



# BLM 2425 ALGORİTMA ANALİZİ

## ÖDEV 4

Backtracking

**Öğrenci Adı:** Sinem SARAĞ

**Öğrenci Numarası:** 22011647

**Dersin Eğitmeni:** M. Elif KARSLIĞİL

**Açıklama Video Linki:** [https://youtu.be/Taw\\_0txyPj8](https://youtu.be/Taw_0txyPj8)

# İçindekiler

Problemin Çözümü.....	4
1. Brute Force.....	4
2. Optimized1.....	4
3. Optimized2.....	4
4. Backtracking.....	5
5. Kullanılan Ana Fonksiyonlar.....	5
Karşılaşılan Sorunlar.....	6
1- Tahta Tasarımında Planlama Hatası.....	6
2- Tehdit Alanlarının Eksik Hesaplanması.....	6
Ekran Çıktıları.....	6
Giriş Ekranı, n ve Modun Kullanıcıdan Alınması.....	6
n = 4.....	7
Detailed Mode: Brute Force.....	7
Detailed Mode:Optimized1.....	7
Detailed Mode: Optimized2.....	7
Detailed Mode: Backtracking.....	7
All Approaches.....	7
n = 5.....	8
Detailed Mode: Brute Force.....	8
Detailed Mode: Optimized1.....	8
Detailed Mode: Optimized2.....	8
Detailed Mode: Backtracking.....	9
All Approaches.....	9
n = 6.....	9
Detailed Mode: Brute Force.....	9
Detailed Mode: Optimized1.....	9
Detailed Mode: Optimized2.....	10
Detailed Mode: Backtracking.....	10
All Approaches.....	10
n = 7.....	11
Detailed Mode: Brute Force.....	11
Detailed Mode: Optimized1.....	11
Detailed Mode: Optimized2.....	12
Detailed Mode: Backtracking.....	12
All Approaches.....	13
n = 8.....	13

Detailed Mode: Brute Force .....	13
Detailed Mode: Optimized1 .....	13
Detailed Mode: Optimized2 .....	14
Detailed Mode: Backtracking.....	14
All Approaches .....	14
n = 9.....	15
Detailed Mode: Optimized1 .....	15
Detailed Mode: Optimized2 .....	15
Detailed Mode: Backtracking.....	16
All Approaches .....	16
n = 10.....	16
Detailed Mode: Optimized2 .....	16
Detailed Mode: Backtracking.....	17
AllApproaches .....	17

## Problemin Çözümü

Bu program, N-Queens problemini çözmek için verilen algoritmaları kullanarak girilen n değerleri için bütün çözümleri bulmayı amaçlamaktadır. Kullanıcı, çözümün detaylarını görmek için iki farklı mod seçeneği arasından seçim yaparak başlar: "Detaylı Mod" ve "Tüm Yaklaşımlar".

Başlangıçta, kullanıcıdan n değerini (yani tahtanın boyutu) 3'ten büyük olacak şekilde istenir. Daha sonra kullanıcıya iki seçenek sunulur:

Detaylı Mod: Brute force, optimize1, optimize2 ve backtracking modları arasından seçilen yöntemi kullanarak vezir problemini çözer ve tüm çözümleri ekranda gösterir.

Tüm Yaklaşımlar: Tüm algoritmaları sırasıyla çalıştırarak her bir algoritma için çözüm süresini gösterir, ancak çözüm sonuçlarını ekranda göstermez.

Bu şekilde bir ayrıma gidilmesinin temel sebebi özellikle n değerleri arttıkça, yazma işleminin problemin çözüm süresini önemli derecede etkilemesidir. Yazma işlemleri tüm yaklaşımlar modu içerisinde kapatılarak algoritmanın gerçek çalışma süresinin ölçülmesi amaçlanmıştır.

Kullanıcının girdiği moda bağlı olarak program 4 farklı çözüm yaklaşımından birini kullanır. Her bir yaklaşımın işleyişi alt başlıklar içerisinde detaylandırılacak ve bu algoritmaları uygulamak için yazılmış fonksiyonlar farklı bir alt başlık altında detaylıca ele alınacaktır:

