Bir algoritmanın karmaşıklığı ne anlama gelir?



www.yzokul.com.tr

Bir algoritmanın karmaşıklığı ne anlama gelir?



Bir algoritmanın karmaşıklığı ne anlama gelir? Karmaşıklık, veri büyüdükçe bir algoritmanın ihtiyaç duyduğu zamanın veya belleğin nasıl arttığını tanımlar. Belirli bir makinedeki kesin saniye sayısıyla ilgili değildir. Bunun yerine büyüme örüntüsüyle ilgilidir: algoritmaya daha fazla veri verdiğinde işin nasıl arttığıyla. Bu kitap bu fikri adım adım açıklar.



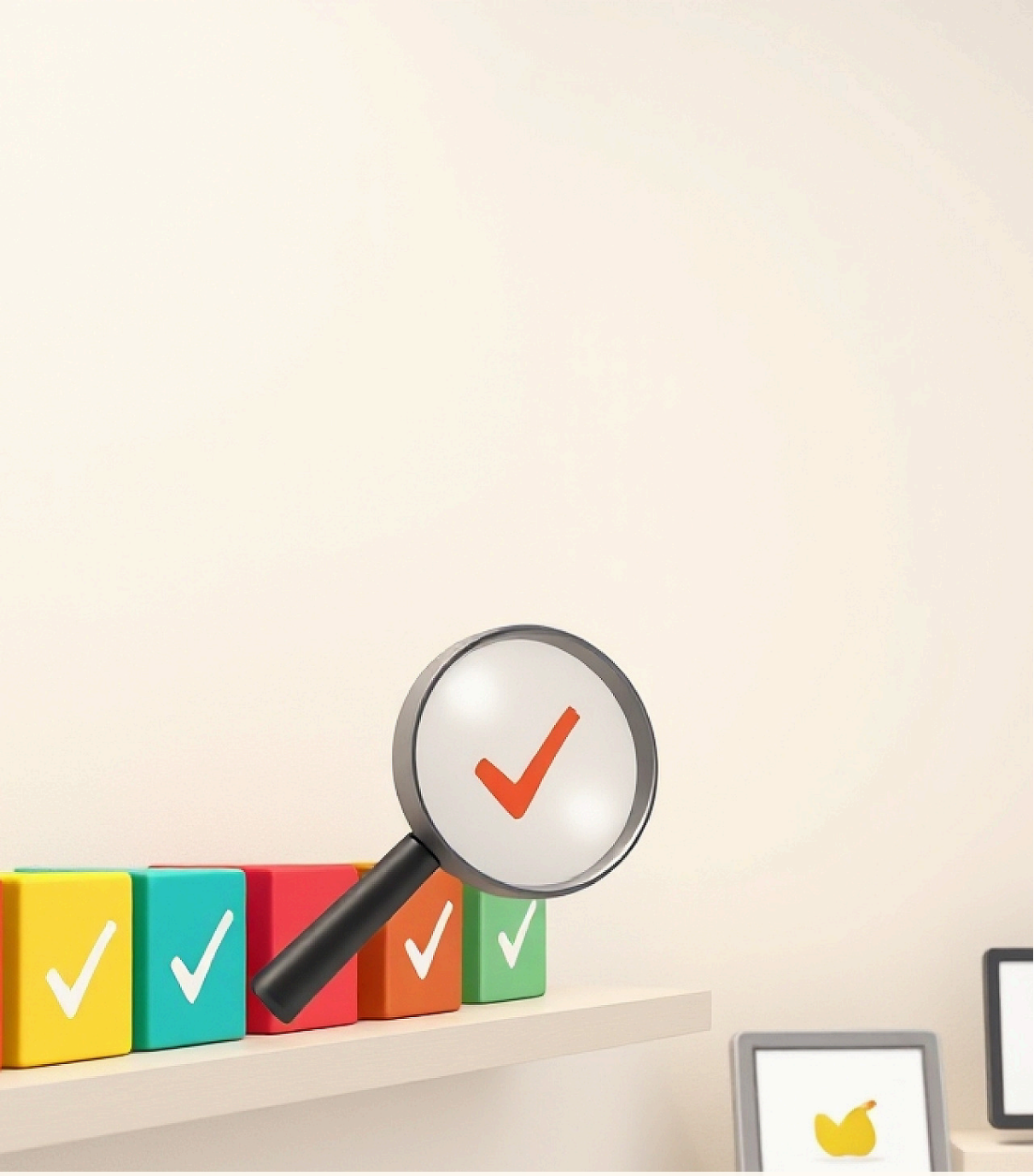
Bir algoritma, bir tarif gibi, bir görevi yapmak için adım adım bir yöntemdir. Ona veri verirsin ve bir sonuç üretir. Örneğin, bir sıralama algoritması bir sayı listesi alır ve listeyi sıralı olarak döndürür. Algoritma, talimatlar kümesidir, onu çalıştıran bilgisayar değil. Karmaşıklık, veri büyüdükçe bu talimatların nasıl davrandığını tanımlar.



