BLM 2425 ALGORİTMA ANALİZİ

ÖDEV 4

Backtracking

Öğrenci Adı: Sinem SARAK
Öğrenci Numarası: 22011647
Dersin Eğitmeni: M. Elif KARSLIGİL
Açıklama Video Linki: https://youtu.be/Taw_0txyPj8

İçindekiler

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 çağrılarak 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 artılır. 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×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 temsilini 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

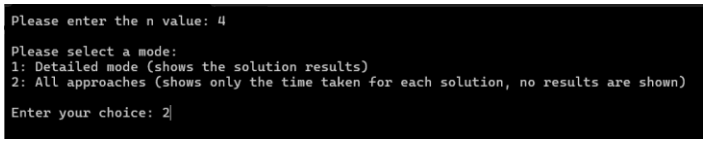
İlk olarak tehdit alanlarının belirlenmesi sırasında, her bir hücrenin tehdit altında olup olmadığına göre 1 ya da 0 değerleri kullanıldı. Ancak bu yöntem, bir hücrenin birden fazla vezir tarafından tehdit edilmesi durumunda hatalara neden oldu. Örneğin, bir vezirin kaldırılması sırasında, tehdit alanlarının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

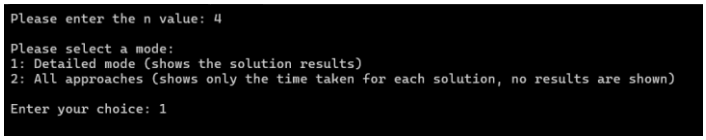
Bu sorunun çözümü için tehdit edilen hücrelere 1 ve 0 değerleri koymak yerine, her bir hücredeki tehdit sayısını takip edecek şekilde artırma ve azaltma yöntemi benimsendi. Böylece, bir hücre birden fazla vezir tarafından tehdit edilse bile, tehdit sayısı doğru şekilde hesaplanıp yönetildi. Bu yaklaşım,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

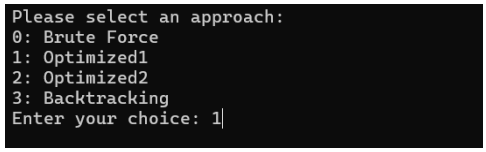
Ekran Çıktıları

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 bulduğu 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