### 1. Brute Force

Brute Force yaklaşımında, algoritma her bir hücre herhangi bir ön koşul olmaksızın kontrol edilerek tüm olasılıklar denenir. Yani, en başta sırasıyla tüm vezirler ilk satıra yerleştirilerek başlanır ve en sondaki vezir bir alt satırın ilk hücresine geçecek şekilde devam ederek tüm vezirler bu örüntü ile hareket eder. Bu yöntem, küçük n değerleri için uygun bir çözüm sunabilir ancak büyük n değerlerinde zaman karmaşıklığı oldukça yüksek olup, çözüm süresi artmaktadır. Bu yaklaşımda, her bir vezir koyulduktan sonra fonksiyon tekrardan kendini çağırarak bir sonraki veziri sıradaki boş alana yerleştirir. Her çağırma işleminde yerleştirilen toplam vezir sayısı kontrol edilir ve n adet vezir yerleştirildiğinde checkQueensSafe fonksiyonu çağırılarak vezirlerin birbirlerinin etki alanında olup olmadıkları kontrol edilir. Vezirler uygun bir şekilde yerleştirilmişse çözüm ekrana yazdırılarak çözüm sayısı bir artırılır.

### 2. Optimized1

Bu yaklaşımda, yalnızca aynı satırda tehdit altında olan hücreler dikkate alınmaktadır. Yani, başlangıçta her satıra birer tane gelecek şekilde vezirler yerleştirilir ve en sondaki vezir, bulunduğu satırın hücreleri içerisinde hareket eder. Tüm vezirler bu şekilde, bir satırda sadece bir vezir olma kuralını koruyarak farklı konumlara yerleştirilir. Bir vezir bir konuma yerleştiğinde aynı satıra başka bir vezirin yerleşmemesi için updateAreas fonksiyonu ile vezirin tehdit ettiği hücreler işaretlenir. Sonraki adımlarda yerleşecek vezirler işaretli hücrelere yerleşmezler. Bu sayede her bir satıra bir vezir yerleştirilmiş olur. Yerleştirilen vezir sayısı n değerine eşit olduğunda yerleştirme checkQueensSafe fonksiyonu ile kontrol edilerek çözümün doğru olup olmadığı tespit edilir. Çözüm doğruysa ekrana bastırılır ve çözüm sayısı bir artırılır. Bu yaklaşım, brute force yaklaşımına göre süre ve deneme sayısında önemli bir kazanç sağlamaktadır.

### 3. Optimized2

Bu yaklaşımda, yalnızca aynı satıra ek olarak sütunlardaki tehditler de dikkate alınmaktadır. Yani, başlangıçta bir vezir yerleştirilir ve bu vezirle aynı satır veya sütunda olmayacak şekilde diğer vezirler eklenir. Tüm vezirler bu şekilde, bir satırda ve bir sütunda sadece bir vezir olma kuralını koruyarak farklı konumlara yerleştirilir. Bir vezir bir konuma yerleştiğinde aynı satır ve sütuna

başka bir vezirin yerleşmemesi için `updateAreas` fonksiyonu içerisinde `Optimized1`'de yapılan satır işaretlemeye ek olarak sütun işaretleme de yapılır ve vezirin tehdit ettiği hücreler işaretlenir. Yine benzer şekilde sonraki adımlarda yerleşecek vezirler işaretli hücrelere yerleşmezler. Bu sayede her bir satıra bir vezir yerleştirilmiş olur. Yerleştirilen vezirler `checkQueensSafe` fonksiyonu ile kontrol edilir.

#### 4. Backtracking

Bu yaklaşımda, çözüm yolunun yanlış olduğu tespit edilirse, geri izleme yaparak başka bir olasılık denemesi sağlar. Bu yaklaşım, gereksiz hesaplamaları ortadan kaldırarak daha hızlı çözümler bulur. Bir vezir konulduktan sonra diğer vezirlerin gelemeyeceği tüm alanlar işaretlenir ve diğer vezir için uygun bir alan aranır. Tahtanın sonuna gelindiğinde `n` sayısı yerleştirilen vezirlere eşit olmadığında gidilen yolun yanlış olduğuna karar verilir. Gidilen yolun yanlış olduğu anlaşıldığında ise geri izleme yapılarak başka bir olasılık denenir. `n` sayısı yerleştirilen vezir sayısına eşit olduğunda ise çözümün doğru olduğu `checkQueensSafe` fonksiyonu çağrılmadan anlaşılabilir çünkü vezirler yerleştirilirken zaten birbirlerinin etki alanlarında olmayacak şekilde konumlandırılırlar.