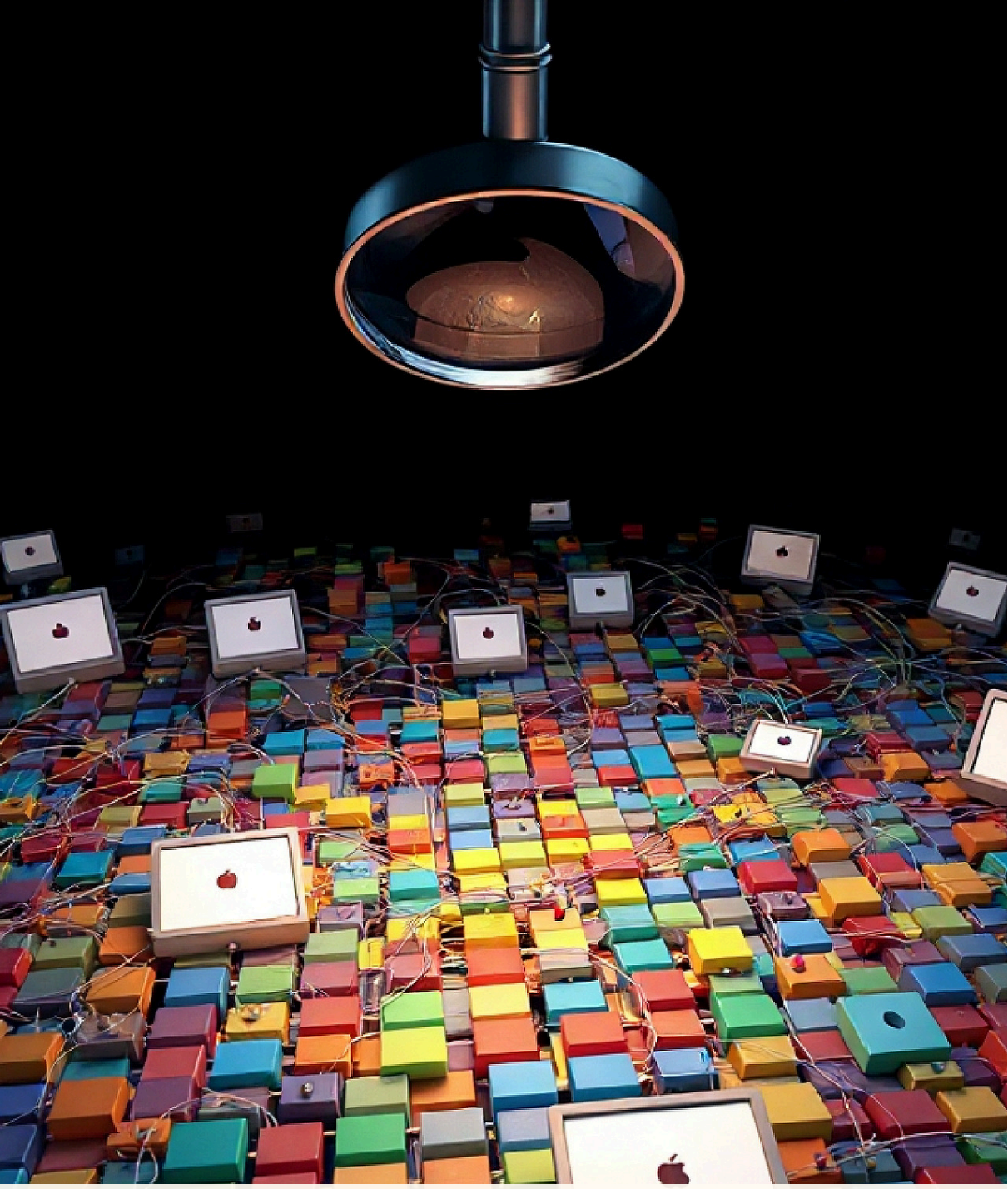
Aynı algoritma, ona ne kadar veri verdiğine bağlı olarak farklı miktarda iş gerektirir. 10 öğelik bir liste, 10.000 öğelik bir listeden farklı ele alınır. Daha fazla veriyle algoritma daha fazla adım veya daha fazla bellek gerektirebilir. Karmaşıklık bu değişimi yakalar: veri boyutu arttığında işin nasıl ölçeklendiğini sana söyler. Veri boyutu, karmaşıklığı anlamak için önemlidir.