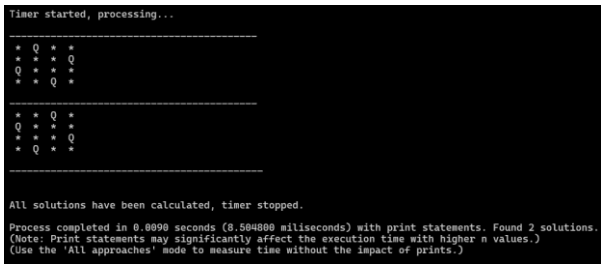




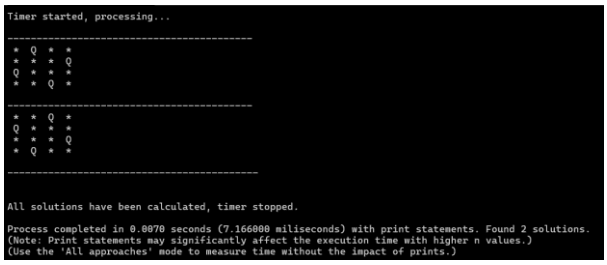


n = 4

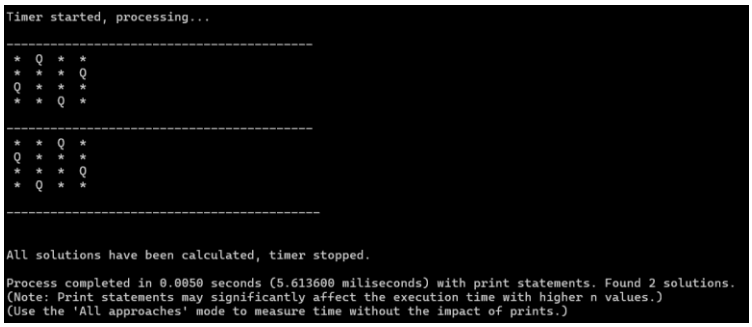
Detailed Mode: Brute Force



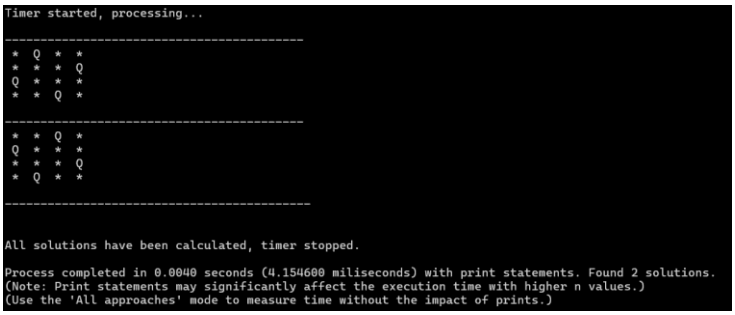
Detailed Mode:Optimized1



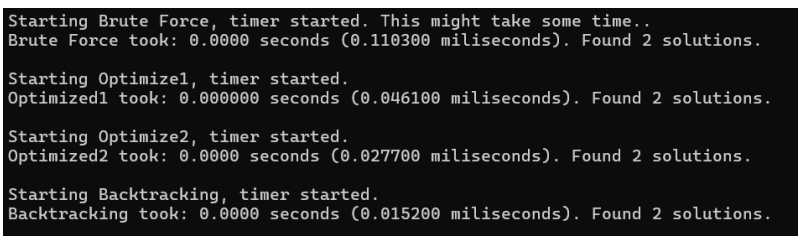
Detailed Mode: Optimized2



Detailed Mode: Backtracking

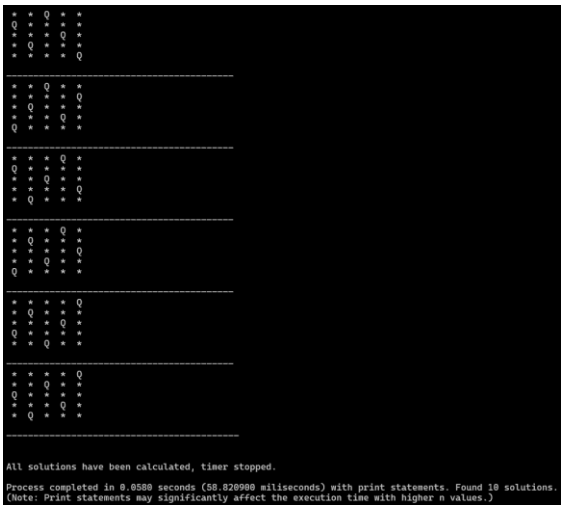


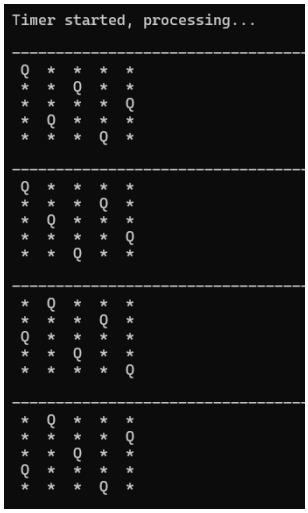
