Algoritma Analizi

Ders 2: Giriş

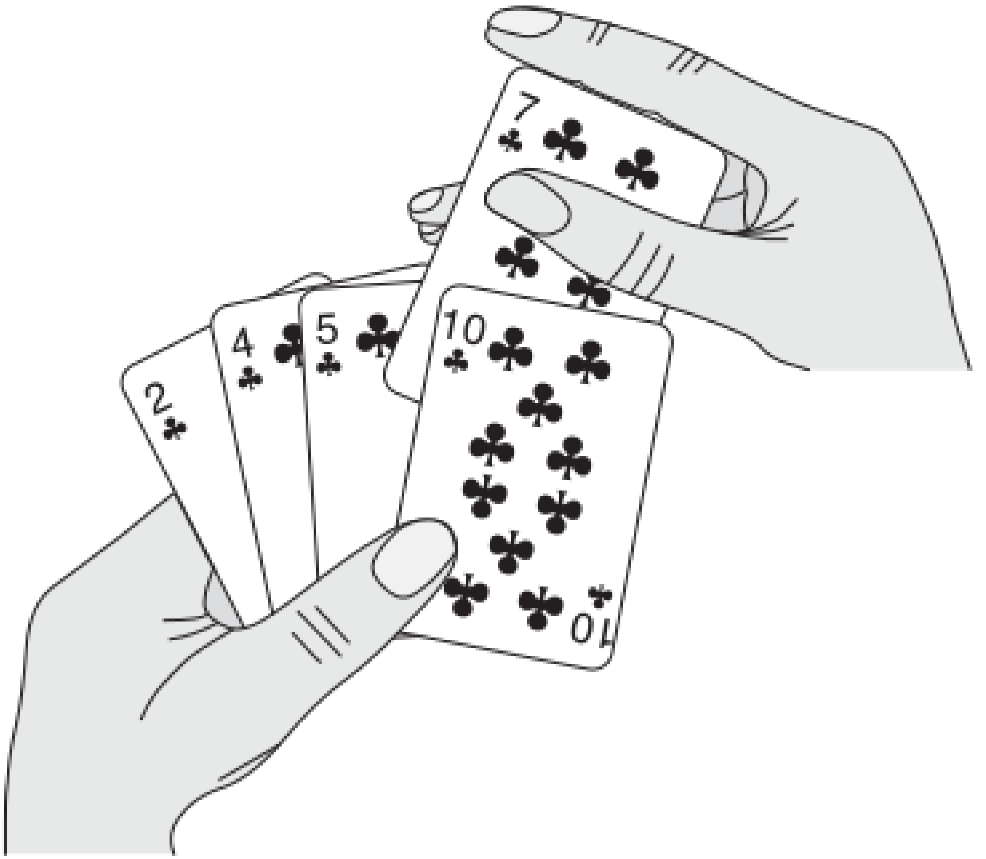
Doç. Dr. Mehmet Dinçer Erbaş
Bolu Abant İzzet Baysal Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Algoritma Analizine Giriş

- Eklemeli sıralama (İng: Insertion sort)
  - Sıralama problemini şu şekilde tanımlamıştık:
    - Girdi: n elemanlı bir sayı dizisi $\{a_1, a_2,\ldots,a_n\}$
    - Çıktı: verilen sayı dizisinin yeni bir sıralaması $\{a_1‘, a_2‘,\ldots,a_n‘\}$, şöyleki $a_1‘ \leq a_2‘ \leq \ldots a_n‘$.
  - Sıralamak istediğimiz elemanlar ayrıca anahtarlar (İng: keys) olarak adlandırılır.
  - Algoritmaları tanımlayabilmek için pseudocode (yalancı kod) kullanacağız.
    - Bu şekilde algoritmanın her basamağını tanımlayacağız.
  - Eklemeli sıralama, az sayıda elemanı hızlı şekilde sıralamak için kullanabilecek bir sıralama algoritmasıdır.
    - Elimize aldığımız bir kart dizisini sıralarken kullandığımız metoda benzemektedir.

Algoritma Analizine Giriş

- Eklemeli sıralama

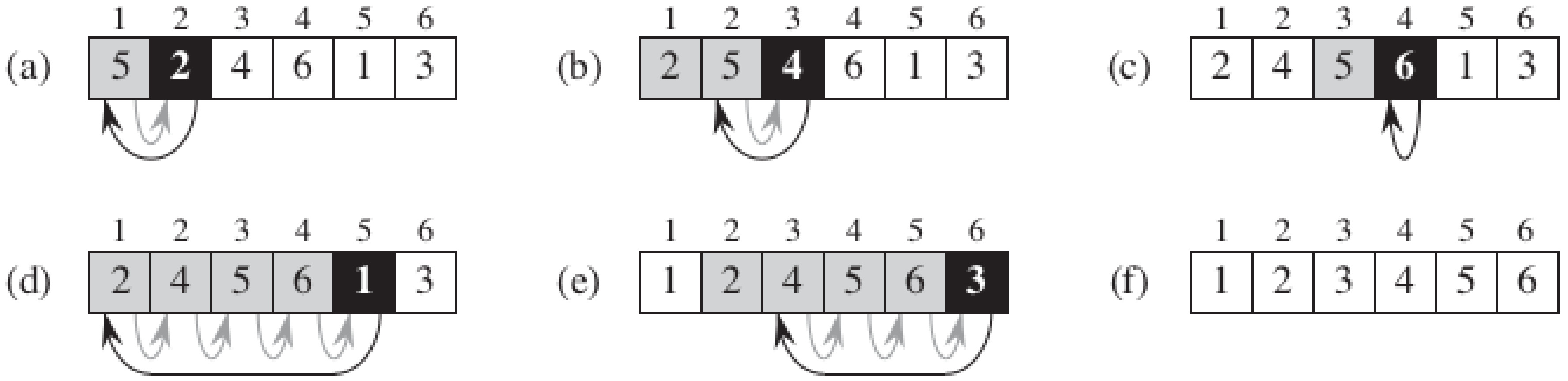


Algoritma Analizine Giriş

```
INSERTION SORT(A)
for j = 2 to A.length
    key = A[j]
    // A[j] elemanını sıralanmış A[1..j-1] dizisine yerleştir.
    i = j - 1
    while i > 0 and A[i] > key
        A[i+1] = A[i]
        i = i - 1
    A[i+1] = key
```