Karmaşıklığı incelemek için, bir algoritmayı saatle zamanlamak yerine gerçekleştirdiği temel adımları say. Adım sayıları daha güvenilirdir çünkü bilgisayarın hızına bağlı değildir. Örneğin, algoritmanın iki sayıyı kaç kez karşılaştırdığını veya bir öğeyi kaç kez taşıdığını say. Bu sayı, donanımdan bağımsız olarak yapılan işin net bir ölçüsünü verir.



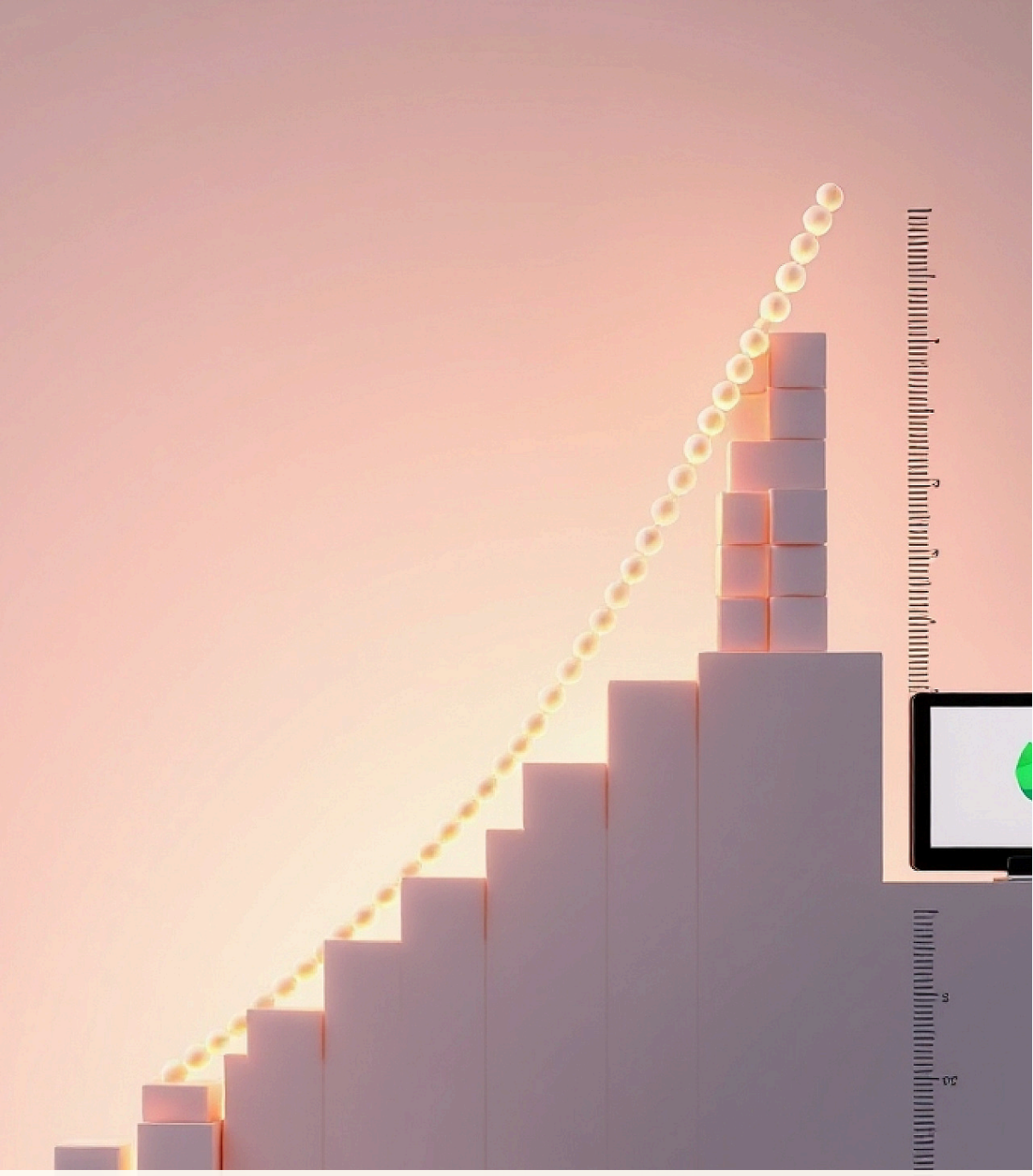
Bir listedeki her öğeye tam olarak bir kez bakan bir yöntem düşün. Örneğin, her öğeyi bir kez kontrol ederek en büyük sayıyı bulmak. Listenin n öğesi varsa, yöntem yaklaşık n adım alır. Liste iki katına çıkarsa, iş iki katına çıkar. Yani adımlar veriyle birlikte artar. Bu basit bir büyüme örüntüsüdür: veri boyutuyla orantılı iş.