All Approaches



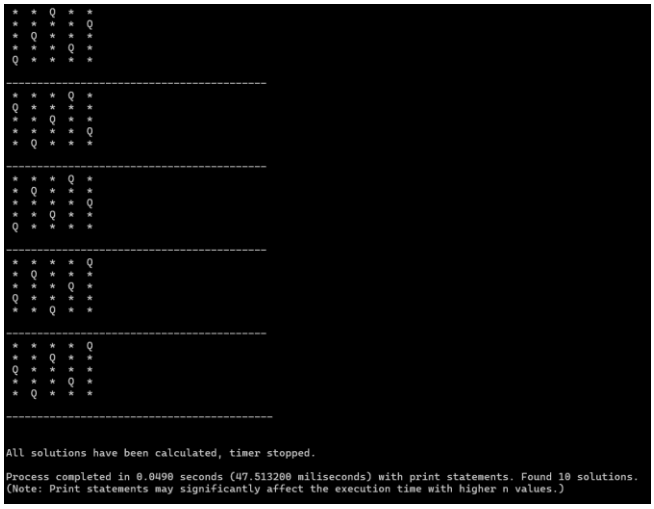
n = 5

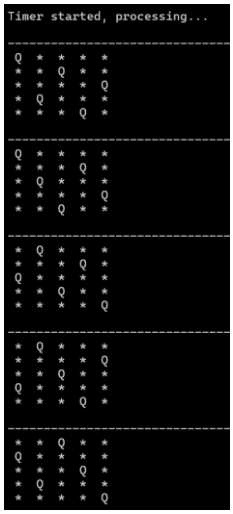
Detailed Mode: Brute Force



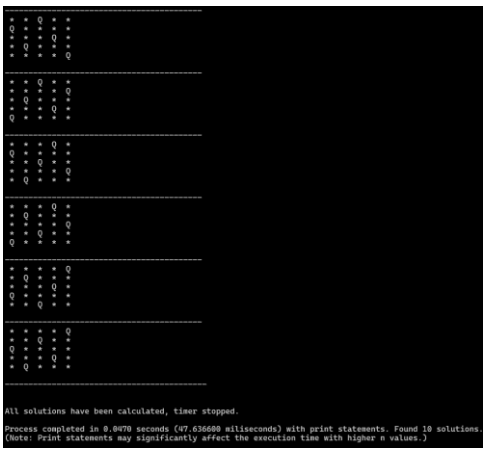


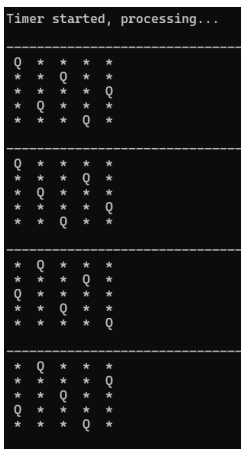
Detailed Mode: Optimized1



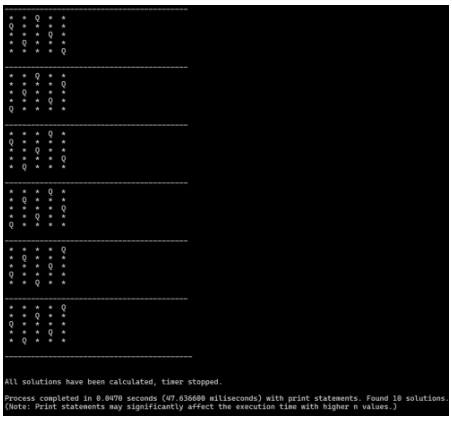


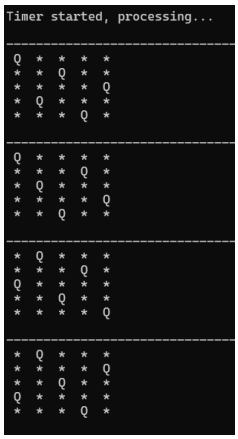
Detailed Mode: Optimized2



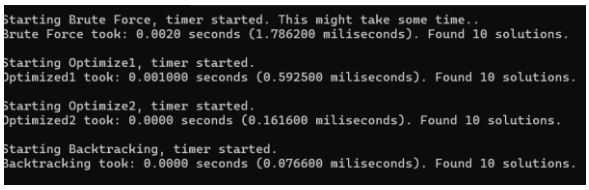