#### 5. Kullanılan Ana Fonksiyonlar

Solve: `solve` fonksiyonu, çözümün temel işleyişini yöneten fonksiyondur. Bu fonksiyon, her bir vezir için uygun, başka bir vezirin etki alanında olmayan bir hücreyi bularak yerleştirir ve çözümün doğru olup olmadığını kontrol eder. Her bir hücre `solve` fonksiyonu tarafından gezilerek 0 olup olmadığı kontrol edilir. 0 hücreler boş ve vezir etkisi alanında olmayan hücrelerdir. 0 olarak bulunan bir hücreye vezir yerleştirilerek etki alanları `updateAreas` fonksiyonu çağrılarak işaretlenir ve sıradaki vezirin yerleştirilmesi için fonksiyon tekrardan çağrılır. Yerleştirilen vezir sayısı `n` değerine eşit olduğunda vezirlerin yerleşimi `checkQueensSafe` ile kontrol edilir ve çözüm uygunsa ekrana yazdırılır.

updateAreas: `updateAreas` fonksiyonu, tahtaya bir vezir yerleştirildiğinde veya kaldırıldığında, vezirin tehdit alanlarını güncelleyen bir yardımcı fonksiyondur. Bu fonksiyon, verilen satır ve sütun parametreleri doğrultusunda, vezirin etki alanına giren hücreleri, programın çalıştığı moda göre belirleyerek board üzerindeki etki alanı belirlenir. `Optimize1` için aynı satırda; `optimize2`'de `optimize1`'e ek aynı sütunda ve `Backtracking`'te `Optimize2`'ye ek olarak çaprazlarda güncellemeler yapılır. Bu güncellemeler sırasında vezirin bulunduğu hücre hariç tutulur. `delta` parametresi 1 olduğunda vezir konulduğu anlamına gelmekte ve tehdit değerleri 1 artırılırken, -1 olduğunda vezirin kaldırıldığı anlaşılır ve tehdit ettiği hücrelerdeki tehdit değeri 1 azaltılır. Artırma ve azaltma işlemleri bu parametrenin direkt etki alanındaki hücrelere eklenmesi ile gerçekleştirilir. Bu sayede, tehdit alanlarının güncel tutulması ve sonraki adımlarda güvenli hücrelerin belirlenmesi sağlanır.

isQueensSafe: `isQueensSafe` fonksiyonu, tahtadaki vezirlerin birbirini tehdit edip etmediğini kontrol eden bir doğrulama fonksiyonudur. Bu fonksiyon, vezirlerin konumlarını dikkate alarak aynı satır, sütun ve çaprazlarda başka bir vezirin olup olmadığını kontrol eder. Board üzerinde her bir vezir için bu kontroller yapılır ve herhangi bir çakışma tespit edilirse çözümün geçersiz olduğu döndürülür. Eğer hiçbir vezir birbirini tehdit etmiyorsa fonksiyon 1 döner, aksi halde 0 döner. Bu fonksiyon `solve` fonksiyonu içerisinde, `n` adet olduğu bilinen vezirlerin birbirlerini tehdit edip etmediğini anlamak için kullanılır.

# Karşılaşılan Sorunlar

## 1- Tahta Tasarımında Planlama Hatası

Problemi çözmek için başlangıçta, yerden tasarruf sağlamak amacıyla bir  $2 \times N$  boyutunda matris oluşturuldu. Bu matris, vezirlerin yalnızca satır ve sütun konumlarını tutuyordu. Ancak bu yaklaşım, vezirlerin tehdit ettiği alanların belirlenmesinde önemli bir zorluk yarattı. Çünkü tahtanın tam bir temsili olmadığından, tehdit bölgelerinin kontrol edilmesi karmaşık hale geldi ve bu durum, checkQueensSafe fonksiyonunun performansını olumsuz etkiledi.

Bu nedenle problem, tahtanın tam bir temsili tutacak şekilde yeniden tasarlanarak çözüldü. Yeni yapıda, tehdit alanlarının kontrolü ve güncellenmesi için tahta matrisi oluşturuldu ve vezirler üzerine yerleştirilerek fonksiyonlara doğrudan iletildi. Böylece tehdit alanlarının hesaplanması ve doğrulanması daha basit ve hızlı hale getirildi. Bu düzenleme, özellikle yüksek  $N$  değerlerinde performans artışı sağladı.

## 2- Tehdit Alanlarının Eksik Hesaplanması

İlk olarak tehdit alanlarının belirlenmesi sırasında, her bir hücrenin tehdit altında olup olmadığına göre 1 ya da 0 değerleri kullanıldı. Ancak bu yöntem, bir hücrenin birden fazla vezir tarafından tehdit edilmesi durumunda hatalara neden oldu. Örneğin, bir vezirin kaldırılması sırasında, tehdit alanlarının sıfırlanması diğer vezirlerin o hücre üzerindeki etkisini de kaldırıyordu. Bu durum, yanlış tehdit hesaplamalarına ve hatalı çözümlere yol açtı.