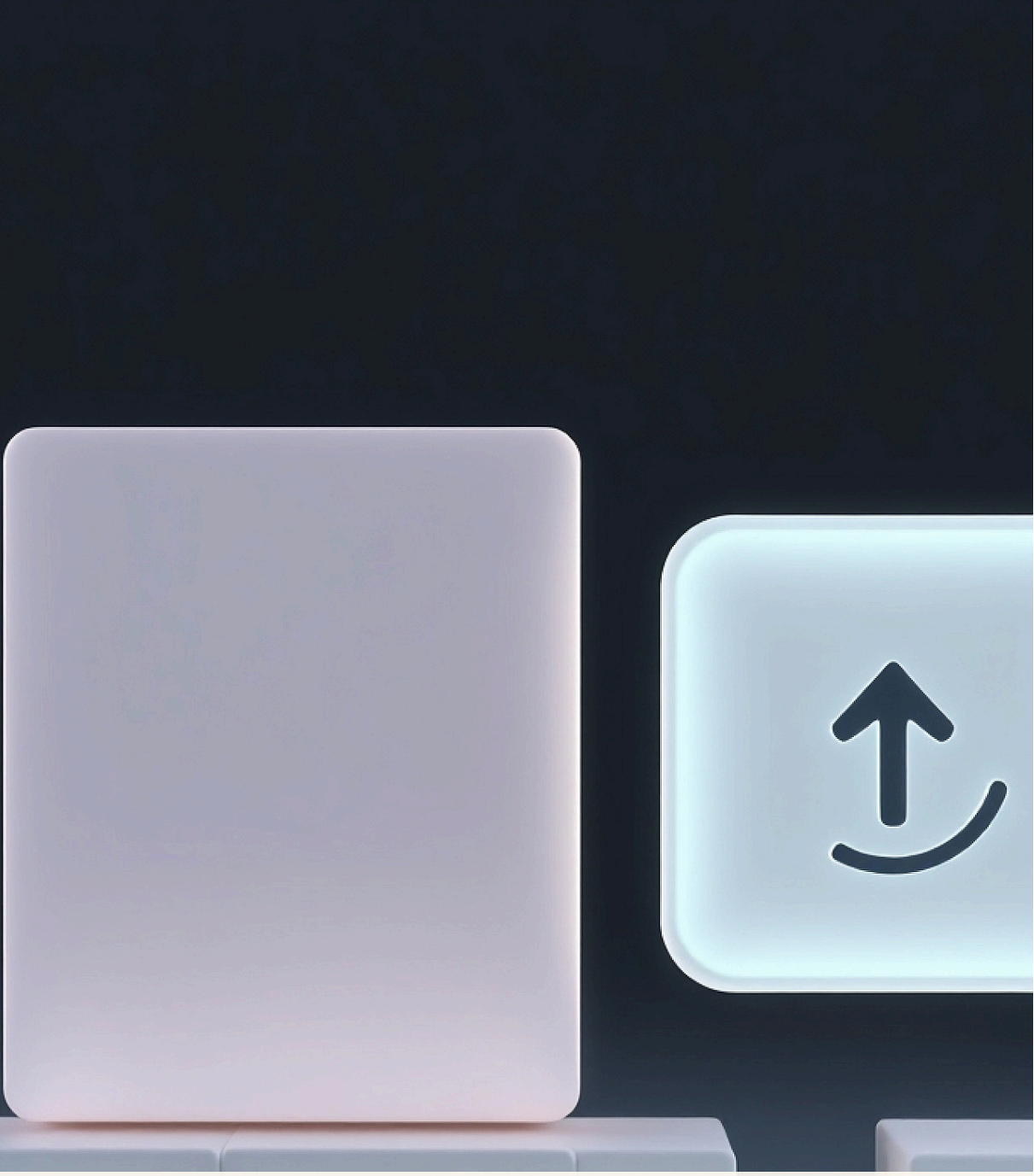
Şimdi her öğeyi diğer her öğeyle karşılaştıran bir yöntem düşün. Örneğin, kopyaları bulmak için tüm çiftleri kontrol etmek. Listenin n öğesi varsa, yaklaşık n çarpı n çift vardır, yani yöntem yaklaşık n kare adım alır. Liste iki katına çıkarsa, iş yaklaşık dört katına çıkar. Bu, veriden çok daha hızlı büyür.



