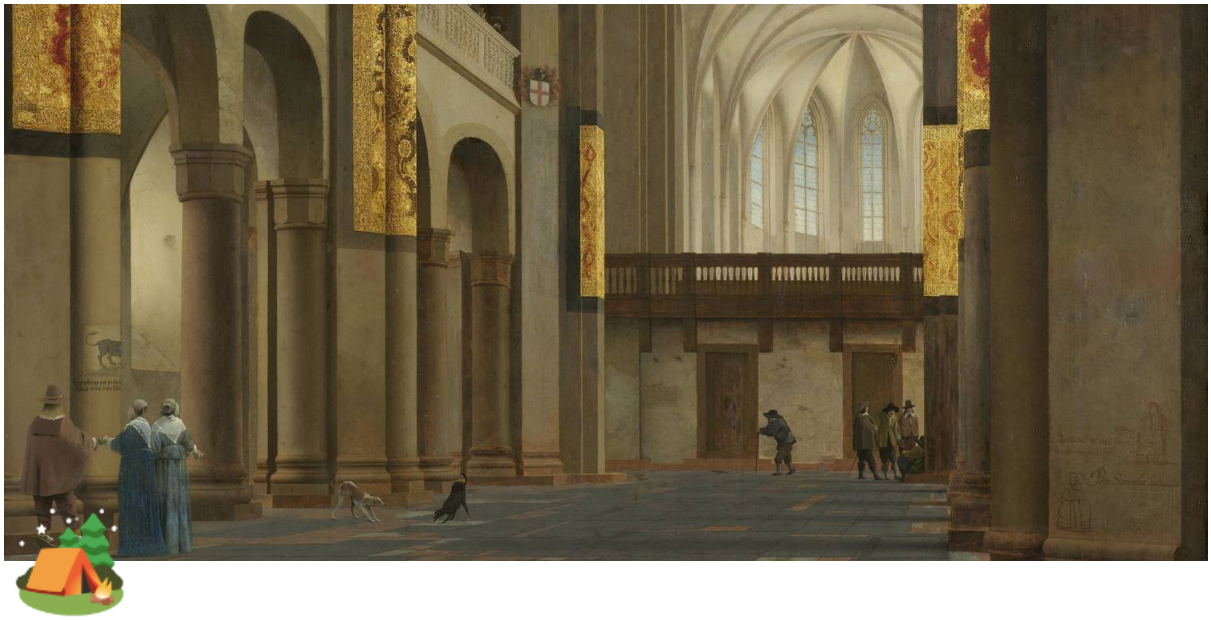

Algoritma analizi

Algoritma Büyüme Hızı

Bir algoritmanın yapacağı işlerinin miktarı işlediği verinin boyutuna bağlı olarak değişebilir. Bir algoritmanın üzerinde çalıştığı veri miktarına bağlı olarak iş miktarını gösteren fonksiyona **zaman karmaşıklığı (time complexity)** denir.

Örneğin bir algoritmanın zaman karmaşıklığı $f(n) = n + 3$ olsun. n'nin değerlerine göre algoritmanın yapacağı işlerin miktarı şu şekildedir:

$n = 1 \Rightarrow f(n) = f(1) = 4$

$n = 5 \Rightarrow f(n) = f(5) = 8$

Yukarıda hesapladığımız iki değeri birbirine bölersek $f(5)/f(1) = 8/4 = 2$ elde ederiz. Bu demek oluyor ki n değerini 5 kat artırırsam iş miktarım 2 kat artar. Her ne

kadar bilgisayardaki herbir işlemin maliyeti aynı olmasada çok büyük farklılıklar göstermezler. Algoritmaları kıyaslarken bütün işlemlerin maliyetini aynı kabul ederiz. Bu kabule dayanırsak, algoritmanın yapacağı iş miktarı artınca çalışma miktarı da onunla aynı oranda artar, diyebiliriz. Bu da demektir ki yukarıdaki algoritmanın iş miktarı 2 kat arttığı için çalışma miktarı da 2 kat artacacaktır.