Bu sorunun çözümü için tehdit edilen hücrelere 1 ve 0 değerleri koymak yerine, her bir hücredeki tehdit sayısını takip edecek şekilde artırma ve azaltma yöntemi benimsendi. Böylece, bir hücre birden fazla vezir tarafından tehdit edilse bile, tehdit sayısı doğru şekilde hesaplanıp yönetildi. Bu yaklaşım, tehdit alanlarının daha hassas ve doğru bir şekilde kontrol edilmesini sağladı.

## Ekran Çıktıları

Bu bölümde  $n = 4$  ile başlanarak  $n = 10$ 'a kadar ekran çıktılarına yer verilecektir. Her bir  $n < 8$  değeri için çıktısında önce Detailed Mode çalıştırılarak 4 modun bulunduğu çözümler ayrı ayrı gösterilecek en sonunda ise All Approaches Mode kullanılarak her bir algoritmanın süresi verilecektir.  $n=8$ ,  $n = 9$  ve  $n = 10$  durumlarında ise çözüm sayısı çok fazla olduğundan detailed mod çalıştırıldıktan sonra elde edilen tüm çözüm çıktıları verilmeyecek, sadece rastgele seçilmiş birkaç örnek konulacak ve all Approaches moduna ait ekran görüntülerine yer verilecektir. Ayrıca  $n = 9$  için brute force ve  $n = 10$  için brute forcea ek olarak optimized1 algoritmalarının işlenmesi çok uzun sürdüğünden rastgele çözümler ve süreler verilirken bahsedilen algoritmalar atlanacaktır.

## Giriş Ekranı, n ve Modun Kullanıcıdan Alınması

```
Please enter the n value: 4
Please select a mode:
1: Detailed mode (shows the solution results)
2: All approaches (shows only the time taken for each solution, no results are shown)
Enter your choice: 2
```

```
Please enter the n value: 4
Please select a mode:
1: Detailed mode (shows the solution results)
2: All approaches (shows only the time taken for each solution, no results are shown)
Enter your choice: 1
```

```
Please select an approach:
0: Brute Force
1: Optimized1
2: Optimized2
3: Backtracking
Enter your choice: 1
```

n = 4

### *Detailed Mode: Brute Force*

```
Timer started, processing...

-----
* Q * *
* * * Q
Q * * *
* * Q *
-----

* * Q *
Q * * *
* * * Q
* Q * *
-----

All solutions have been calculated, timer stopped.

Process completed in 0.0090 seconds (0.504000 milliseconds) with print statements. Found 2 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
(Use the 'All approaches' mode to measure time without the impact of prints.)
```

### *Detailed Mode: Optimized1*

```
Timer started, processing...

-----
* Q * *
* * * Q
Q * * *
* * Q *
-----

* * Q *
Q * * *
* * * Q
* Q * *
-----

All solutions have been calculated, timer stopped.

Process completed in 0.0070 seconds (7.166000 milliseconds) with print statements. Found 2 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
(Use the 'All approaches' mode to measure time without the impact of prints.)
```

### *Detailed Mode: Optimized2*

```
Timer started, processing...

-----
* Q * *
* * * Q
Q * * *
* * Q *
-----

* * Q *
Q * * *
* * * Q
* Q * *
-----

All solutions have been calculated, timer stopped.

Process completed in 0.0050 seconds (5.613600 milliseconds) with print statements. Found 2 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
(Use the 'All approaches' mode to measure time without the impact of prints.)
```

### *Detailed Mode: Backtracking*

```
Timer started, processing...

-----
* Q * *
* * * Q
Q * * *
* * Q *
-----

* * Q *
Q * * *
* * * Q
* Q * *
-----

All solutions have been calculated, timer stopped.

Process completed in 0.0040 seconds (4.154600 milliseconds) with print statements. Found 2 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
(Use the 'All approaches' mode to measure time without the impact of prints.)
```

### *All Approaches*

```
Starting Brute Force, timer started. This might take some time..
Brute Force took: 0.0000 seconds (0.110300 milliseconds). Found 2 solutions.

Starting Optimized1, timer started.
Optimized1 took: 0.000000 seconds (0.046100 milliseconds). Found 2 solutions.

Starting Optimized2, timer started.
Optimized2 took: 0.0000 seconds (0.027700 milliseconds). Found 2 solutions.

Starting Backtracking, timer started.
Backtracking took: 0.0000 seconds (0.015200 milliseconds). Found 2 solutions.
```

n = 5

### Detailed Mode: Brute Force

```
* * Q * *
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
All solutions have been calculated, timer stopped.
Process completed in 0.0588 seconds (58.820980 milliseconds) with print statements. Found 10 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
```

```
Timer started, processing...
-----
Q * * * *
* * Q * *
* * * * Q
* Q * * *
* * * Q *
-----
Q * * * *
* * * Q *
* Q * * *
* * * Q *
* * Q * *
-----
* Q * * *
* * * Q *
Q * * * *
* * Q * *
* * * * Q
-----
* Q * * *
* * * Q *
* * * * Q
Q * * * *
* * * Q *
```