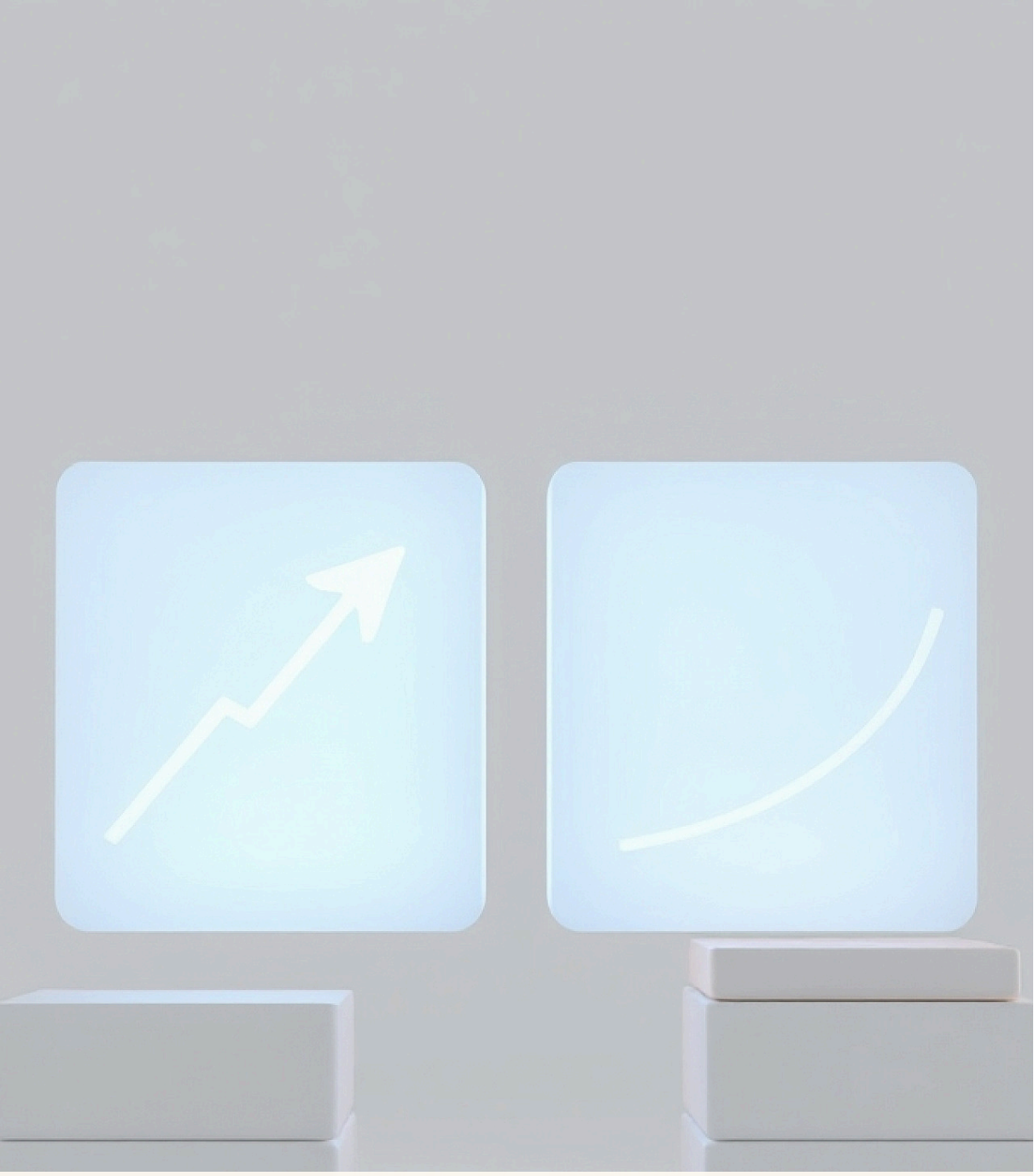
Aynı büyüyen listeler üzerinde iki yöntemi karşılaştır. n = 10 için, her öğeyi bir kez kontrol etmek yaklaşık 10 adım alırken, tüm çiftleri karşılaştırmak yaklaşık 100 adım alır. n = 100 için, birincisi yaklaşık 100 adım, ikincisi yaklaşık 10.000 adım alır. Veri büyüdükçe adım sayıları giderek daha fazla ayrışır. Farklı yöntemler farklı şekilde büyür.