Algoritma Analizine Giriş

- Eklemeli sıralama
  - Örnek: verilen girdi <5, 2, 4, 6, 1, 3>



Algoritma Analizine Giriş

- Dögünün her çalışmasında aşağıdaki özellikler gözlemlenebilir:
  - A[j] şu an sıralayacağımız eleman.
  - Verilen dizinin A[1..j-1] altdizisi doğru sıralanmış durumda.
  - Dizinin geri kalan kısmı A[j+1..n] altdizisi sıralanmayı bekliyor.
- Algoritmanın doğru çalışmasını ispatlamak için A[1..j-1] altdizisinin belirtilen özelliğini döngü sabiti (İng: loop invariant) olarak tanımlıyoruz.
  - 1-8 satırlarında gösterilen döngünün her başlangıcında, A[1..j-1] altdizisi, girdi olarak verilen dizinin ilk j-1 elemanının doğru sıralanmış durumunu içerir.
- Döngü sabitleri bir algoritmanın doğru olduğunu göstermek için kullanılabilir.

Algoritma Analizine Giriş

- Bir algoritmanın doğru çalıştığını gösterebilmek için, döngü sabiti ile ilgili üç durumu göstermeliyiz:
  - Başlangıç: Döngü sabiti döngünün ilk çalışmasında doğrudur.
  - Sürdürme: Döngü sabiti belli bir döngü çalışması öncesi doğru ise bir sonraki döngü çalışması öncesi doğruluğunu korur.
  - Sonlanma: Döngü sonlandığında döngü sabiti bize algoritmanın doğru çalıştığını gösterir.
- Bu metot, matemetiksel tümevarım (İng: Mathematical Induction) yöntemine benzerlik göstermektedir.

Algoritma Analizine Giriş

- Şimdi döngü sabiti metodunu Eklemeli sıralama algoritmasını analiz için kullanalım:
  - Başlangıç: Öncelikle, ana döngünün ilk çalışmasında döngü sabitinin doğru olduğunu göstereceğiz.
    - Döngü ilk olarak çalıştığında j = 2
    - Bu durumda A[1..j-1] altdizisi sadece A[1] elemanını içermektedir.
    - Sadece bir eleman mevcut olduğu için bu altdizi doğru şekilde sıralanmıştır.
  - Sürdürme: Burada döngünün her çalıştığında döngü sabitinin doğru kaldığını göstereceğiz.
    - A[j] elemanının doğru yerini bulmak için A[j-1], A[j-2], A[j-3] … elemanlarını, karşılaştırma sonucu gerektiğinde birer kez sağa çekiyoruz(4-7 satırlar). Bu sayede A[j] elemanının doğru pozisyonunu buluyoruz ve A[j] elemanı bu pozisyona yerleştiriliyor(satır 8).

Algoritma Analizine Giriş

- Şimdi döngü sabiti metodunu Eklemeli sıralama algoritmasını analiz için kullanalım:
  - Sonlanma: son olarak döngü sonlandığında durumu gözlemliyoruz.
    - j>A.uzunluk = n olduğunda döngü sonlanıyor.
    - Her döngü çalıştığında j değerini 1 artırdığımıza göre, son durumda j = n+1 olmalı.
    - Döngü sabitindeki cümledeki j değeri yerine n+1 yazdığımızda şu sonuca varıyoruz:
      - Döngü sonlandığında A[1..n] dizisi, girdi olarak verilen dizinin ilk n elemanının doğru sıralanmış durumunu içerir.
    - Dizide n eleman olduğunda göre, döngü sonlandığında A[1..n] dizisinin tamamı sıralanmıştır.

Algoritma Analizi

- Algoritmaları analiz etmek için bir algoritmanın çalıştığında kullandığı kaynakları belirlemeliyiz.
- Bazı durumlarda hafıza, bağlantı genişliği veya belirli bir donanım kullanılan kaynak olarak hesaplanır.
- Ancak çoğunlukla hesaplayacağımız kaynak **çalışma zamanıdır.**
  - Buna bağlı olarak algoritmaları çalışma zamanları açısından karşılaştıracağız.
- Ayrıca algoritmaların üzerinde çalıştığı teknolojiyi modellememiz gerekiyor.
  - Bu amaçla genel tek işlemcili, RAM (Random Access Machine) modeli kullanan bir sistem kullanacağız.
  - Bu sistemde komutlar birer birer uygulanır, eş zamanlı birden fazla operasyon yapılmaz.

Algoritma Analizi

- RAM modelinde bilgisayarlar tarafından uygulanabilen basit komutlar bulunur
  - Bu komutlar ekle, çıkar, çarp, böl, kalan hesapla, taban ve tavan hesapla vb. Komutlarıdır.
  - Bu komutların her biri sabit birim zamanda gerçekleştirilir.
- RAM modelinde kullanılan veri tipleri tam sayılar (integers) ve gerçel sayılardır (floating point numbers).