### Detailed Mode: Optimized1

```
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
All solutions have been calculated, timer stopped.
Process completed in 0.0490 seconds (47.513200 milliseconds) with print statements. Found 10 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
```

```
Timer started, processing...
-----
Q * * * *
* * Q * *
* * * * Q
* Q * * *
* * * Q *
-----
Q * * * *
* * * Q *
* Q * * *
* * * Q *
* * Q * *
-----
* Q * * *
* * * Q *
Q * * * *
* * Q * *
* * * * Q
-----
* Q * * *
* * * Q *
* * * * Q
Q * * * *
* * * Q *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
* * Q * *
```

### Detailed Mode: Optimized2

```
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
* * Q * *
* * Q * *
* * Q * *
* * Q * *
-----
All solutions have been calculated, timer stopped.
Process completed in 0.0470 seconds (47.636600 milliseconds) with print statements. Found 10 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
```

```
Timer started, processing...
-----
Q * * * *
* * Q * *
* * * * Q
* Q * * *
* * * Q *
-----
Q * * * *
* * * Q *
* Q * * *
* * * Q *
* * Q * *
-----
* Q * * *
* * * Q *
Q * * * *
* * Q * *
* * * * Q
-----
* Q * * *
* * * Q *
* * * * Q
Q * * * *
* * * Q *
```



## Detailed Mode: Backtracking

```
+ + Q + +
+ + + Q +
+ + + + Q
+ + + + + Q

+ + Q + +
+ + + Q +
+ + + + Q
+ + + + +

+ + + Q +
+ + + + Q
+ + + + + Q
+ + + + +

+ + + + Q
+ + + + + Q
+ + + + +
+ + + + +

+ + + + Q
+ + + + +
+ + + + +
+ + + + +

+ + + + Q
+ + + + +
+ + + + +
+ + + + +

All solutions have been calculated, timer stopped.
Process completed in 0.0470 seconds (47.436000 milliseconds) with print statements. Found 10 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
```

```
Timer started, processing...

Q * * * *
* * Q * *
* * * * Q
* Q * * *
* * * Q *

Q * * * *
* * * Q *
* Q * * *
* * * * Q
* * Q * *

* Q * * *
* * * Q *
Q * * * *
* * Q * *
* * * * Q

* Q * * *
* * * Q *
Q * * * *
* * Q * *
* * * * Q

* Q * * *
* * * Q *
Q * * * *
Q * * * *
* * * Q *
```

## All Approaches

```
Starting Brute Force, timer started. This might take some time..
Brute Force took: 0.0020 seconds (1.786200 milliseconds). Found 10 solutions.

Starting Optimize1, timer started.
Optimize1 took: 0.001000 seconds (0.592500 milliseconds). Found 10 solutions.

Starting Optimize2, timer started.
Optimize2 took: 0.0000 seconds (0.161600 milliseconds). Found 10 solutions.

Starting Backtracking, timer started.
Backtracking took: 0.0000 seconds (0.076600 milliseconds). Found 10 solutions.
```

n = 6

## Detailed Mode: Brute Force

```
Timer started, processing...

+ + Q + + +
+ + + Q + +
+ + + + Q +
+ + + + + Q
+ + + + +

+ + Q + + +
+ + + Q + +
+ + + + Q +
+ + + + + Q
+ + + + +

+ + + Q + +
+ + + + Q +
+ + + + + Q
+ + + + +

+ + + + Q +
+ + + + + Q
+ + + + +
+ + + + +

All solutions have been calculated, timer stopped.
Process completed in 0.0000 seconds (0.346700 milliseconds) with print statements. Found 4 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
```

## Detailed Mode: Optimized1

```
Timer started, processing...

+ + Q + + +
+ + + Q + +
+ + + + Q +
+ + + + + Q
+ + + + +

+ + + Q + +
+ + + + Q +
+ + + + + Q
+ + + + +

+ + + + Q +
+ + + + + Q
+ + + + +
+ + + + +

+ + + + Q +
+ + + + + Q
+ + + + +
+ + + + +

All solutions have been calculated, timer stopped.
Process completed in 0.0300 seconds (29.143100 milliseconds) with print statements. Found 4 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
```