Karmaşıklık, kesin adım sayısıyla değil, büyüme örüntüsüyle ilgilidir. İşin nasıl ölçeklendiğiyle ilgileniriz, 100 mü yoksa 105 adım mı olduğuyla değil. Adım sayılarındaki küçük farklar, veri büyük olduğunda pek önemli değildir. Önemli olan, işin doğrusal olarak mı, karesel olarak mı yoksa başka bir şekilde mi büyüdüğüdür. Eğilime odaklan.



Big O gösterimi, bir büyüme örüntüsünü yazmanın kısa ve genel bir yoludur. Ana eğilimi yakalar ve küçük ayrıntıları yok sayar. Örneğin, O(n) işin kabaca veri boyutuyla orantılı olarak büyüdüğü anlamına gelir. O(n kare) işin kabaca veri boyutunun karesiyle büyüdüğü anlamına gelir. Big O, algoritmaları hızlıca karşılaştırmana yardımcı olur.



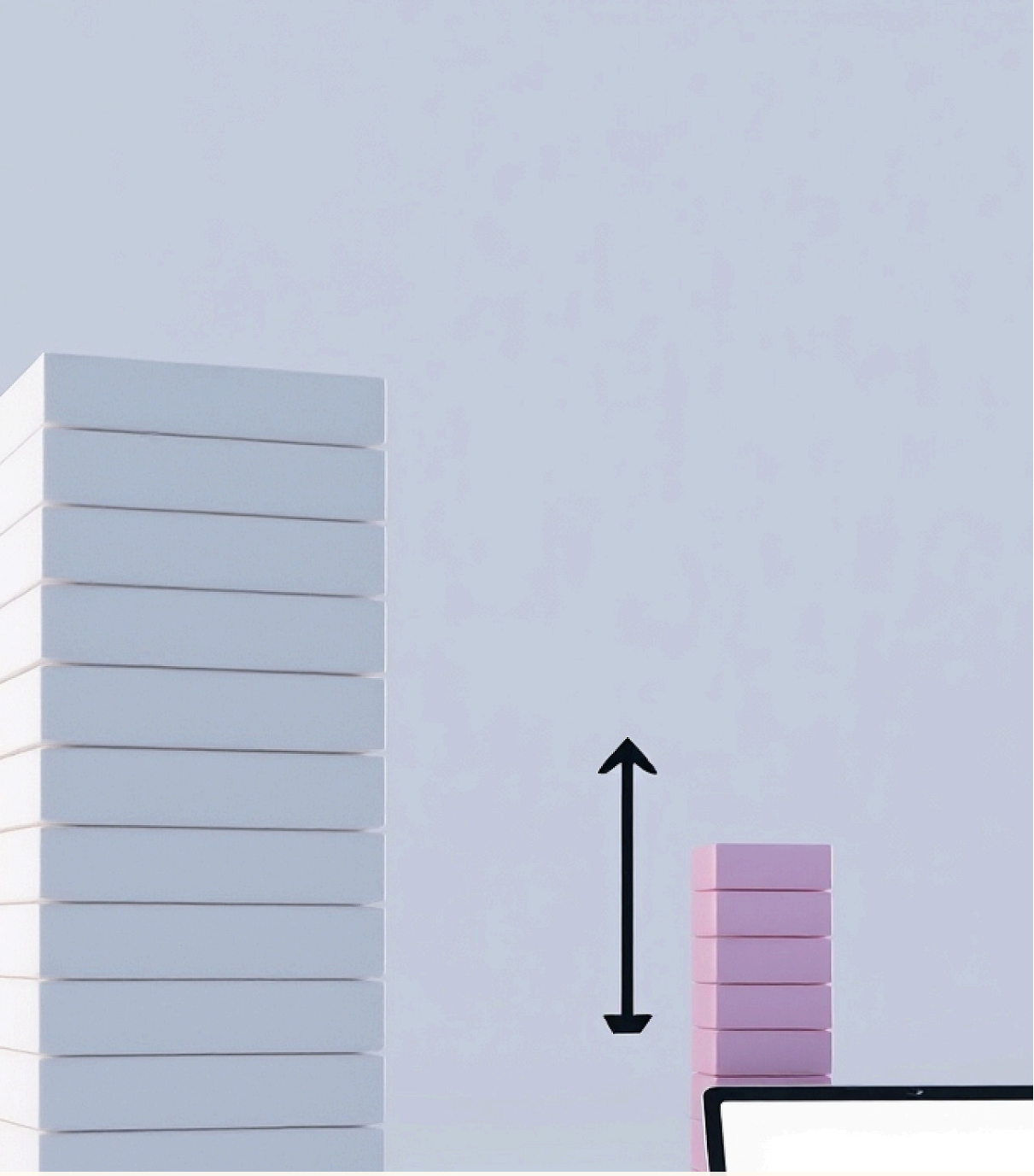
Önceki iki yöntemin Big O ile yazılışını gör. Her öğeyi bir kez kontrol etmek O(n)'dir: iş n ile doğrusal olarak büyür. Tüm çiftleri karşılaştırmak O(n kare)'dir: iş n çarpı n ile büyür. Her simgeyi büyüme örüntüsüyle eşleştir: doğrusal büyüme için O(n), karesel büyüme için O(n kare). Bu etiketler işin nasıl ölçeklendiğini özetler.