Big O Gösterimi (Big O Notation)

Bir algoritmanın büyüme fonksiyonunu daha kısaca belirtmek isteyebiliriz. Bunun için Big O gösterimini kullanırız. Bu gösterim bize algoritmanın büyüme hızının üst sınırını çizmemizi sağlar. Şöyle ki;

$n \rightarrow$Veri Miktarı

$f(n) \rightarrow$Büyüme Hızı

Elimizde büyüm hızı $f(n) = 2n^2 + 3n + 5$ olan bir algoritmamız olsun. Bunun Big O gösterimi şudur: $O(n^2)$.

$n = 1 \Rightarrow f(n) = 10$ $\qquad$ $n = 1 \Rightarrow O(n^2) = 1$

$n = 10 \Rightarrow f(n) = 235$ $\qquad$ $n = 10 \Rightarrow O(n^2) = 100$

Bu algoritmadaki n, 10 kat artarsa $O(n^2)$'den 100 kattan daha fazla artmayacaktır. Bir algoritmanın büyüme fonksiyonunun Big O gösterimi fonksiyondaki en yüksek dereceli değişken katsayısı alınarak bulunur.

Örnek Big O Gösterimleri

- $O(1) \rightarrow$ Sabit süre.
- $O(\log_2 n)$
- $O(n)$
- $O(n \log_2 n)$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$
- $O(n!)$
- $O(n^n)$

Sıralama Algoritmaları

Kabarcık Sıralaması (Bubble Sort)

```
// Küçükten büyüğe sıralama
void BubbleSort(int dizi[], int eleman_sayisi){
  for(int i = 0; i < eleman_sayisi - 1; ++i){
    for(int j = eleman_sayisi - 1; j > i; --j){
      if(dizi[j] < dizi[j - 1]){
        int temp = dizi[j];
        dizi[j] = dizi[j - 1];
```

```
        dizi[j - 1] = temp;
      }
    }
  }
}
```

Seçerek Sıralama (Selection Sort)

```
// Küçükten büyüğe sıralama
void SelectionSort(int dizi[], int eleman_sayisi){
  int son_indeks = eleman_sayisi - 1;
  int en_kucuk;
  int temp;
  for(int i = 0; i < son_indeks; ++i){
    en_kucuk = i;
    for(int j = i + 1; j <= son_indeks; ++j){
      if(dizi[j] < dizi[en_kucuk])
        en_kucuk = j;
    }
    temp = dizi[i];
    dizi[i] = dizi[en_kucuk];
    dizi[en_kucuk] = temp;
  }
}
```

Hızlı Sıralama (Quick Sort)

```
// Küçükten büyüğe sıralama
void QuickSort(int dizi[], int baslangic, int son){
    if(!(baslangic < son)
        return;

    int pivot = dizi[baslangic];
    int sol = baslangic + 1;
    int sag = son;
    int temp;

    while(sol <= sag){
        while(dizi[sol] <= pivot)
            ++sol;
        while(dizi[sag] > pivot)
            --sag;
        if(sol < sag){
            temp = dizi[sol];
            dizi[sol] = dizi[sag];
            dizi[sag] = temp;

            ++sol;
            --sag;
        }
    }
```

```
    temp = dizi[sag];
    dizi[sag] = dizi[baslangic];
    dizi[baslangic] = temp;

    QuickSort(dizi, baslangic, sag - 1);
    QuickSort(dizi, sag + 1, son);
}
```

