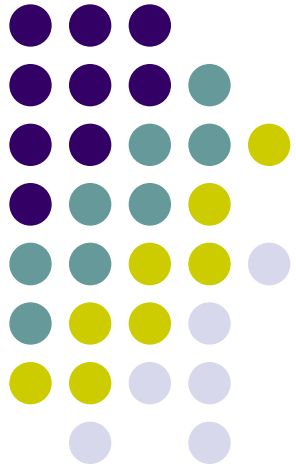
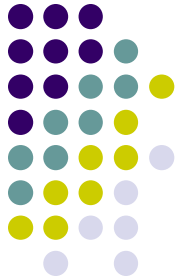
Introduction to Algorithm Design

Lecture Notes 6

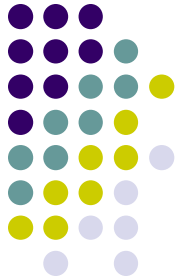


ROAD MAP



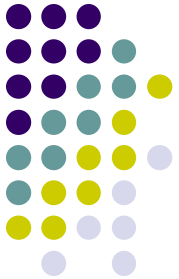
- **Decrease And Conquer**
  - Insertion Sort
  - Depth-First Search
  - Breadth-First Search
  - Topological Sorting
  - Algorithms For Generating Combinatorial Objects
  - **Decrease By a Constant-Factor Algorithms**
  - Variable-Size-Decrease Algorithms

Decrease By a Constant-Factor Algoritmaları



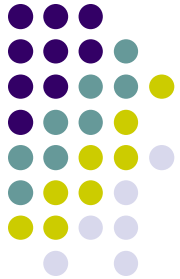
- Şimdiye kadar çeşitli örneklerini gördük.
  - Binary search
  - Karesini alma ile üs alma
- Decrease by a constant-factor fikri üzerine kurulmuş başka algoritmalar göreceğiz.
  - *Fake-Coin Problemi, Josephus Problemi, Russian Multiplication*
- Bu algoritmalar hızlı çalışır.
  - Genellikle logaritmik zamanda

Fake-Coin Problemi (Sahte-Para Problemi)



- Fake-coin probleminin çeşitli versiyonları bulunmaktadır.
- **Tanım:**
  - Elimizde eşit kollu terazi ve n tane aynı şekilde görülen madeni para bulunmaktadır.
  - Bir tanesinin daha hafif olduğunu varsayın.
    - Terazide hangi kümenin daha ağır çektiğini görebiliyoruz ama ağırlığının ne olduğunu (kaç gr olduğunu) bilemiyoruz.
    - Sahte olan parayı bulunuz.

Fake-Coin Problemi



- **Yaklaşım:**
  - N tane madeni parayı ikiye bölünüz her birinde n/2 tane madeni para olsun. Eğer n tek sayı ise 1 tane madeni parayı kenara ayırıyoruz. Diğer iki grupta (n-1)/2 tane madeni para bulunuyor.
  - Eğer kümeler eşit çekilirse, sahte olan para kenara ayırdığımızdır. Değilse hafif gelen kümeden devam edebiliriz.
  - Problem ikiye bölündükten sonra problem halif olan kümeden işlem yapmaya devam eder yani decrease and conquer'dir.

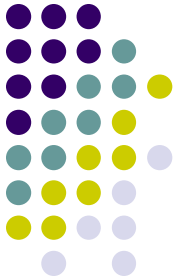
Fake-Coin Problemi

- *W(n)* en kötü durumda gerçekleştirilmesi gereken ölçüm sayısı olsun.

$$W(n) = W(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1,\ W(1) = 0$$

- Binary search'de yapılması gereken karşılaştırma sayısına neredeyse eşittir.
  - Tek fark initial condition(ilk koşul)'dır.

Fake-Coin Problemi

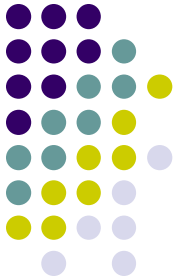


- Analiz :
  - Recurrence'ın çözümü

$$W(n) = \lfloor \log_2 n \rfloor$$

  - Daha verimli bir algoritma mümkün müdür?

Fake-Coin Problemi



- **Tartışma:**
  - Madeni paraları üçe bölmek daha iyi bir çözüm verebilir. Her kümede neredeyse **n/3** tane madeni para içermektedir.
  - İki kümeyi eşit kollu terazide tarttıktan sonra, problem boyutunu üçe düşürebiliriz.
  - Toplam tartma sayısının yaklaşık olarak $\log_3 n$ **'e** indiğini görebiliriz.
    - Kaç katı olduğunu söyleyebilir misiniz?

Russian Peasant Multiplication

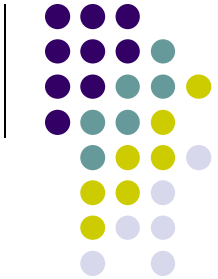
nm'i hesaplamak istiyoruz.

n çift ise

$$n \cdot m = \frac{n}{2} \cdot 2m.$$

n tek ise

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m.$$



n	m	
50	65	
25	130	
12	260	(+130)
6	520	
3	1040	
1	2080	(+1040)
	2080	$+(130 + 1040) = 3250$

(a)

n	m	
50	65	
25	130	130
12	260	
6	520	
3	1040	1040
1	2080	2080
		3250

(b)

FIGURE 4.11 Computing $50 \cdot 65$ by the Russian peasant method.