Algoritma Analizi

- Eklemeli sıralama algoritmasının analizi:
  - Öncelikle girdi boyutu tanımlanmalıdır.
    - Algoritma sıralama amaçlı kullanıldığı için girdi boyutu sıralanacak eleman sayısı olarak tanımlanabilir.
    - Bazı durumlarda farklı metrikler girdi boyutunu tanımlamada kullanılabilir
      - Örneğin iki tam sayıyı çarpan bir algoritmada, tam sayıları gösterebilmek için kullanılan bit sayısı girdi boyutunu tanımlar.
      - Bir graf algoritması için girdi boyutu graftaki köşe ve kenar sayısı olabilir.

Algoritma Analizi

- Eklemeli sıralama algoritmasının analizi
  - Ikinci olarak çalışma zamanı tanımlanmalıdır.
    - Çalışma zamanı algoritma süresince gerçekleştirilecek temel operasyon veya adım sayısı olarak tanımlanabilir.
    - Herhangi bir adımda yapılacak temel operasyonun sabit bir miktar zaman alacağını farzedeceğiz.
    - Farklı operasyonlar farklı sabit sürelerde zaman alabilir.
      - Bu sebeple *i* numaralı satırda yer alan operasyonun c_i kadar zaman alacağını farzedeceğiz.

Algoritma Analizi

INSERTION SORT(A)

	Süre	Tekrar
`for j = 2 to A.length`	c_1	n
`  key = A[j]`	c_2	n-1
`  // A[j] elemanını sıralanmış` `  // A[1..j-1] dizisine yerleştir.`	0	n-1
`  i = j - 1`	c_4	n-1
`  while i > 0 and A[i] > key`	c_5	$\sum_{j=2}^{n} t_j$
`    A[i+1] = A[i]`	c_6	$\sum_{j=2}^{n} (t_j - 1)$
`    i = i - 1`	c_7	$\sum_{j=2}^{n} (t_j - 1)$
`  A[i+1] = key`	c_8	n-1

Algoritma Analizi

Eklemeli sıralama algoritmasının çalışma süresi aşağıdaki şekilde hesaplanır:

$$T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5\sum_{j=2}^{n} t_j+c_6\sum_{j=2}^{n}(t_j-1)+c_7\sum_{j=2}^{n}(t_j-1)+c_8(n-1)$$

En iyi durumda sıralanacak elemanlar önceden sıralanmış ise Eklemeli sıralama aşağıdaki kadar süre alacaktır:

$$T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5(n-1)+c_8(n-1)$$

$$(c_1+c_2+c_4+c_5+c_8)\,n-(c_2+c_4+c_5+c_8)$$

Yukarıda hesaplanan çalışma süresini an + b şeklinde a ve b sabitleri ile ifade edebiliriz. Bu sebeple bu değer n‘e bağlı lineer (birinci dereceden) bir fonksiyondur.

Algoritma Analizi

- Eklemeli sıralama algoritmasının en kötü durumda çalışma süresi ise şu şekilde hesaplanır:
  - En kötü durumda sıralanacak sayı listesi tersten sıralanmış durumdadır.
  - Bu durum A[1..j-1] elemanlarını sıralamak için j = 2,3,…,n değerleri için döngü listedeki bütün elemanlar için çalışacaktır.

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

Algoritma Analizi

- Eklemeli sıralama algoritmasının en kötü durumda çalışma süresi şu şekilde hesaplanır:

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\
&\quad - (c_2 + c_4 + c_5 + c_8)\ .
\end{aligned}
$$

- En kötü çalışma süresi an^2+bn+c şeklinde a, b ve c sabitleri için yazılabilir.
- Bu sebeple belirtilen değer n‘e bağlı quadratic (ikinci dereceden) bir fonksiyondur.

Algoritma Analizi

- Eklemeli sıralama algoritmasının çalışma süresini analiz ederken iki farklı durum için hesap yaptık.
- Bunlar en iyi durum ve en kötü durum analizidir.
- Bundan sonraki analizlerimizde en kötü durum çalışma süresini inceleyeceğiz. Bunun nedenleri şu şekilde açıklanabilir:
  - En kötü durumda çalışma süresi algoritmanın çalışma süresi için bir üst limit belirtmektedir.
  - Bazı tür algoritmalar için en kötü durum çoklukla karşılaşılabilir.
    - Örneğin bir veritabanında arama yapma.
  - Ortamala çalışma süresi çoğunlukla en kötü çalışma süresine yakın olarak hesaplanır.

Algoritma Analizi

- Eklemeli sıralama algoritmasının çalışma süresi analizi:
  - En kötü durumda çalışma süresinin quadratic bir fonksiyon tarafından hesaplandığı görülmüştür.
    - $T(n) = an^2 + bn + c$
    - Sabitler ve düşük-değerli terimleri görmezden geldiğimizde n^2 içeren bir terim bulunduğunu görüyoruz.
    - Bu sebeple Eklemeli sıralama algoritmasının en kötü çalışma süresi $\Theta(n^2)$ (teta n kare) olarak belirlenir.

Algoritma dizayn yöntemleri

- Algoritma oluştururken farklı yöntemler kullanılabilir
  - Örneğin Eklemeli sıralama kademeli bir yaklaşımla oluşturulmuştur.
    - Her aşamada önceden sıralanmış A[1..j-1] dizisine yeni bir eleman olan A[j]'yi doğru pozisyonuna yerleştiriyor. Bu sayede sıralanmış A[1..j] dizisine sahip oluyoruz.
  - Bu bölümde bir başka sıkça yöntem olan divide-and-conquer (böl-ve-yönet) yöntemini öğreneceğiz.