Big O saniye değildir. O(n), n saniye veya herhangi bir sabit süre anlamına gelmez. Bu, büyümenin genel bir tanımıdır, kronometre ölçümü değil. İkisi de O(n) olan iki algoritma belirli bir makinede farklı hızlarda çalışabilir, ancak büyüme davranışları benzerdir. Big O, şu anda ne kadar hızlı çalıştıklarını değil, nasıl ölçeklendiklerini karşılaştırır.



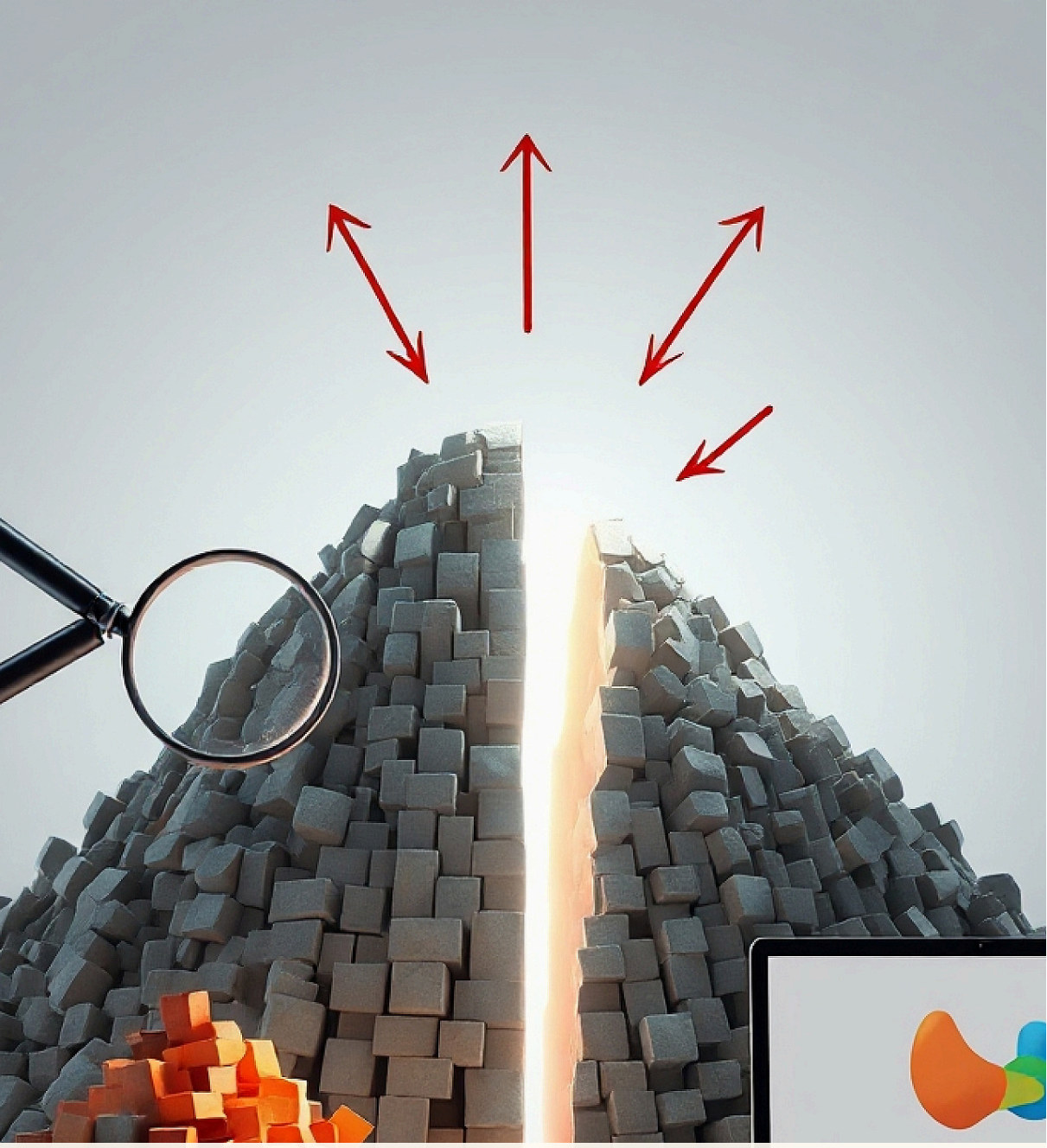
Big O, sabitleri ve küçük terimleri atar çünkü veri büyüdükçe ana eğilim baskın olur. Örneğin, 3n + 5 adım alan bir algoritma hâlâ O(n)'dir. Sabit 3 ve eklenen 5, n çok büyük olduğunda pek önemli değildir. Bunları yok saymak karşılaştırmaları basit ve adil tutar. Baskın büyüme terimine odaklan.



Karmaşıklık yalnızca zamanı değil, belleği de kapsar. Her öğenin bir kopyasını saklayan bir algoritma, veri büyüdükçe daha fazla bellek gerektirir. Örneğin, aynı boyutta yeni bir liste oluşturmak O(n) bellek alır. Tüm çiftlerden oluşan bir tablo oluştururursa, O(n kare) bellek alır. Bellek büyümesi karmaşıklığın bir parçasıdır.



