Algoritmaların Sınıflandırılması

- Algoritmalar, problemlerin **çözümü için uyguladıkları yönteme göre** sınıflandırılabilir.
- Algoritmaları sınıflandırmadaki temel amaç, problemlerin çözümünde başvurulabilecek değişik **metotları** ve **alternatifleri** tespit edebilmektir.

Algoritmaların Sınıflandırılması

a	Özyinelemeli Algoritmalar	Simple Recursive Algorithms
b	Geri İzlemeli Algoritmalar	Backtracking Algorithms
c	Böl ve Yönet Algoritmaları	Divide and Conquer Algorithms
d	Dinamik Programlama	Dynamic Programming
e	Açgözlü Algoritmalar	Greedy Algorithms
f	Kaba Kuvvet Algoritmaları	Brute Force Algorithms

Algoritmaların Sınıflandırılması

a	Özyinelemeli Algoritmalar	Simple Recursive Algorithms

- Kendisini **doğrudan** veya **dolaylı** olarak çağıran algoritmalara özyinelemeli algoritma adı verilir.
- Bu algoritmalarda, problemler daha **küçük ve basit** parçalara indirgenir.
- Küçük parçalar için oluşturulan **çözümlerin birleştirilmesiyle** ana problemin çözümü elde edilir.

```
int factorial(int n) {

if(n <= 1) {
    return 1;
}

    return n * (factorial(n-1));
}
```

Algoritmaların Sınıflandırılması

b	Geri İzlemeli Algoritmalar	Backtracking Algorithms

- Geri izlemeli algoritmalar, genellikle **optimizasyon** problemlerinde kullanılan, problem çözümünde **tüm olasılıkları deneyen** algoritmalardır.
- Bu algoritmalarda **çözüm kademeli şekilde** oluşturulur.
- Algoritma çözüm aşamasında ilerlerken, **olası** çözüm yollarının **hepsini** deneyerek bir sonraki adıma geçmeye çalışır.
- Algoritmanın denediği çözüm yolundan **sonuç alınamazsa**, algoritma bir **önceki** adımda bulunan **diğer olası çözüm yollarına geri döner**.

- **Sudoku:** 9 x 9'luk bir tablonun her satır ve sütununda 1'den 9'a kadar sayıların olması gerekmektedir. **Bazı değerlerin dolu olarak verildiği bulmaca nasıl çözülür?**
- **Sekiz Vezir Problemi:** Sekiz vezir, bir satranç tahtasına birbirlerine hamle yapamayacak şekilde **nasıl yerleştirilir**?