Böl-ve-yönet yöntemi

- Birçok yararlı ve verimli algoritma recursive (yinelemeli) çalışır.
- Bu yöntemle oluşturulan algoritmalar, problemi çözmek için kendilerini bir veya daha çok kez çalıştırarak birbirleriyle alakalı alt problemleri çözerler.
- Bu yöntem böl-ve-yönet metoduna güzel bir örnektir.
  - Problem daha ufak alt problemlere bölünür, alt problemlere bulunan çözümler birleştirilerek asıl probleme çözüm bulunur.
- Böl-ve-Yönet yöntemiyle çalışan algoritmalar üç aşamadan oluşur:
  - Böl: Asıl problem belli sayıda alt problemlere bölünür.
  - Yönet: Alt problemlerin her biri recursive olarak çözülür. Eğer alt problemler yeteri kadar küçükse direk olarak çözüm hesaplanır.
  - Birleştir: Alt problemlere bulunan çözümler birleştirilir ve asıl probleme çözüm bulunur.

Böl-ve-Yönet yöntemi

- Birleştirmeli Sıralama algoritması (İng: Merge sort)
  - Böl: n elemanlı birleştirilmek istenen diziyi n/2 elemanlı iki altdiziye ayır.
  - Yönet: Önceki adımda oluşturulan altdizileri recursive olarak ayrı şekilde sırala.
  - Birleştir: Önceki adımda sıralanmış olan altdizileri birleştirerek sıralanmış diziyi elde et.
- Yönet adımındaki recursive çağrılar, sıralanmak istenen dizi sadece bir elemana sahip olduğunda sonlanır.
- Bu algoritmanın anahtar işlemi birleştir adımında yapılan birleştirme operasyonudur.

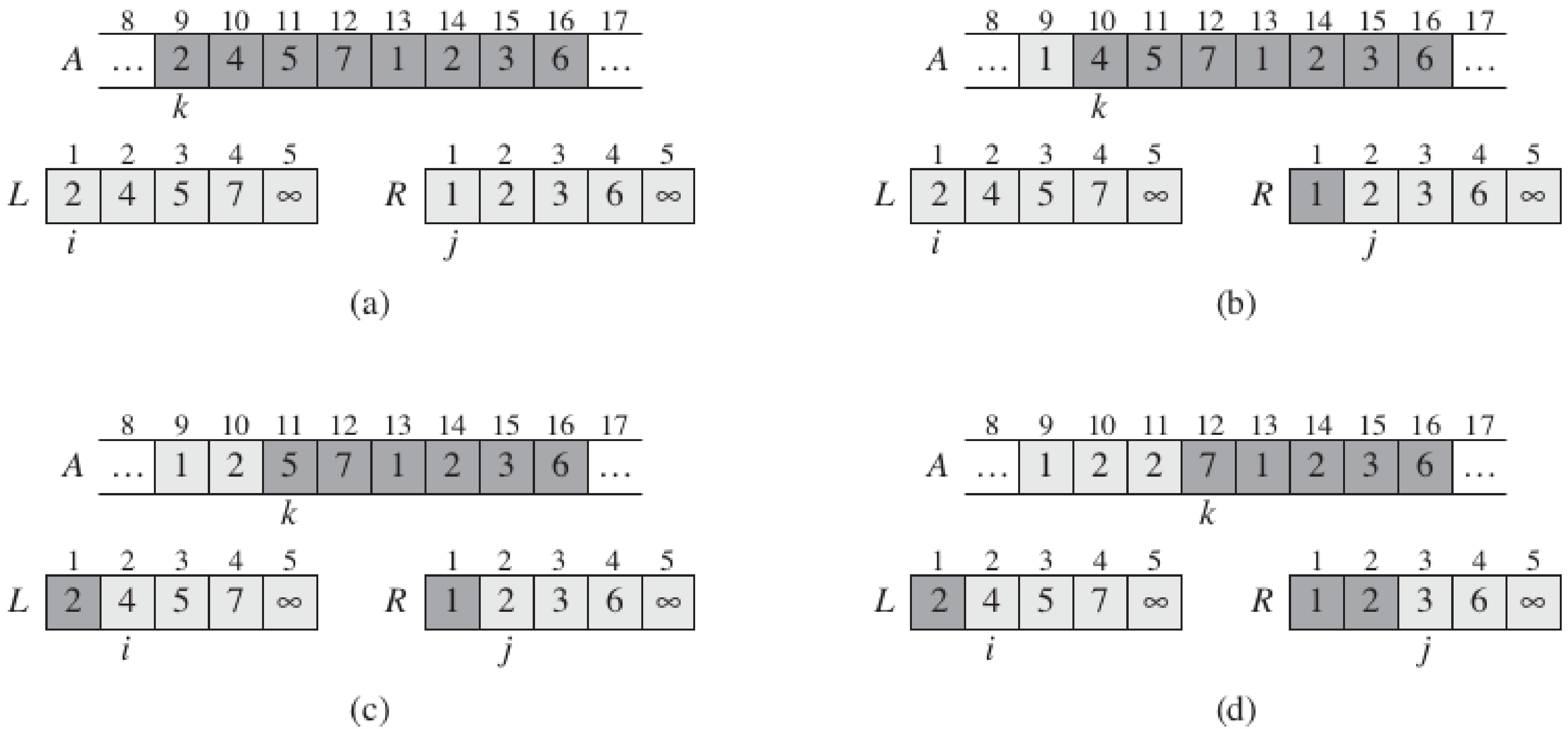
Böl-ve-Yönet yöntemi

- Birleştirmeli sıralama algoritması (birleştirerek sıralama)
  - Birleştir: MERGE(A,p,q,r)
  - A elemanların bulunduğu sıra.
  - p, q, r sıranın elemanlarının indisler, $p \leq q < r$
  - Bu işlem A[p..q] ve A[q+1..r] altdizilerinin sıralanmış olduğunu farzeder ve bu iki sıralanmış altdiziyi birleştirerek A[p..r] sıralanmış altdizisini oluşturur.
  - MERGE işlemi $\Theta(n)$ süre almaktadır ve n değeri sıralanacak eleman sayısıdır ve $n = r - p + 1$.

