamına gelmektedir.

- Problemlerin herhangi birisi polinomial en kötü-durum zaman algoritmaları ile çözülebildiğinde, NP sınıfındaki tüm problemler de polinomsal en kötü-durum zaman algoritmaları ile çözülebilir. Bu özelliği taşıyan algoritmalara **NP-tam problemler** (**NP-complete problems**)
- **NP-Hard,** polinomsal zamanda bir çözümü olduğunu ispatlayamadığımız karar problemlerinin karmaşıklık sınıfıdır.

Kaynaklar

- Levitin "Introduction to the Design & Analysis of Algorithms," 3rd ed., Ch. 1 ©2012 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved
- https://www.javatpoint.com
- https://algorithms.tutorialhorizon.com
- https://www.tutorialspoint.com/