Detailed Mode: Backtracking



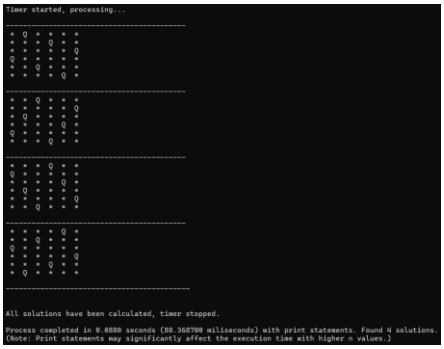


All Approaches

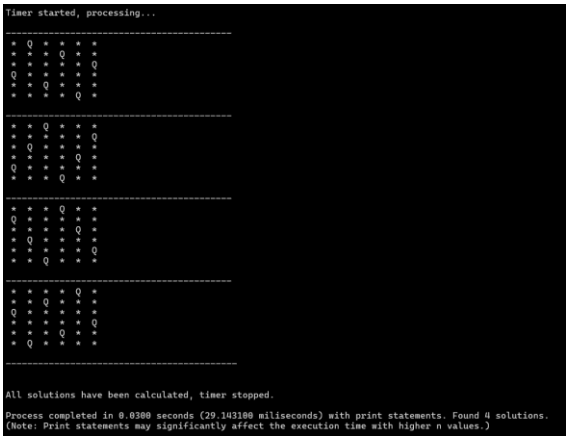


<u>n = 6</u>

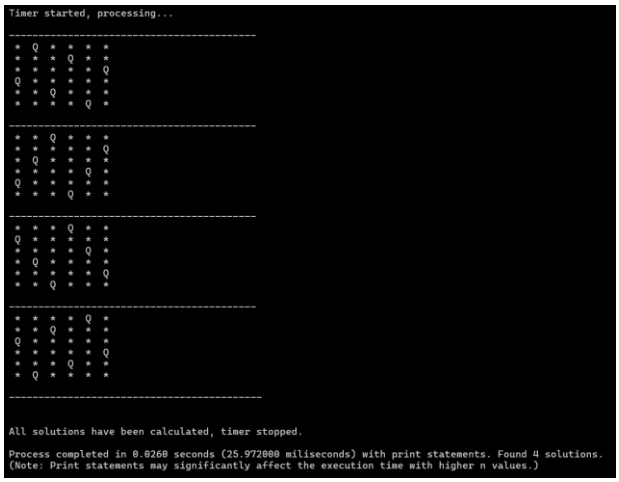
Detailed Mode: Brute Force



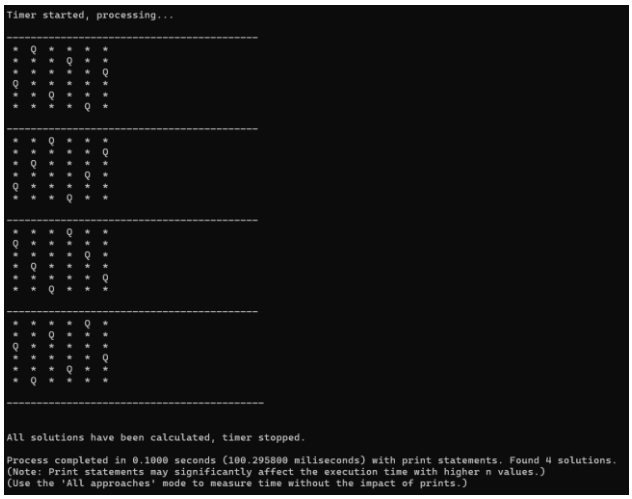
Detailed Mode: Optimized1



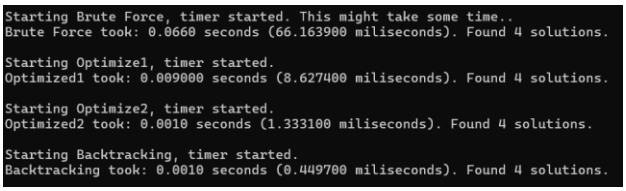
Detailed Mode: Optimized2



Detailed Mode: Backtracking