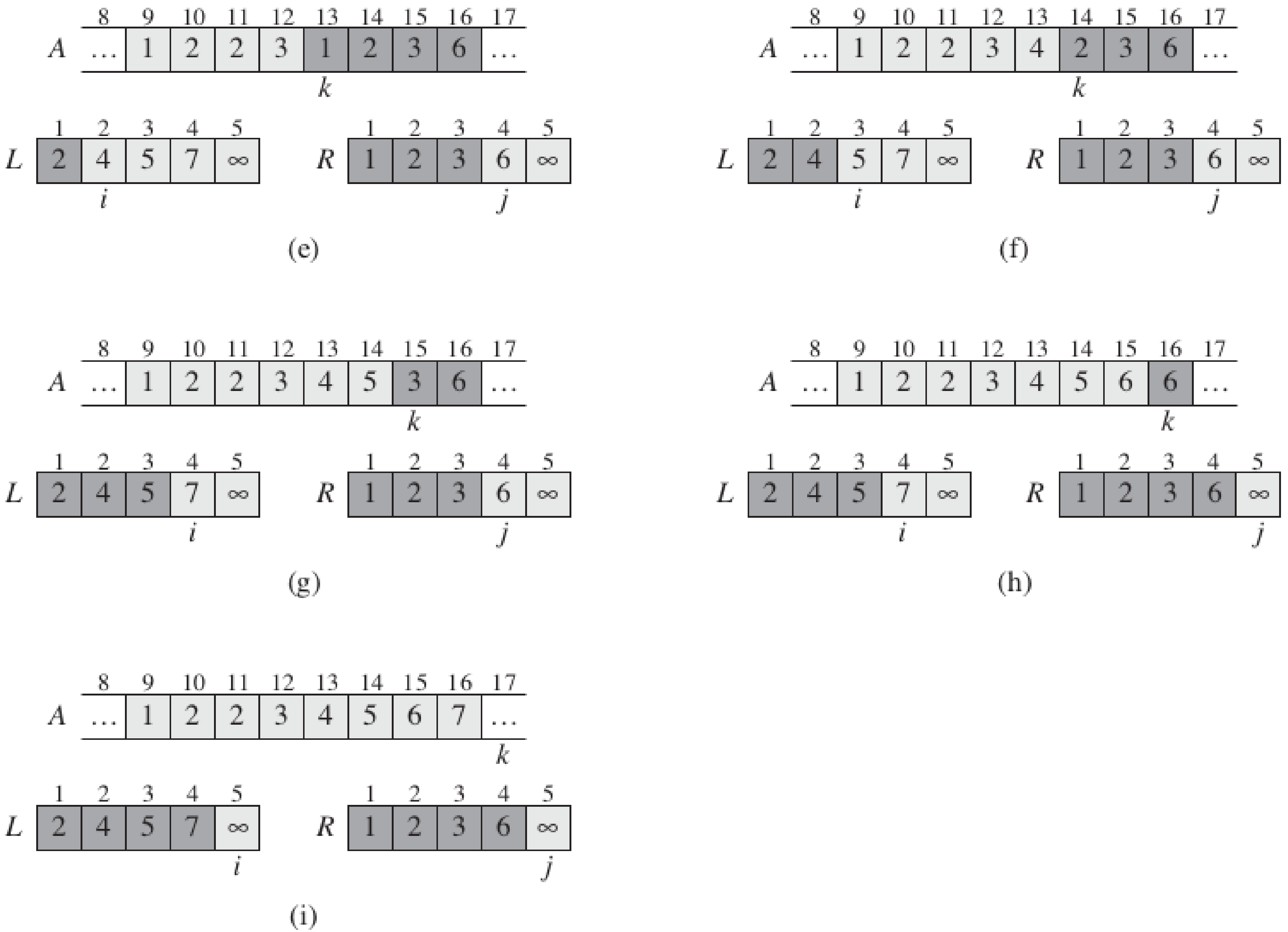
Böl-ve-Yönet yöntemi

```
MERGE(A, p, q, r)
n1 = q − p + 1
n2 = r − q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i − 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

Böl-ve-Yönet yöntemi



Böl-ve-Yönet yöntemi



Böl-ve-Yönet yöntemi

- Birleştirmeli sıralama algoritması
  - MERGE metodundaki döngü sabiti (loop invariant):
    - 12-17 satırları arasındaki döngünün her çalışması öncesinde A[p..k-1] altdizisi, L[1..n_1+1] ve R[1..n_2+1] altdizilerinin en küçük k-p elemanını içermektedir. Ayrıca L[i] ve R[j] ilgili dizilerin en küçük henüz A dizisine kopyalanmayı bekleyen elemanlarıdır.
  - Algoritmanın doğru çalıştığını gösterebilmek için döngü sabitinin döngü sonlandığında geçerli olduğunu göstermeliyiz.
    - Başlangıç: Döngünün ilk çalışması öncesinde k=p durumundadır. Bu durumda A[p..k-1] altdizisi boştur ve L ve R altdizilerinin k-p=0 tane en küçük elemanını içermektedir. Ayrıca i=j=1 olduğu için L[1] ve R[1] ilgili altdizilerin en küçük A dizisine kopyalanmamış elemanlarıdır.

Böl-ve-Yönet yöntemi

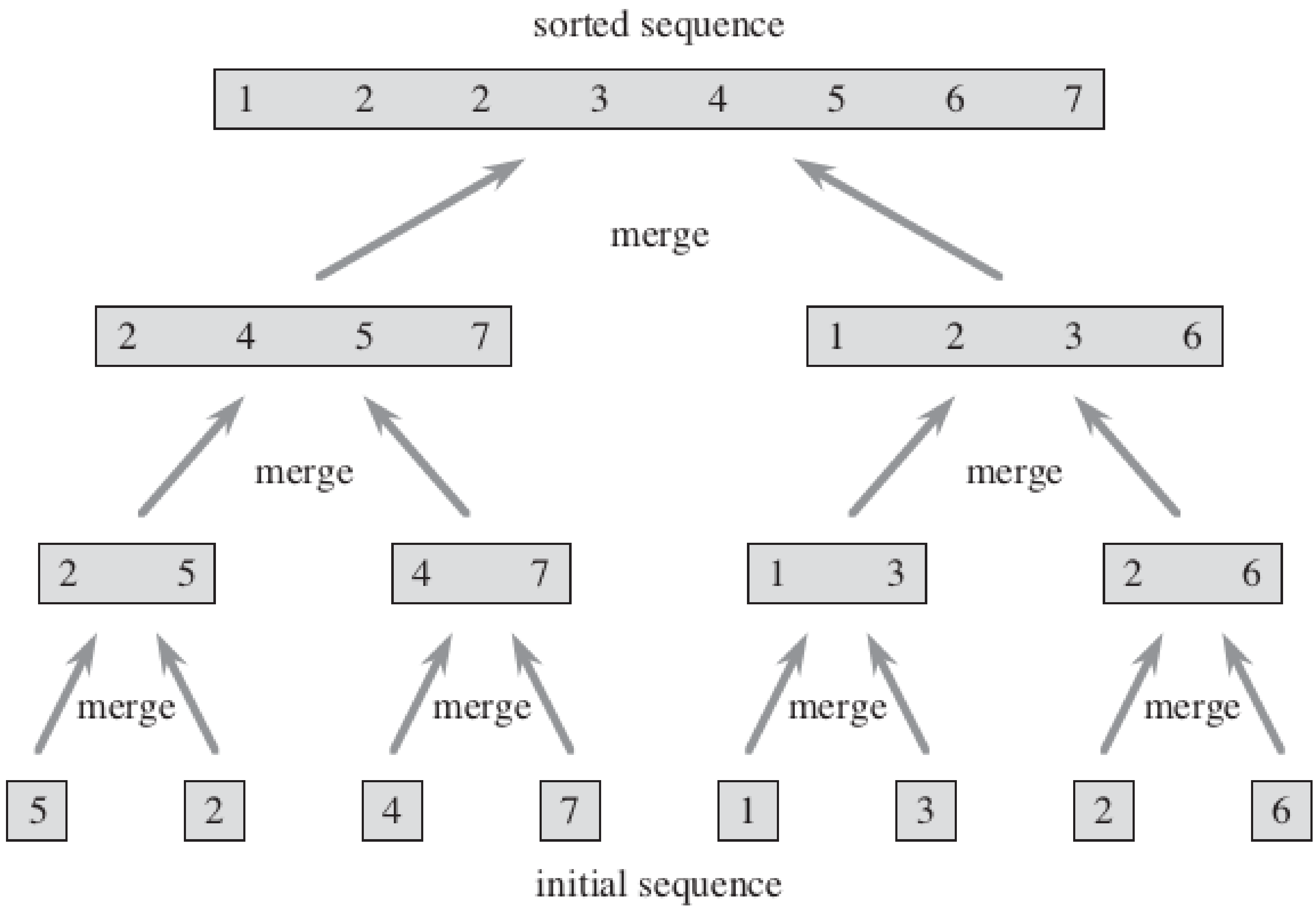
- Birleştirmeli sıralama algoritması