## Detailed Mode: Optimized2

```
Timer started, processing...

-----
* Q * * * *
* * * * Q *
* * * * * Q
Q * * * * *
* * * * Q *
* * * * * Q
-----

* * Q * * *
* * * * * Q
* Q * * * *
* * * * Q *
Q * * * *
* * * * Q *
-----

* * * * Q *
Q * * * *
* * * * Q *
* Q * * * *
* * * * Q *
-----

* * * * Q *
* * Q * * *
Q * * * *
* * * * Q *
* * * * Q *
* Q * * * *
-----

All solutions have been calculated, timer stopped.

Process completed in 0.0260 seconds (25.972000 milliseconds) with print statements. Found 4 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
```

## Detailed Mode: Backtracking

```
Timer started, processing...

-----
* Q * * * *
* * * * Q *
* * * * * Q
Q * * * * *
* * * * Q *
* * * * * Q
-----

* * Q * * *
* * * * * Q
* Q * * * *
* * * * Q *
Q * * * *
* * * * Q *
-----

* * * * Q *
Q * * * *
* * * * Q *
* Q * * * *
* * * * Q *
* * Q * * *
-----

* * * * Q *
* * Q * * *
Q * * * *
* * * * Q *
* * * * Q *
* Q * * * *
-----

All solutions have been calculated, timer stopped.

Process completed in 0.1000 seconds (100.295800 milliseconds) with print statements. Found 4 solutions.
(Note: Print statements may significantly affect the execution time with higher n values.)
(Use the 'All approaches' mode to measure time without the impact of prints.)
```

## All Approaches

```
Starting Brute Force, timer started. This might take some time..
Brute Force took: 0.0660 seconds (66.163900 milliseconds). Found 4 solutions.

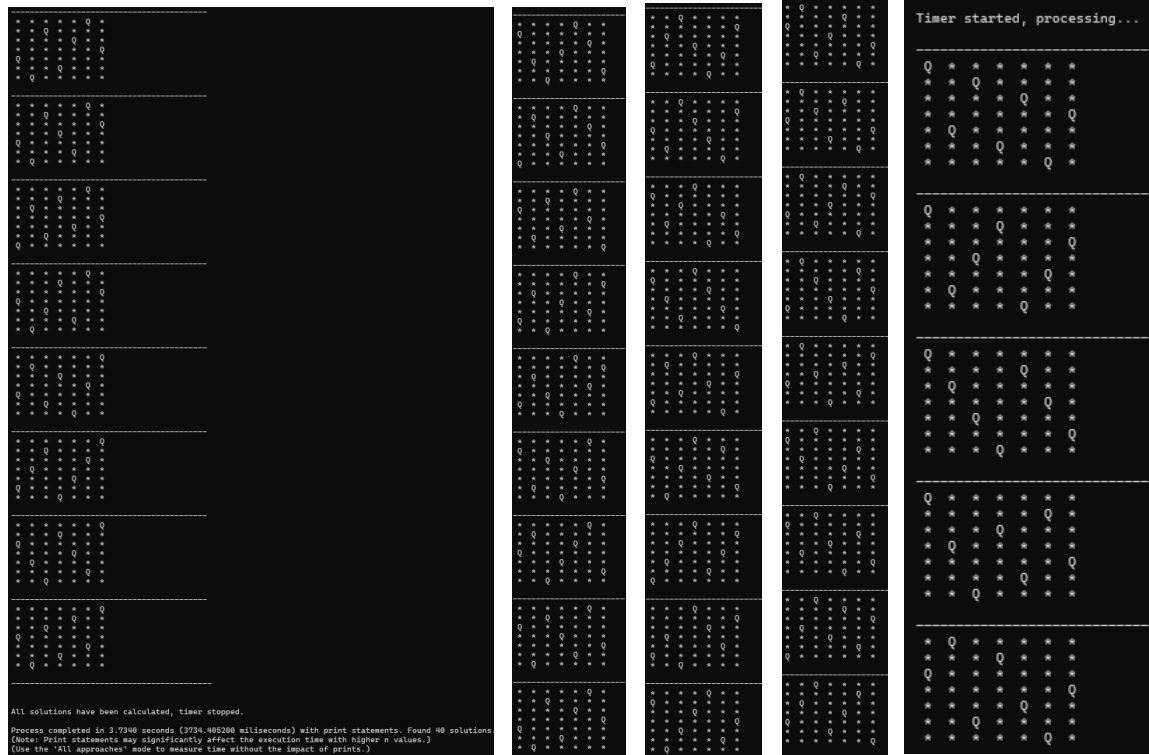
Starting Optimized1, timer started.
Optimized1 took: 0.009000 seconds (8.627400 milliseconds). Found 4 solutions.

Starting Optimize2, timer started.
Optimized2 took: 0.0010 seconds (1.333100 milliseconds). Found 4 solutions.

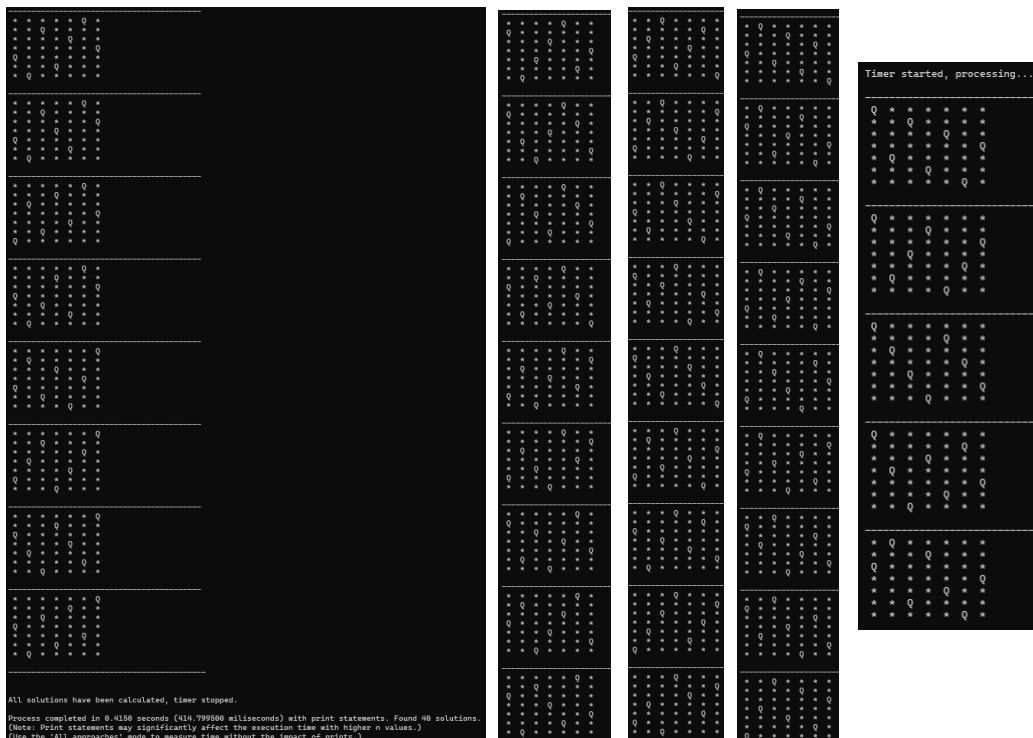
Starting Backtracking, timer started.
Backtracking took: 0.0010 seconds (0.449700 milliseconds). Found 4 solutions.
```