Birleştirme Sıralaması (Merge Sort)

```
// Küçükten büyüğe sıralama
void MergeSort(int dizi[], int baslangic, int son){
    if(baslangic == son)
        return;

    int orta = (baslangic + son) / 2;

    MergeSort(dizi, baslangic, orta);
    MergeSort(dizi, orta + 1, son);

    // Merge İşlemi
    int sol_eleman_sayisi = orta - baslangic + 1;
    int sag_eleman_sayisi = son - orta;
    int sol_dizi[sol_eleman_sayisi + 1];
    int sag_dizi[sag_eleman_sayisi + 1];

    for(int i = 0; i < sol_eleman_sayisi; ++i){
        sol_dizi[i] = dizi[baslangic + i];
    }

    for(int i = 0; i < sag_eleman_sayisi; ++i){
        sag_dizi[i] = dizi[orta + 1 + i];
    }

    sol_dizi[sol_eleman_sayisi] = INT_MAX; // Sonsuz değer
    sag_dizi[sag_eleman_sayisi] = INT_MAX; // Sonsuz değer

    for(int i = 0, j = 0, k = baslangic; k <= son; ++k){
        if(sol_dizi[i] <= sag_dizi[j]){
            dizi[k] = sol_dizi[i];
            ++i;
        }
        else{
            dizi[k] = sag_dizi[j];
            ++j;
        }
    }
}
```

Araya Sokma Sıralaması (Insertion Sort)

```
// Küçükten büyüğe sıralama
void InsertionSort(int dizi[], int eleman_sayisi){
    int temp;

    for(int i = 1; i < eleman_sayisi; ++i){
        for(int j = i; j > 0; --j){
            if(dizi[j] < dizi[j - 1]){
                temp = dizi[j];
                dizi[j] = dizi[j - 1];
                dizi[j - 1] = temp;
            }
            else
                break;
        }
    }
}
```

Ana Teorem (Master Theory)

Bu teorem, parçala ve yönet tipindeki algoritmaların analizinde kullanılır. Bu tipteki algoritmalardan birini analiz ettiğimizde aşağıda belirtilen $T(n)$ fonksiyonuna benzer bir fonksiyon bulduğumuzu varsayalım.

$$a, b, c \in R^{+} \rightarrow a \geq 1 \quad b > 1 \quad n \in N \rightarrow f(n) > 0$$

$$T(n) = a.T(\frac{n}{b}) + f(n)$$

Aşağıdaki 3 duruma göre $T(n)$ değeri bulunur.

$$c, \epsilon, k \in Q^{+} \rightarrow c < 1 \quad \epsilon > 0 \quad k \geq 0$$

$$f(n) = \mathrm{O}(n^{log_b a - \epsilon}) \quad \Rightarrow \quad T(n) = \Theta(n^{log_b a}) \tag{1}$$

$$f(n) = \Theta(n^{log_b a}.\log^{k} n) \quad \Rightarrow \quad T(n) = \Theta(n^{log_b a}.\log^{k+1} n) \tag{2}$$

$$f(n) = \Omega(n^{log_b a + \epsilon}) \quad \wedge \quad a.f(n/b) \leq c.f(n) \quad \Rightarrow \quad T(n) = \Theta(f(n)) \tag{3}$$

```
// Fibonacci Sayıları - Dinamik Programlama
int Fibonacci(int eleman){
    if(eleman < 0)
        return -1;
    else if(eleman == 0)
        return 0;
    else if(eleman == 1)
        return 1;
```

```
    int x = 0;
    int y = 1;

    for(int i = 2, temp; i <= eleman; ++i){
        temp = y;
        y += x;
        x = temp;
    }

    return y;
}
```

```
// Kombinasyon Hesabı - Dinamik Programlama
int Combination(int n, int k){
    if(n < k)
        return -1;
    else if(k == 0)
        return 1;

    k = (k > n/2) ? (n-k) : k;

    int kayma = n - k;

    int Comb[n + 1][k + 1];
    for(int i = 0; i <= n; ++i){
        int j = (i > kayma) ? (i - kayma) : 0;
        int z = (i<k) ? i : k;
        for(; j <= z; ++j){
            if(j == 0 || i == j)
                Comb[i][j] = 1;
            else{
                Comb[i][j] = Comb[i - 1][j - 1] + Comb[i - 1][j];
            }
        }
    }

    return Comb[n][k];
}
```