  - Sürdürme: Döngü sabiti doğru kaldığını göstermek için öncelikle L[i] <= R[j] oldğunu farzedelim. A[p..k-1] aldizisi k-p en küçük elemanı içermekte. Satır 14 L[i] elemanını A[k] pozisyonuna yerleştirecek. Sonuç olarak A[p..k] k-p+1 en küçük elemanı içerir. k ve i değerleri birer artınca bir sonraki döngü çalışması öncesinde döngü sabiti doğru olur. R[j] < L[i] ise aynı şekilde işlemler yapılır ve döngü sabiti doğru olarak devam eder.
  - Sonlanma: Döngü sonlandığında k = r+1. Döngü sabitine göre A[p..k-1] altdizi, A[p..r] altdizisi olmuştur ve k-p = r-p+1 tane L[1..n_1+1] ve R[1..n_2+1] altdizilerinin en küçük elemanlarını içermektedir. L ve R altdizileri toplam $n_1+n_2+2 = r – p + 3$ eleman içermektedir. Son iki en büyük eleman dışındaki elemanlar A dizisine geri kopyalanmıştır ve bu iki eleman sonradan eklediğimiz sentinel değerleridir.

Böl-ve-Yönet yöntemi

- MERGE işlemi Θ(n) zaman almaktadır.
  - 1-3 ve 8-11 satırlarında yapılan işlemler sabit zaman almaktadır.
  - 4-7 satırlarındaki döngü $\Theta(n_1+n_2) = \Theta(n)$ zaman almaktadır.
  - 12-17 satırlarındaki döngü n kez çalışmakta ve her döngü sabit zaman almaktadır.
- MERGE işlemi kullanarak Birleştirmeli sıralama algoritması şu şekilde oluşturulabilir:

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

Böl-ve-Yönet yöntemi



Böl-ve-Yönet yöntemi