n=7

*Detailed Mode: Brute Force*



*Detailed Mode: Optimized1*



## Detailed Mode: Optimized2

[illegible][illegible][illegible][illegible]

```

Timer started, processing...

Q  * * * * *
  * * Q * * *
  * * * Q * *
  * * Q * * *
  * * * Q * *
  * * * * Q
  * * * * *

-----

Q  * * * * *
  * * Q * * *
  * * * Q * *
  * * * * Q
  * * Q * * *
  * * * * Q
  * * * * *

-----

Q  * * * * *
  Q * * * Q *
  * * * * Q
  * * Q * * *
  * * * Q *
  * * * * Q
  * * * * *

-----

Q  * * * * *
  * * Q * * *
  * * * Q * *
  * * * * Q
  * * Q * * *
  * * * Q *
  * * * * *

-----

*  Q * * * *
  Q * * * *
  * * * * Q
  * * * * Q
  * * Q * *
  * * * Q
  * * * *

```

### Detailed Mode: Backtracking

[illegible][illegible][illegible][illegible]

```

Timer started, processing...

Q * * * * *
* * * * Q
* * * * Q
* * * * Q
* * * * Q
* * * * Q

-----

Q * * * * *
* * * * Q
* * * * Q
* * * * Q
* * * * Q
* * * * Q

-----

Q * * * * *
* * * * Q
* * * * Q
* * * * Q
* * * * Q
* * * * Q

-----

Q * * * * *
* * * * Q
* * * * Q
* * * * Q
* * * * Q
* * * * Q

-----

Q * * * * *
* * * * Q
* * * * Q
* * * * Q
* * * * Q
* * * * Q