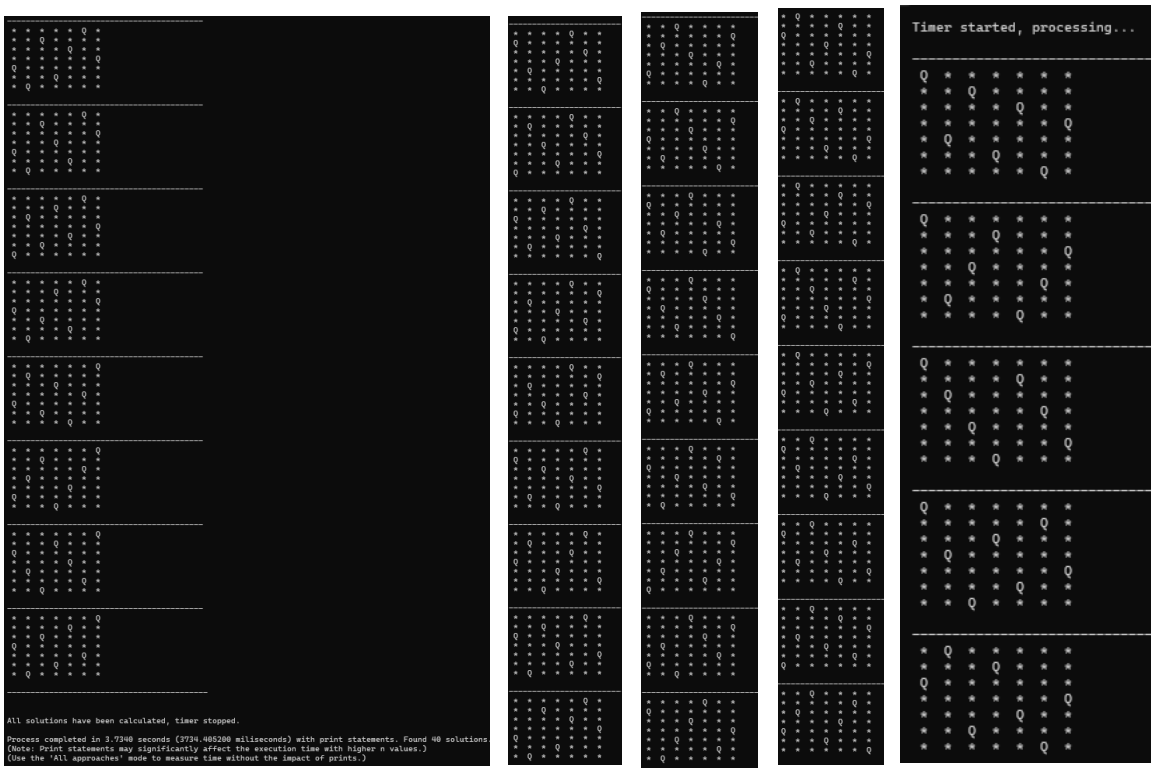


All Approaches

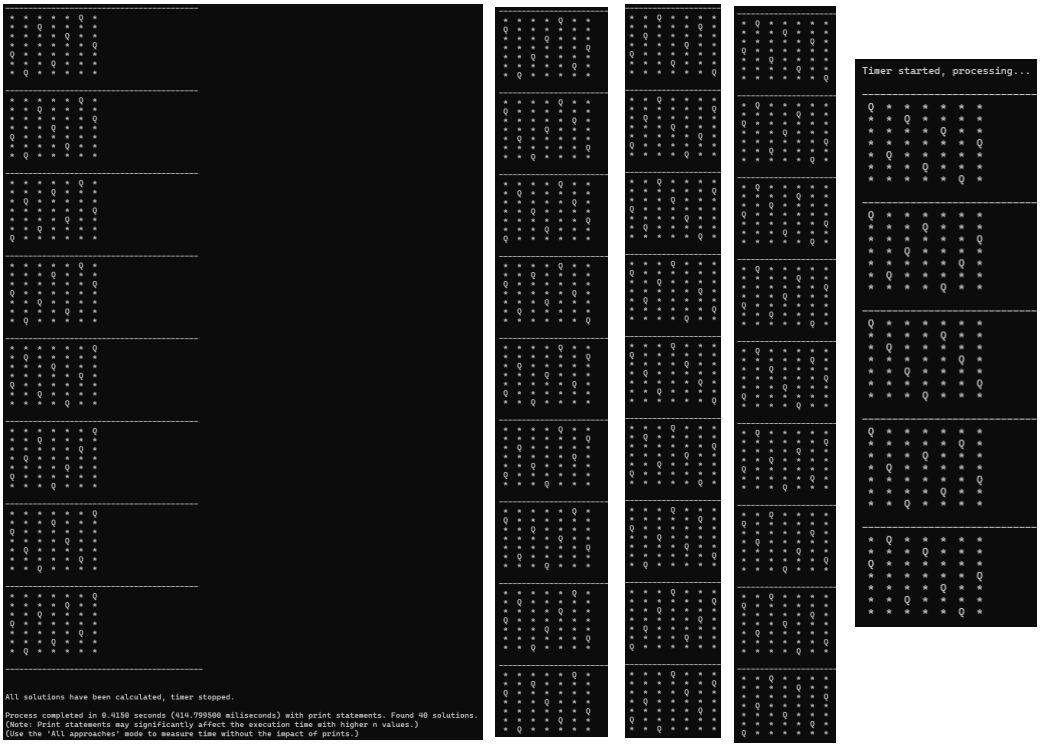


n = 7

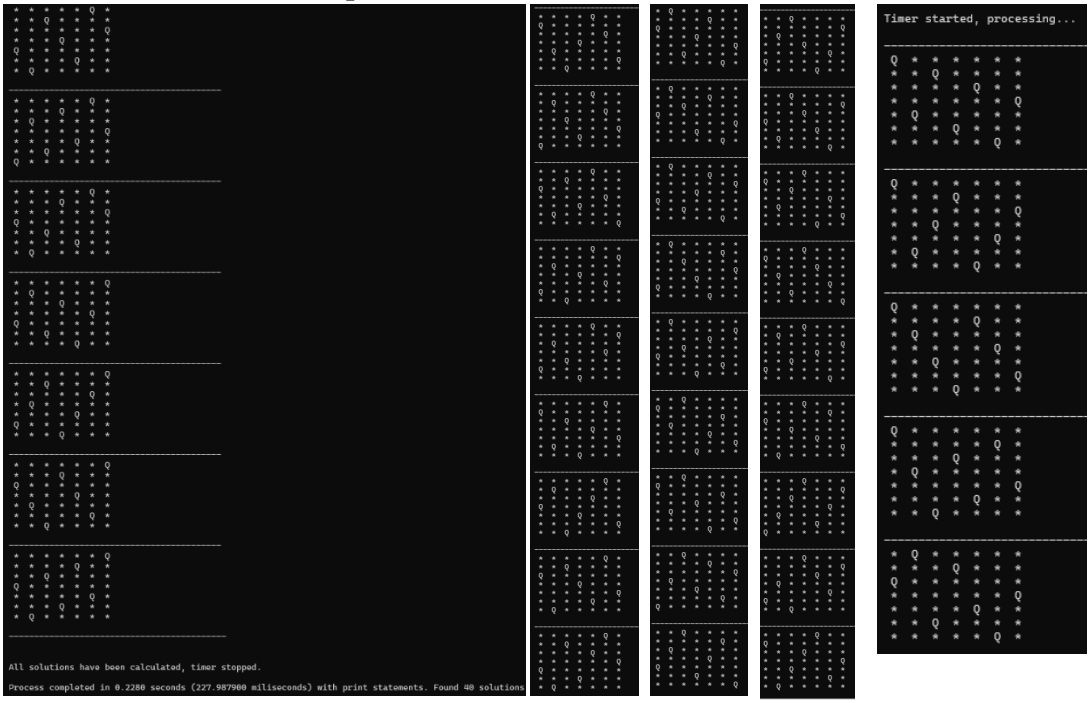
Detailed Mode: Brute Force



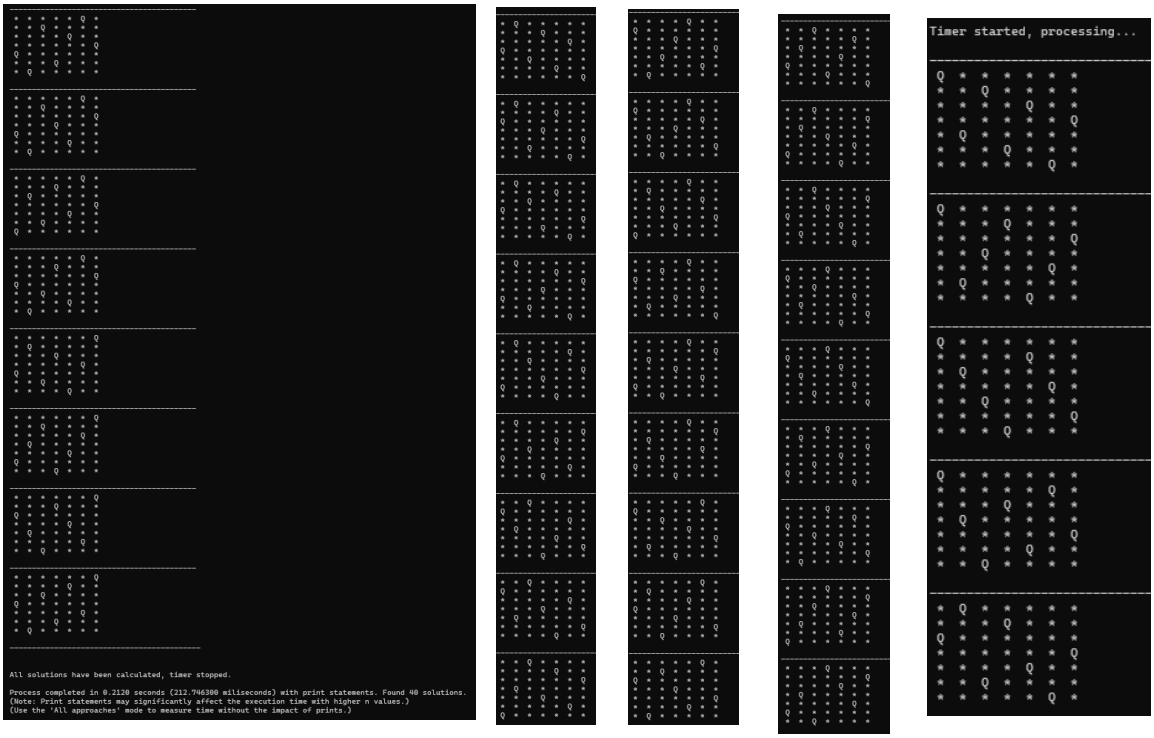
Detailed Mode: Optimized1



Detailed Mode: Optimized2



Detailed Mode: Backtracking

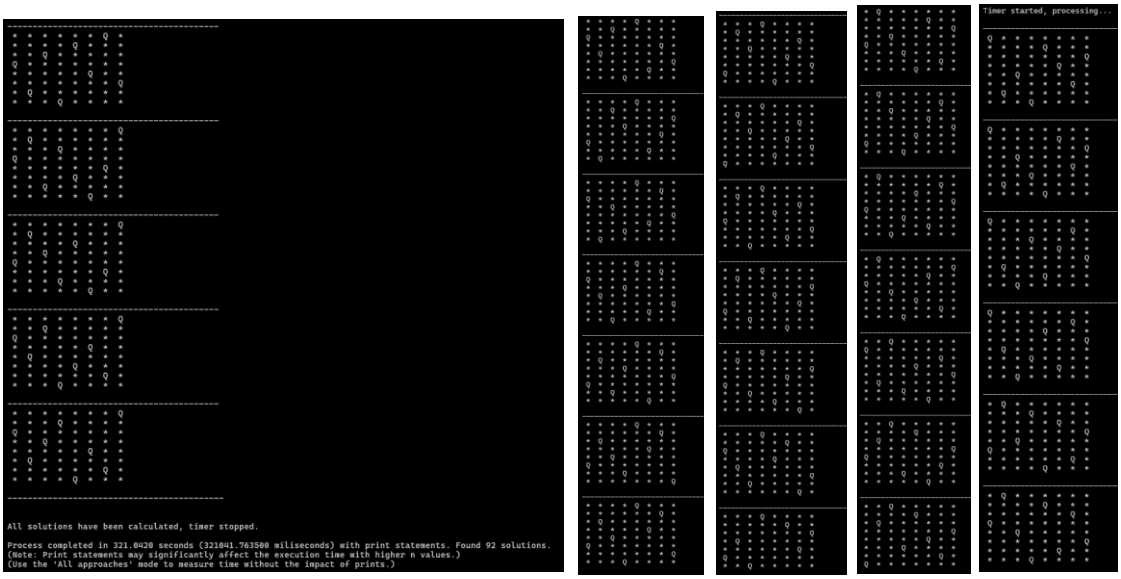


All Approaches

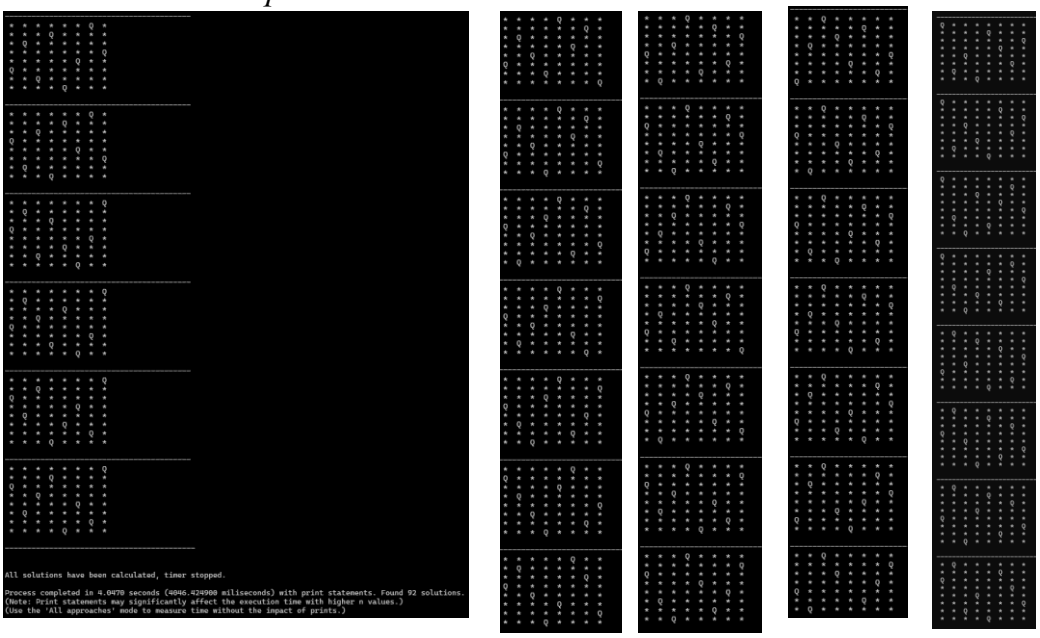
```