- Böl-ve-Yönet algoritmalarının analizi
  - T(n), algoritmanın n büyüklüğünde girdi ile çalışma süresi şu şekilde hesaplanabilir:
    - Problem büyüklüğü her recursive çağrıda daha küçülecektir.
    - Problem büyüklüğü yeteri kadar küçüldüğünde, bu küçük problem tek bir işlem ile çözülebilir. Bu işlem Θ(1) zaman alacaktır.
    - Problemi her böldüğümüzde a tane yeni küçük problem oluşuyor ve bu problemler asıl problemin 1/b'si büyüklüğünde ise
    - Ayrıca problemi bölme işlemi D(n) kadar zaman alıyor ve küçük problemlerin çözümlerini birleştirmek C(n) kadar zaman alıyorsa

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c \text{ ,} \\ aT(n/b) + D(n) + C(n) & \text{otherwise .} \end{cases}$$

Böl-ve-Yönet yöntemi

- Birleştirmeli sıralama algoritmasının analizi
  - Birleştirmeli sıralama algoritmasının üç aşaması bulunmaktadır:
    - Böl: Altdizinin orta noktası hesaplanıyor. Bu işlem sabit zaman alınır. Sonuç olarak D(n) = Θ(1).
    - Yönet: Recursive şekilde oluşturulan iki küçük problemi çözüyor ve bu problemlerin büyüklüğü n/2. Bu durumda toplam süreye 2T(n/2) ekleniyor.
    - Birleştir: MERGE işleminin Θ(n) süre aldığını önceden göstermiştik. Sonuç olarak C(n) = Θ(n).