```

## All Approaches

```
Starting Brute Force, timer started. This might take some time..
Brute Force took: 3.0700 seconds (3069.247000 milliseconds). Found 40 solutions

Starting Optimized1, timer started.
Optimized1 took: 0.124000 seconds (124.324400 milliseconds). Found 40 solutions

Starting Optimize2, timer started.
Optimized2 took: 0.0100 seconds (10.493700 milliseconds). Found 40 solutions.

Starting Backtracking, timer started.
Backtracking took: 0.0020 seconds (2.452400 milliseconds). Found 40 solutions.
```

n = 8

### Detailed Mode: Brute Force

The visualization displays 40 solutions for the N-Queens problem with n=8. Each solution is represented by an 8x8 grid where 'Q' indicates a queen and '.' indicates an empty cell. The solutions are arranged in a grid of 10 rows and 4 columns. The first column shows the first 4 solutions, the second column shows the next 4, and so on. The text at the bottom of the first column reads: "All solutions have been calculated, timer stopped. Process completed in 321.0420 seconds (321041.763500 milliseconds) with print statements. Found 92 solutions. (Note: Print statements may significantly affect the execution time with higher n values.) (Use the 'All approaches' mode to measure time without the impact of prints.)"

### Detailed Mode: Optimized1

The visualization displays 40 solutions for the N-Queens problem with n=8, using the Optimized1 approach. Each solution is represented by an 8x8 grid where 'Q' indicates a queen and '.' indicates an empty cell. The solutions are arranged in a grid of 10 rows and 4 columns. The text at the bottom of the first column reads: "All solutions have been calculated, timer stopped. Process completed in 4.0470 seconds (4046.424000 milliseconds) with print statements. Found 92 solutions. (Note: Print statements may significantly affect the execution time with higher n values.) (Use the 'All approaches' mode to measure time without the impact of prints.)"

*Detailed Mode: Optimized2*

[illegible][illegible]

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099
1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	

[illegible]

### Detailed Mode: Backtracking

[illegible]

Case	Case	Case	Case	Case	Case	Case	Case
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88
89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104
105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136
137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152
153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168
169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184
185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216
217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232
233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248
249	250	251	252	253	254	255	256
257	258	259	260	261	262	263	264
265	266	267	268	269	270	271	272
273	274	275	276	277	278	279	280
281	282	283	284	285	286	287	288
289	290	291	292	293	294	295	296
297	298	299	300	301	302	303	304
305	306	307	308	309	310	311	312
313	314	315	316	317	318	319	320
321	322	323	324	325	326	327	328
329	330	331	332	333	334	335	336
337	338	339	340	341	342	343	344
345	346	347	348	349	350	351	352
353	354	355	356	357	358	359	360
361	362	363	364	365	366	367	368
369	370	371	372	373	374	375	376
377	378	379	380	381	382	383	384
385	386	387	388	389	390	391	392
393	394	395	396	397	398	399	400
401	402	403	404	405	406	407	408
409	410	411	412	413	414	415	416
417	418						

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099
1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	

[illegible]

### All Approaches

```
Starting Brute Force, timer started. This might take some time..
Brute Force took: 315.9350 seconds (315934.588000 milliseconds). Found 92 solutions.

Starting Optimizer1, timer started.
Optimizer1 took: 5.753000 seconds (5752.530400 milliseconds). Found 92 solutions.

Starting Optimizer2, timer started.
Optimizer2 took: 0.2430 seconds (242.460200 milliseconds). Found 92 solutions.

Starting Backtracking, timer started.
Backtracking took: 0.0370 seconds (36.304400 milliseconds). Found 92 solutions.
```

n = 9

*Detailed Mode: Optimized!*

[illegible]

*Detailed Mode: Optimized2*

[illegible][illegible][illegible]

### Detailed Mode: Backtracking

[illegible]

### All Approaches

```
Starting Optimized1, timer started.
Optimized1 took: 102.255000 seconds (102256.055500 milliseconds). Found 352 solutions.

Starting Optimize2, timer started.
Optimized2 took: 3.1050 seconds (3104.952900 milliseconds). Found 352 solutions.

Starting Backtracking, timer started.
Backtracking took: 0.2640 seconds (263.213300 milliseconds). Found 352 solutions.
```

n = 10

*Detailed Mode: Optimized2*

[illegible]



### Detailed Mode: Backtracking

[illegible]*AllApproaches*

```
Starting Optimize2, timer started.  
Optimized2 took: 41.676000 seconds. Found 724 solutions.
```

```
Starting Backtracking, timer started.  
Backtracking took: 2.411000 seconds. Found 724 solutions.
```

```
Process exited after 45.77 seconds with return value 0
Press any key to continue . . .
```