Starting Brute Force, timer started. This might take some time..
Brute Force took: 3.0700 seconds (3069.247000 miliseconds). Found 40 solutions

Starting Optimize1, timer started.
Optimized1 took: 0.124000 seconds (124.324400 miliseconds). Found 40 solutions

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

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

n = 8

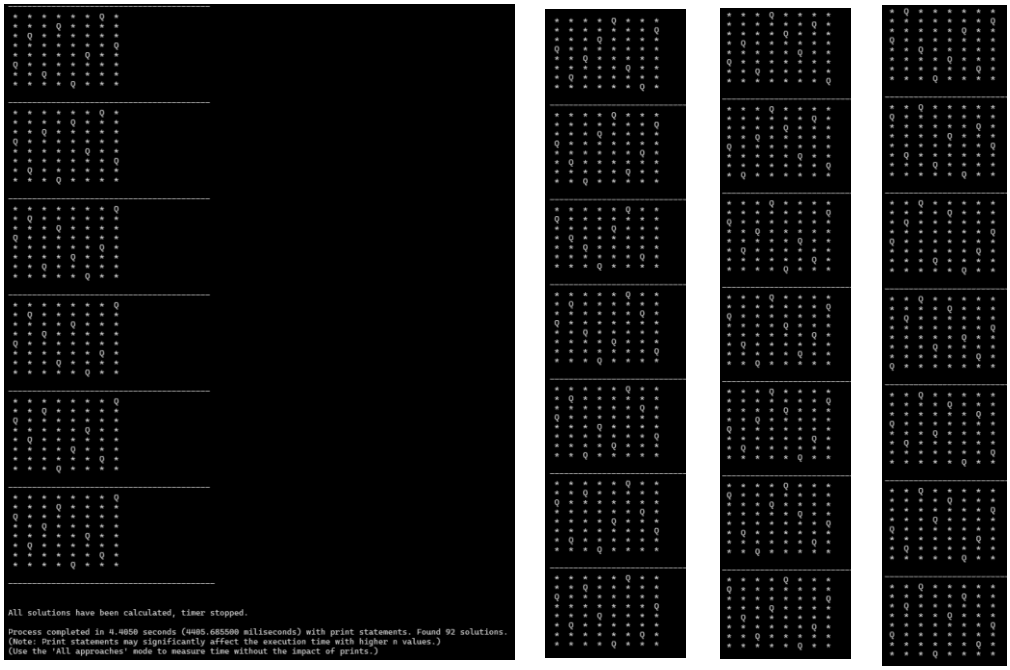