Yaygın bir hata, karmaşıklığı bir makinenin algoritmayı ne kadar hızlı çalıştırdığına göre yargılamaktır. Hız donanıma bağlıdır, ancak büyüme davranışı aynı kalır. Yavaş bir bilgisayar küçük veri için daha uzun sürebilir, ancak daha büyük veriyle büyüme örüntüsü karmaşıklığı tanımlayan şeydir. Ham saniyeleri değil, adım sayılarını veya büyüme örüntülerini karşılaştır.



Bir başka yaygın hata, yalnızca çok küçük girdileri test etmektir. Küçük veride, hem doğrusal hem de karesel bir yöntem iyi görünebilir. Fark, veri büyüdükçe ortaya çıkar. Örneğin, 10 öğeyle, 10 adım ve 100 adımın ikisi de hızlıdır. 10.000 öğeyle, 10.000 adım ile 100.000.000 adım arasında büyük bir fark vardır. Her zaman daha büyük veriyi göz önünde bulundur.



Karmaşıklık, zamanın veya belleğin veriyle nasıl büyüdüğünü tanımlar ve Big O bu büyümeyi genel terimlerle ifade eder. Saniyeleri değil, büyüme örüntülerini karşılaştırmayı unutma. O(n) bir yöntem, büyük veri için O(n kare) bir yöntemden daha iyi ölçeklenir. Veri büyüdükçe algoritmaların nasıl davrandığını anlamak için adım sayılarını ve Big O'yu kullan.