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \, , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \, . \end{cases}$$

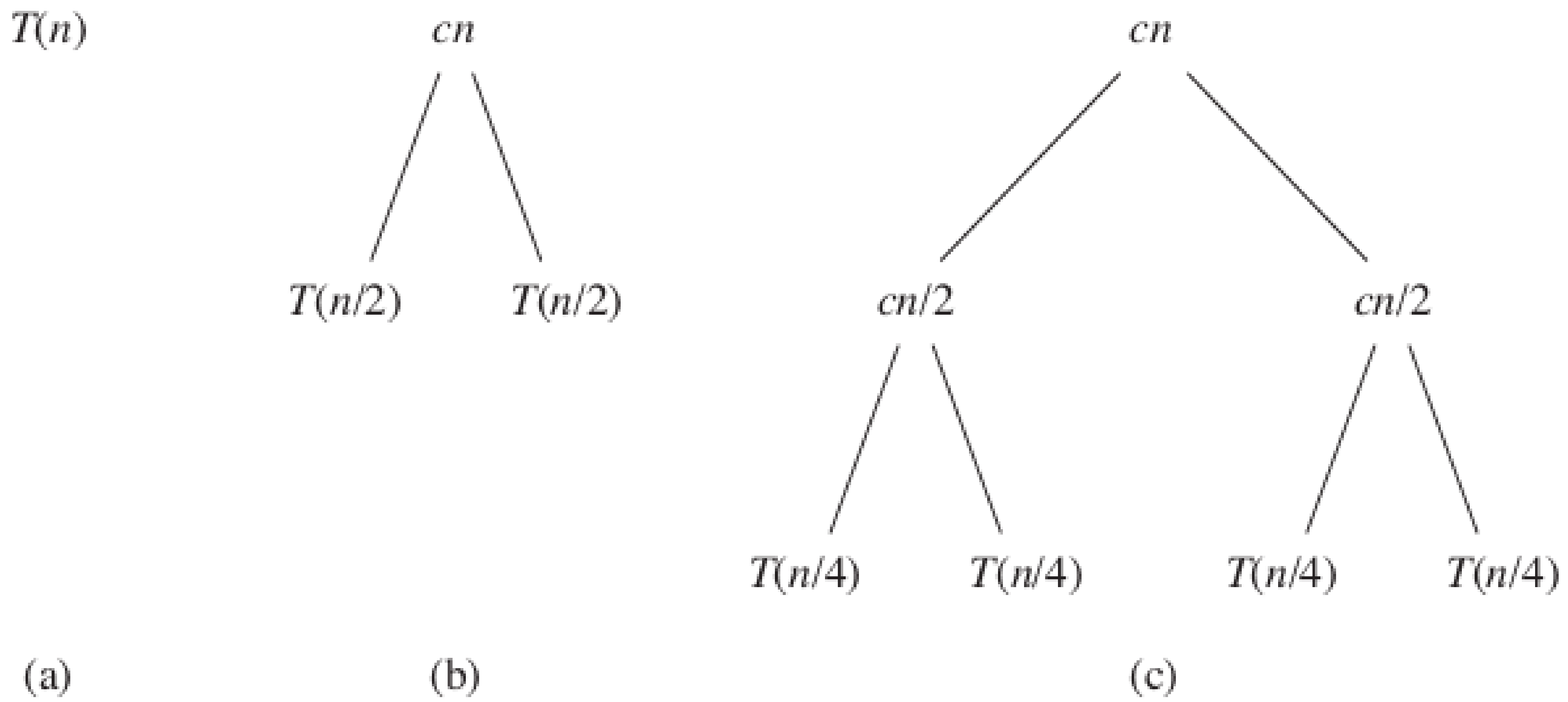
Böl-ve-Yönet yöntemi

- Birleştirmeli sıralama algoritmasının analizi

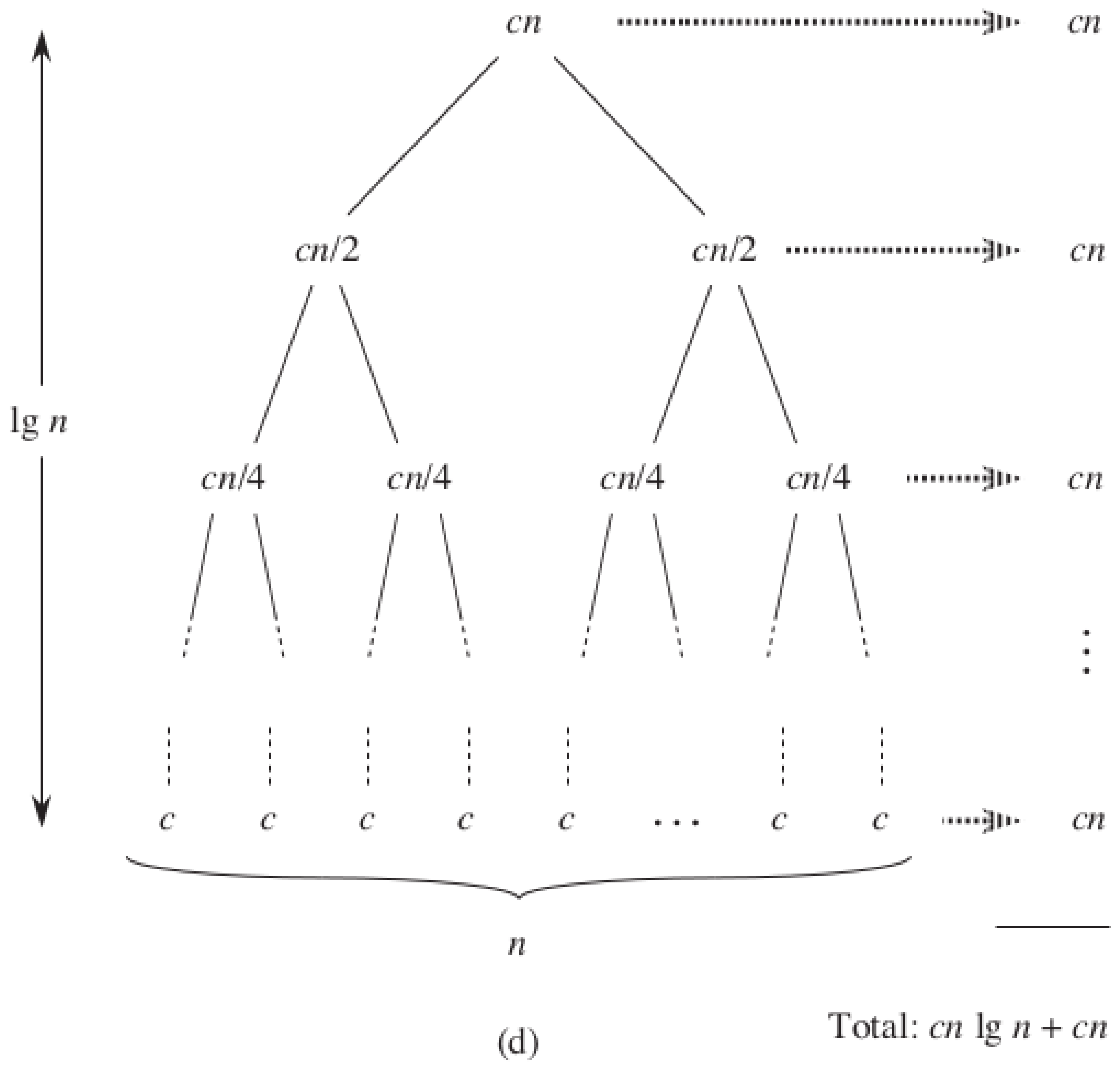
$$T(n) = \begin{cases} c & \text{if } n = 1\ , \\ 2T(n/2) + cn & \text{if } n > 1\ , \end{cases}$$

- c, bir uzunluğundaki en basit problemin çözüm süresi ve combine adımındaki birleştirilen elemanların her biri için harcanan süredir.
- Analizimizi basitleştirmek için n sayısının 2^i şeklinde yazılabileceğini farzedeceğiz.

Böl-ve-Yönet yöntemi



Böl-ve-Yönet yöntemi



Böl-ve-Yönet yöntemi

- Tepe noktasının altındaki i numaralı katman $2^i c\left(n/2^i\right) = cn$ zaman alıyor
- Katman sayısı ile lg n + 1
  - Tümevarım ile ispatlayabiliriz.
  - Taban durumu => n = 1, lg n + 1 = 0 + 1 = 1
  - Hipotez: 2^i yaprak içeren bir ağacın kademe sayısı lg n + 1 = lg 2^i + 1 = i + 1
  - Buna göre bir sonraki kademedeki ağacın kademe sayısı i + 2 olmalı
    - Sonraki kademe: lg (2^{i+1}) + 1 = lg(2^i x 2) + 1 = (lg 2^i + 1) + 1 = i+1 + 1 = i+2
- Öyleyse, n yaprağı olan ağacın katman sayısı lg n + 1.
- Bu durumda toplam süre cn(lg n + 1) = cn lg n + cn
- Düşük dereceli terimler ve sabitleri görmezden gelirsek Θ(n lg n).