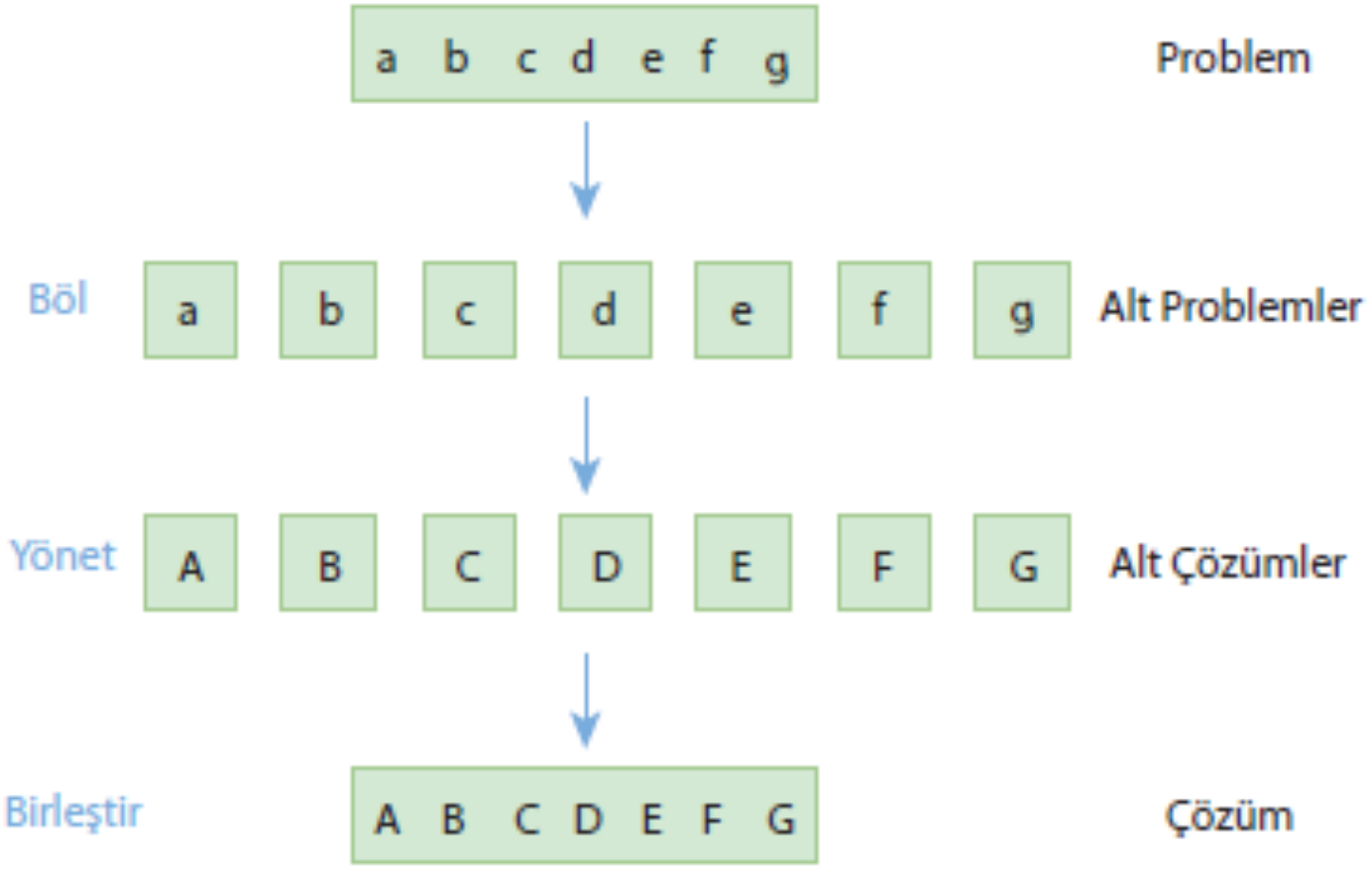
Algoritmaların Sınıflandırılması

C	Böl ve Yönet Algoritmaları	Divide and Conquer Algorithms

- Böl ve yönet algoritmaları, problemlerin mümkün olan **en küçük alt parçalara** ayrıldığı, her bir alt parçanın diğerlerinden **bağımsız** şekilde çözüldüğü algoritmalardır.
- Problemin genel çözümü elde edilirken alt parçalara ait çözümler **belirli bir sırayla** bir araya getirilir.
- Böl ve yönet algoritmaları, genellikle **üç ana aşamadan** meydana gelmektedir:
- **Bölme (Divide):** Problemin daha **küçük parçalara ayrıldığı** aşamadır. Problem daha küçük alt parçalara **bölünemeyecek hale gelene kadar**, özyinelemeli bir yaklaşımla bölme işlem gerçekleştirilir.
- **Yönetme (Conquer):** Problemin alt parçalarının, birbirlerinden **bağımsız** olarak **çözüldüğü** aşamadır.
- **Birleştirme (Merge):** Problemin alt parçalarına ait çözümlerin, **özyinelemeli bir yaklaşımla birleştirildiği** aşamadır.

Algoritmaların Sınıflandırılması

C	Böl ve Yönet Algoritmaları	Divide and Conquer Algorithms



Algoritmaların Sınıflandırılması

d	Dinamik Programlama	Dynamic Programming

- Dinamik programlama, **karmaşık** problemleri **küçük** parçalar halinde çözen, elde edilen **sonuçları** bilgisayar hafızasında bir **veri yapısında saklayan**, **genel çözümü** elde ederken de veri yapılarında **saklanan sonuçları kullanan** bir programlama yöntemidir.
- Bir problemin dinamik programlama ile çözülebilmesi için problemin **alt parçalara ayrılabilmesi** ve genel çözümün bu alt parçalardan **oluşturulabilmesi** gerekmektedir.
- Dinamik programlama yaygın olarak **optimizasyon** problemlerinde kullanılır.
- Daha önce özyinelemeli algoritma aracılığıyla programlanan faktöriyel hesabını dinamik programlama ile de yapabiliriz.

Algoritmaların Sınıflandırılması

d	Dinamik Programlama	Dynamic Programming

```
#include<stdio.h>

int memory[100] = {1, 1};

int factorial(int n) {
    if(n <= 1) {
        return 1;
    }
    else {
        int i;

        for(i=2; i<=n; i++) {
           if(memory[i] == 0) {
                memory[i] = i * memory[i-1];
           }
        }

        return memory[n];
    }
}

int main(void) {
      int n;

      for(n=1; n<10; n++) {
            printf("%d! = %d\n", n, factorial(n));
      }

      getch();
      return 0;
}
```