Detailed Mode: Brute Force



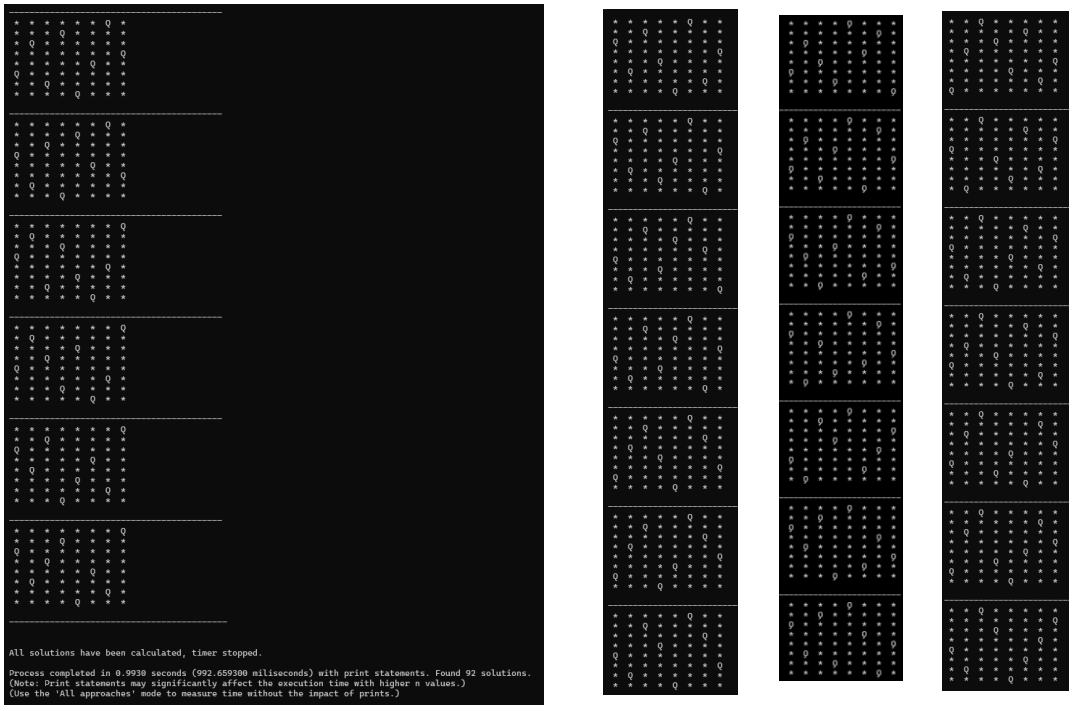
Detailed Mode: Optimized1



Detailed Mode: Optimized2



Detailed Mode: Backtracking

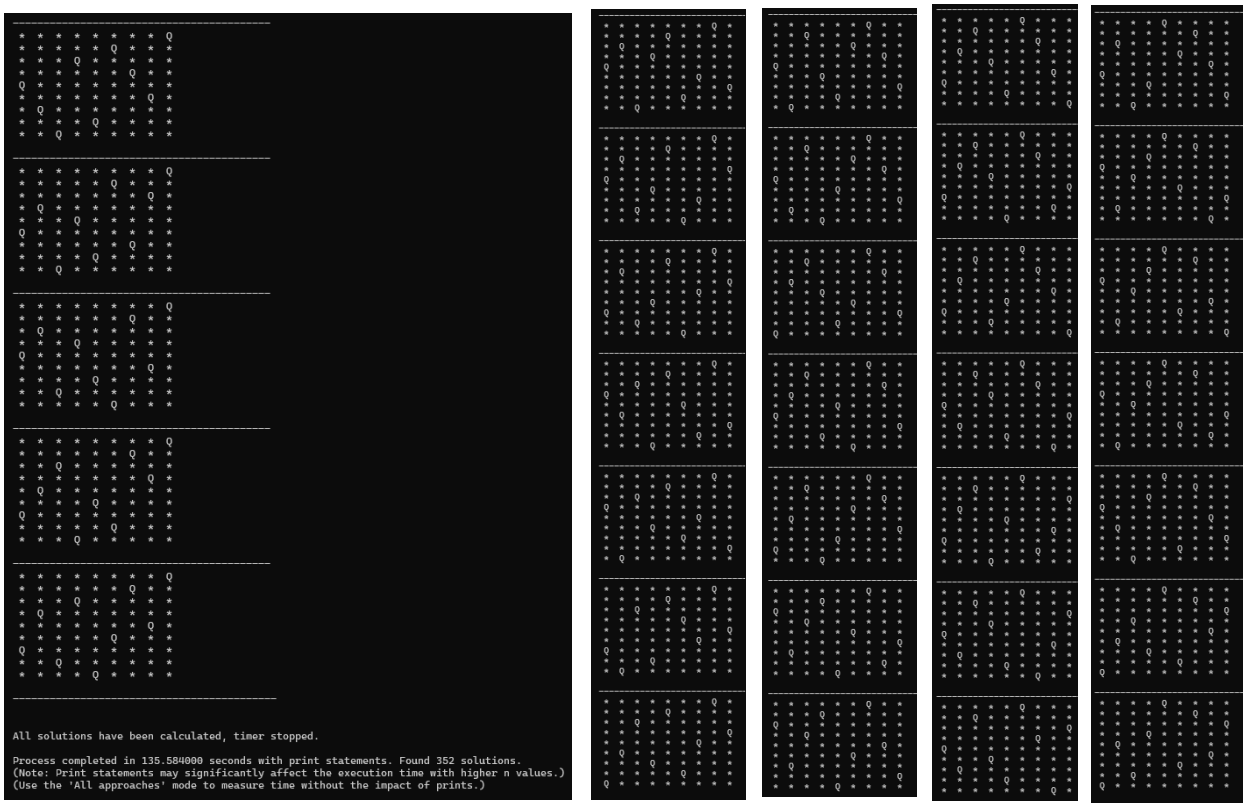


All Approaches

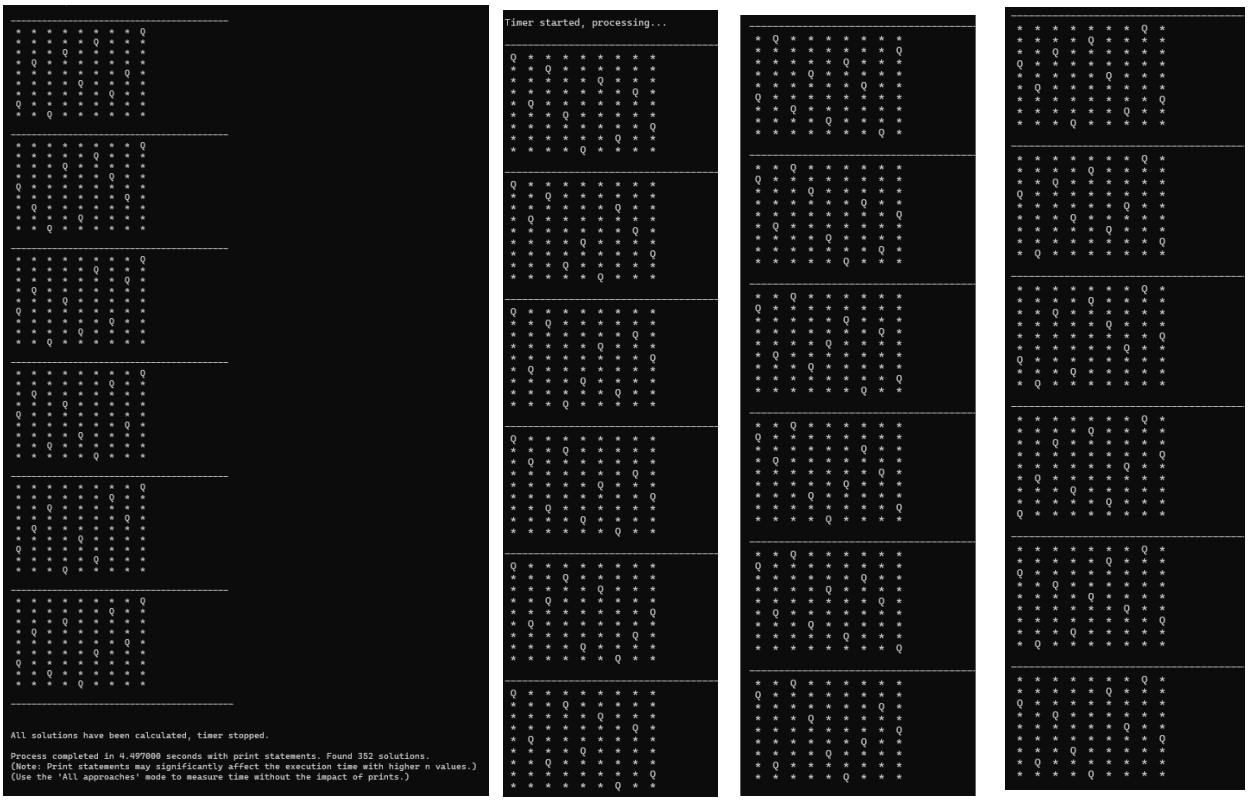
```
Starting Brute Force, timer started. This might take some time..
Brute Force took: 315.9350 seconds (315934.588000 miliseconds). Found 92 solutions.

