Ders 8:

- Özyinelemeli Fonksiyonlar: Recursive Functions: Özünü yineleyen, kendini yineleyen, kendini çağıran.
- Kendini çağırmanın 2 yolu var. Direkt f→f, dolaylı f→ g, g→ f
- Faktoriyel: n!=n*(n-1)!   F(n)=n*F(n-1)
    - Algoritmaya dönüştürelim

```
[T]=fakto(N)
T=N*fakto(N-1);
```

    - Ne olur? N → -∞ , düzeltmek için

```
[T]=fakto(N)
if N==1
    T=1;
else
    T=N*fakto(N-1);
end
```

fakto(5)=5*fakto(4)
fakto(5)=5*4*fakto(3)
fakto(5)=5*4*3*fakto(2)
fakto(5)=5*4*3*2*fakto(1)
fakto(5)=5*4*3*2*1
fakto(5)=5*4*3*2
fakto(5)=5*4*6
fakto(5)=5*24
fakto(5)=120

    - Özyinelemeli bir fonksiyonun her zaman stackoverflow'a sebep olmaması için
        - f(n), f(n)'i çağırmamalı
        - f(n)'de f'in çağrılmadığı bir bölüm olmalı
    - Bu 2 şarta sahip olursa yine de stackoverflow olabilir. İleride göreceğiz.
    - M=fakto(-3) dersek M ne olur?
- Üs alma $x^n=x*x^{n-1}$
    - Algoritmaya dönüştürelim

```
[T]=US(x,n)
if n==1
    T=x;
else
    T=x*US(x,n-1);
end
```

    - M=US(-4,2) dersek M ne olur?
    - M=US(4,-2) dersek M ne olur?
- Aşağıdaki fonksiyon ne iş yapar? M=f(12,4) dersek M ne olur?

```
[T]=f(a,b)
if b==0
    T=a;
else
    T=f(a+1,b-1);
end
```

    - b>0 için f(a,b)=a+b
- Aşağıdaki fonksiyon ne iş yapar?

```
[T]=f(dizi,b)
if b==1
    T=dizi(b);
else
    T=dizi(b)+f(dizi,b-1);
end
```

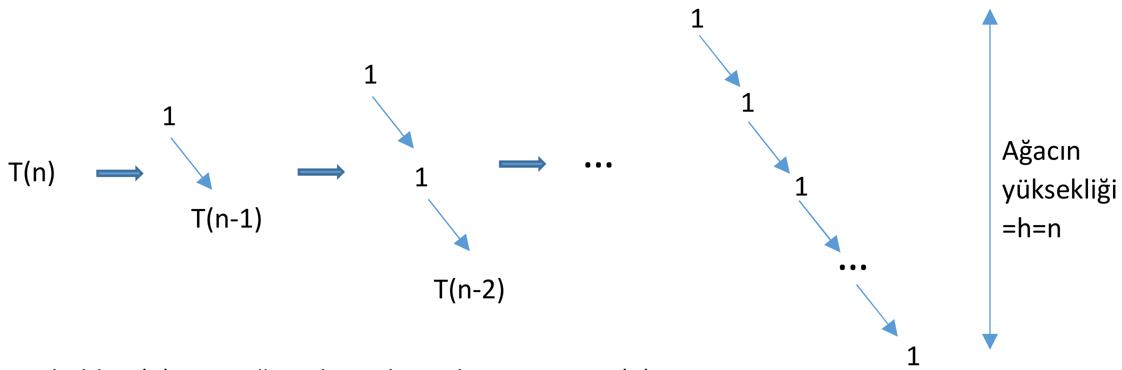
- b=dizinin eleman sayısı için, diziyi toplar

- Aşağıdaki fonksiyon ne iş yapar?

```
[T]=f(dizi,b,x)
if b<1
    T=-1;
else
    if dizi(b)==x
        T=b;
    else
        T=f(dizi,b-1,x);
    end
end
```

- b= dizinin eleman sayısı için, x dizide varsa yerini, yoksa -1 döndürür.

- Buraya kadar ki özyinelemeli algoritmalar bize bir şey kazandırmaz. Sadece özyinelemenin mantığını kavramamıza yardımcıdırlar.
  - Faktöriyelin karmaşıklığı, F(n)=1+F(n-1) işlem, 1 çarpma işlemi
  - Şimdi recursion tree'sini çizelim.