Algoritmaların Sınıflandırılması

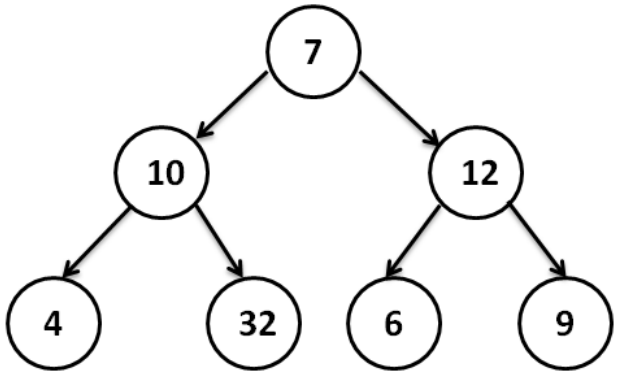
e	Açgözlü Algoritmalar	Greedy Algorithms

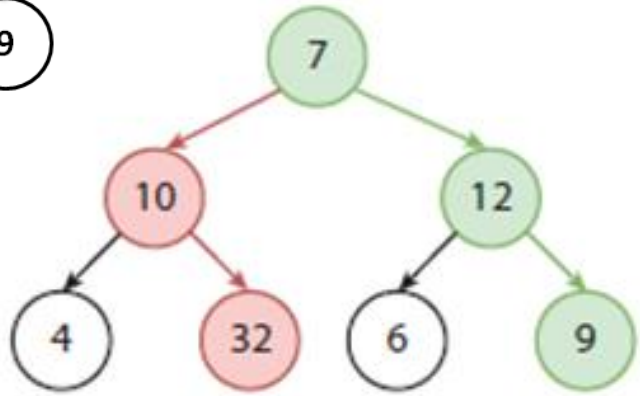
- Bir problem için **mümkün olan en doğru çözümü** hedefleyen algoritmalara **açgözlü** algoritmalar adı verilir.
- Açgözlü algoritmalarda yerel olarak **optimum sonuç** elde edilirken, bulunan sonuç her zaman için **en iyi çözüme karşılık gelmeyebilir**.
- Açgözlü algoritmalar ile problem çözümündeki temel yaklaşım, problemin **küçük bir alt kümesi** için **çözüm oluşturmak** ve bu çözümü **problemin geneline** yaymaktır.
- Algoritma içerisinde **yapılan bir seçim**, o an için **doğru olsa bile** sonraki seçimlerde **olumsuz etki** yapabilir.

Algoritmaların Sınıflandırılması

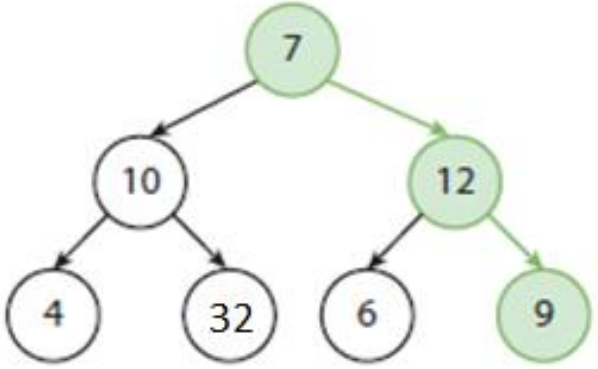
e	Açgözlü Algoritmalar	Greedy Algorithms

- Bir şehirden yola çıkan **gezginin <u>en fazla</u>** seyahat edeceği **<u>yolu</u>** hesaplama problemi, nasıl çözülebilir?





b. Aç gözlü algoritma ile optimum çözüm (en iyi çözüm değil)



a. Aç gözlü algoritma ile en iyi çözüm

Algoritmaların Sınıflandırılması

f	Kaba Kuvvet Algoritmaları	Brute Force Algorithms

- Bir problemin çözümü aşamasında, **kabul edilebilir** bir çözüm elde edene kadar **tüm olasılıkları deneyen** algoritmalara kaba kuvvet algoritmaları denir.
- Kaba kuvvet algoritmaları, genellikle problemin tanımından yola çıkarak **en basit çözüm yolunu uygular** ve **rahatlıkla** kodlanır.
- Fakat bu algoritmalarda **çok fazla işlem yapılır** ve çözüm yolu **optimumdan uzaktır**.
- Problemdeki veri hacmi **büyüdükçe**, kaba kuvvet algoritması ile çözüm sansı da **azalır**.

Algoritmaların Sınıflandırılması

e	Açgözlü Algoritmalar	Greedy Algorithms

- Bir liste içerisinde **eleman aramak**, kaba kuvvet algoritmaların kullanımıyla çözülebilecek problemlere bir örnektir.
- Listenin tüm elemanları **sırayla kontrol edilerek**, aranan elemanın **listede olup olmadığına** bakılabilir.
- Listenin **eleman sayısı arttıkça**, kaba kuvvet algoritmasının çalışma süresi ve yaptığı karşılaştırmalar da **artacaktır**.