Starting Optimize1, timer started.
Optimized1 took: 5.753000 seconds (5752.530400 miliseconds). Found 92 solutions.

Starting Optimize2, timer started.
Optimized2 took: 0.2430 seconds (242.460200 miliseconds). Found 92 solutions.

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

n = 9

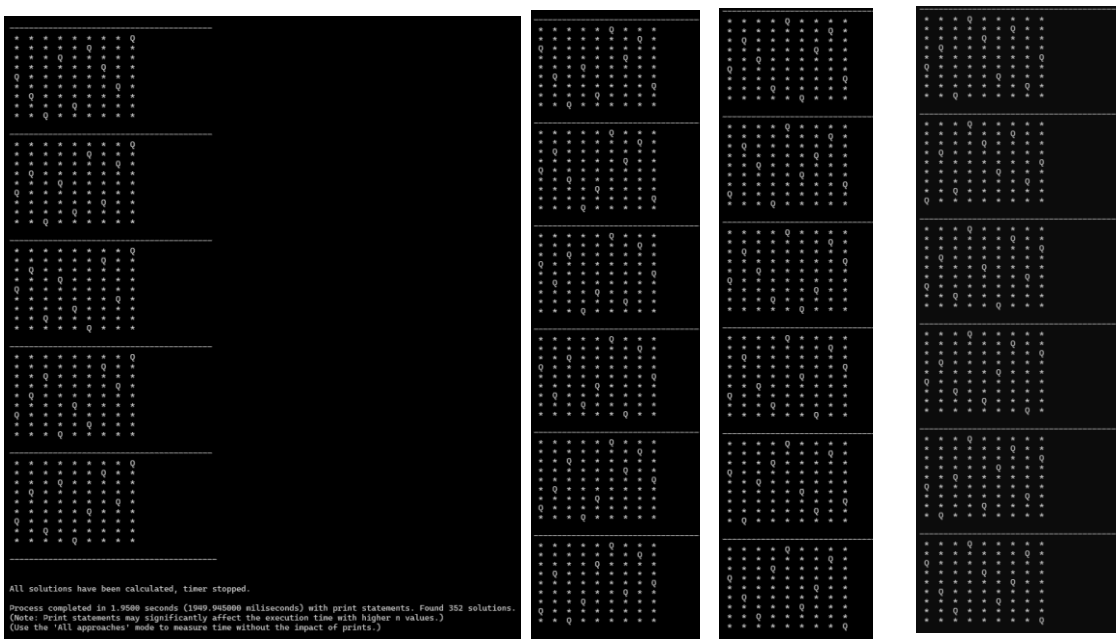
Detailed Mode: Optimized1



Detailed Mode: Optimized2



Detailed Mode: Backtracking

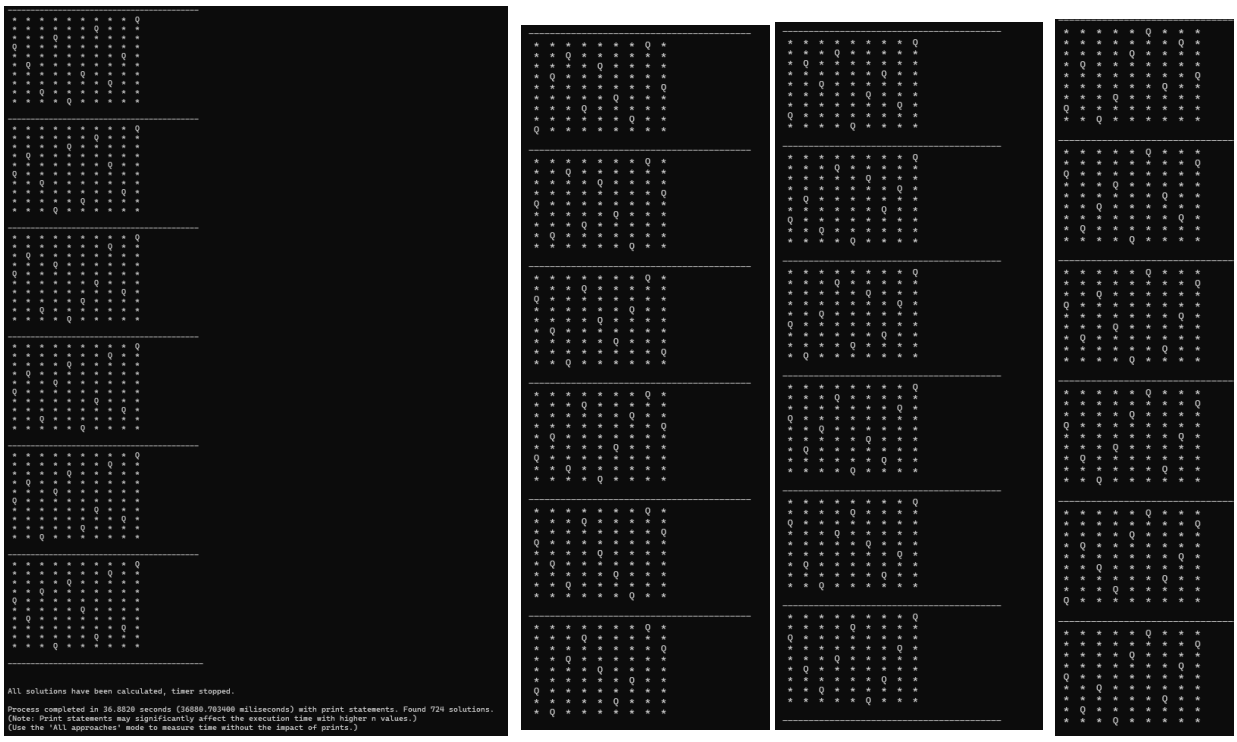


All Approaches

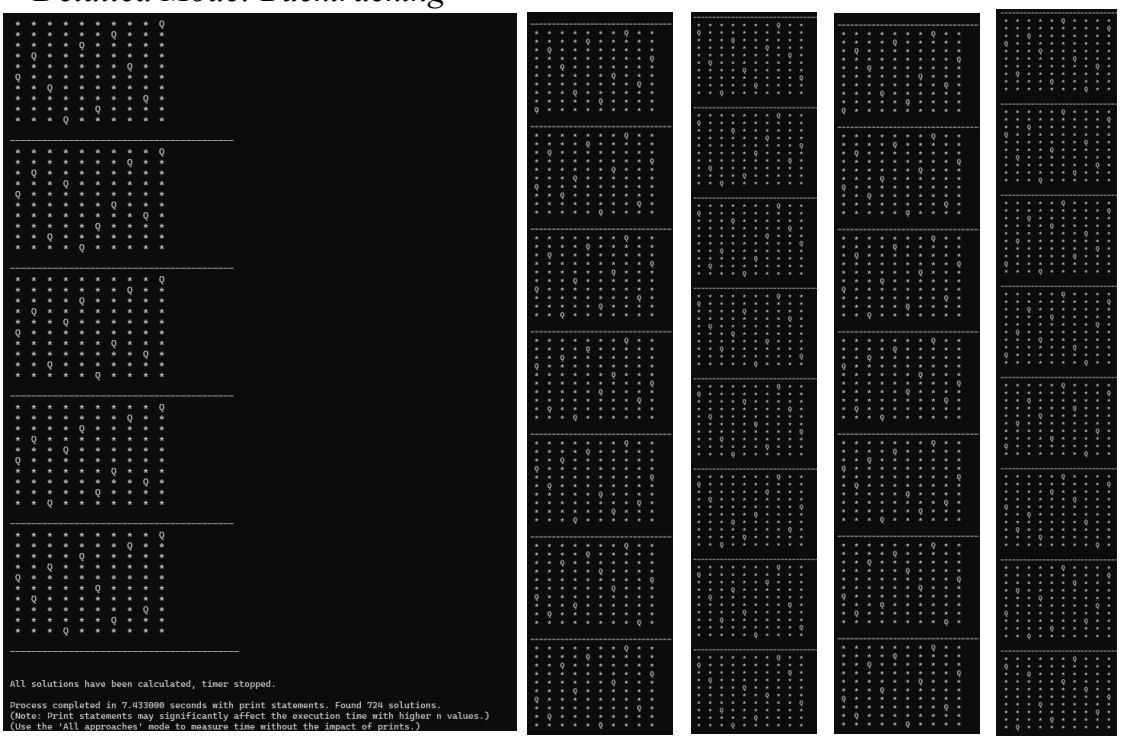
```
Starting Optimize1, timer started.
Optimized1 took: 102.255000 seconds (102256.055500 miliseconds). Found 352 solutions.

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

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

n = 10

Detailed Mode: Optimized2



Detailed Mode: Backtracking



AllApproaches