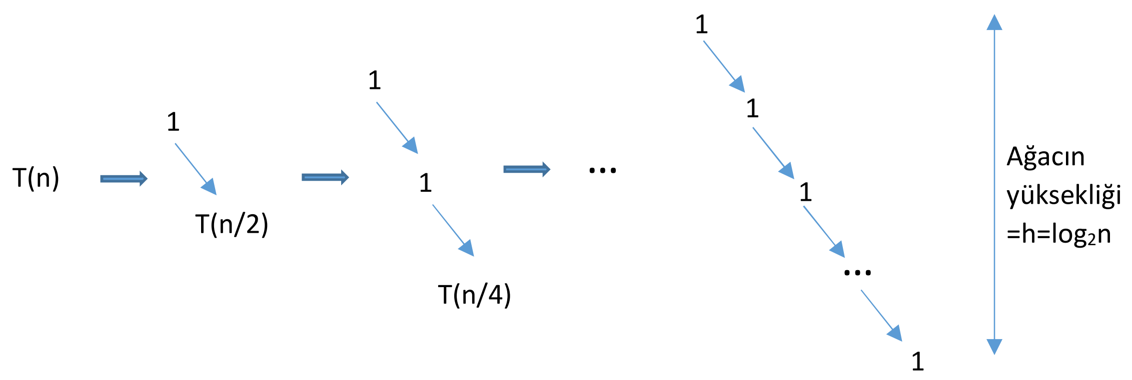


  - O halde T(n)=tüm ağaçtaki toplam işlem sayısı = O(n)
  - Zaten normal faktöriyel için de işlem sayısı O(n)'di. Yani zamandan kazanmadık.
  - Önceki diğer örnekler içinde durum aynı.
  - Ayrıca fonksiyon çağırmaktan dolayı zaman da kaybettik.

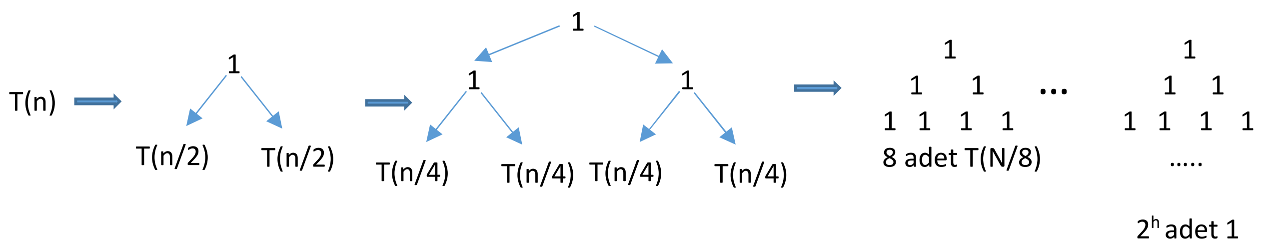
- Şimdi işe yarayan bir özyinelemeli bir fonksiyon görelim.
  - $$hpower(x,p) = \begin{cases} 1 \; if \; p == 0 \\ x * hpower\left(x, \downarrow\left(\frac{p}{2}\right)\right)^2 \; if \; p == tek \; // \; x^p = x^{\downarrow\left(\frac{p}{2}\right)} x^{\downarrow\left(\frac{p}{2}\right)} x \\ hpower\left(x, \downarrow\left(\frac{p}{2}\right)\right)^2 \; else \; // \; x^p = x^{\downarrow\left(\frac{p}{2}\right)} x^{\downarrow\left(\frac{p}{2}\right)} \end{cases}$$
  - ↓= aşağı yuvarlama
  - Algoritmaya çevirelim.

```
[T]=hpower(x,p)
if p==0
    T=1;
else
    k= hpower(x,floor(p/2));
    if mod(p,2)==1
        T=x*k*k;
    else
        T=k*k;
    end
end
```

  - T(n)=1+T(n/2), bunun reursion tree'sini çizelim.

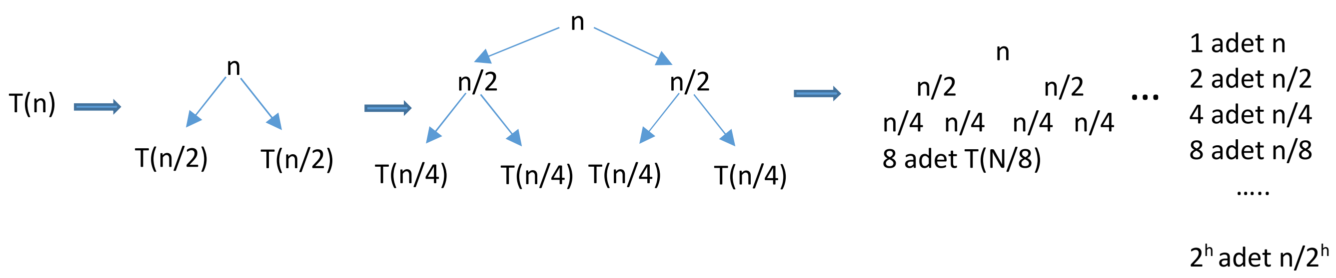


- O halde T(n)=tüm ağaçtaki toplam işlem sayısı = $\log_2 n$ tane 1 = $O(\log_2 n)$, işe yaradı. Çünkü normali O(n)'di. ☺
- `T=hpower(x,floor(p/2))* hpower(x,floor(p/2))` yazsaydık? T(n)=1+2T(n/2) olurdu. Bunun recursion tree'sini çizelim.



- $T(n) = \sum_{i=0}^{h} 2^i$, h = $\log_2 n$, $T(n) = \sum_{i=0}^{\log_2 n} 2^i = \frac{2^{(\log_2 n)+1}-1}{2-1} = 2 * 2^{\log_2 n} - 1 = 2N - 1 = O(n)$ ve normal üs almadan bir farkı olmazdı.

* Mergesort ve Quicksort için T(n)=2T(n/2)+n, karmaşıklığını bulalım.



- O halde T(n)=h adet n, h = $\log_2 n$, T(n)=n* $\log_2 n$, O(n)= n* $\log_2 n$,
- O halde karmaşıklığı $O(n^2)$ olan algoritmalara göre avantaj sağlıyorlar.