JOSEPHUS PROBLEMİ

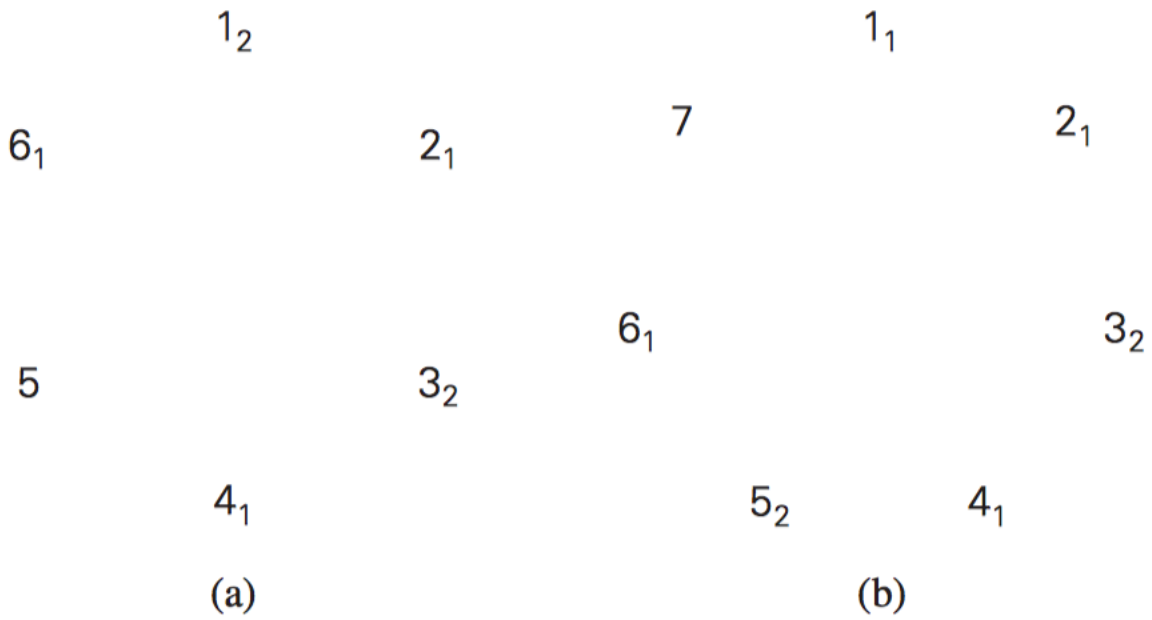
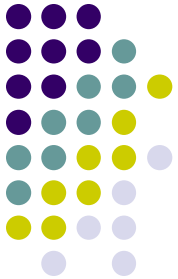


FIGURE 4.12 Instances of the Josephus problem for (a) $n = 6$ and (b) $n = 7$. Subscript numbers indicate the pass on which the person in that position is eliminated. The solutions are $J(6) = 5$ and $J(7) = 7$, respectively.

JOSEPHUS PROBLEMİ



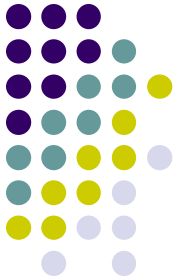
- Kişi Sayısı Çift ise

$$J(2k) = 2J(k) - 1.$$

- Kişi Sayısı Tek İse

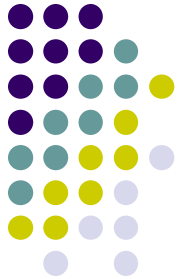
$$J(2k+1) = 2J(k) + 1.$$

ROAD MAP



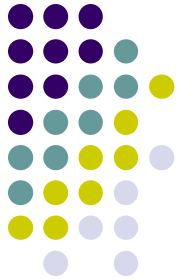
- **Decrease And Conquer**
  - Insertion Sort
  - Depth-First Search
  - Breadth-First Search
  - Topological Sorting
  - Algorithms For Generating Combinatorial Objects
  - Decrease By a Constant-Factor Algorithms
  - **Variable-Size-Decrease Algorithms**

Variable-Size-Decrease Algoritmalar



- **Tanım:**
  - Boyutu düşürme deseni bir iterasyondan diğerine değişebilir.
  - Örnekler
    - OBEB'i bulan Euclid'in algoritması
    - Selection problem
    - Ortanca elemanı hesaplama (Computing median)

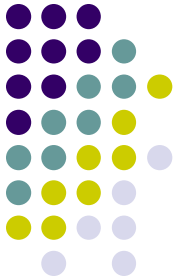
Selection Problemi



Problem Tanımı:

- n elemanlı bir listedeki k'ıncı en küçük elemanı bulma
  - Bu sayı *k*'ıncı order statistics olarak adlandırılır.
  - *k=1* ya da *k=n* ise sırasıyla listenin en küçük ve en büyük elemanlarını buluruz.
  - Daha ilginç olan ise *k=n/2 içindir*
    - Bu değer **ortanca** (***median)*** olarak adlandırılır.
      - İstatistikte en önemli değerlerden bir tanesidir.
      - Mergesort ile k'ıncı en küçük elemanı bulmanın zaman karmaşıklığı *O(nlogn)'dir.*

Selection Problem



- **Yaklaşım:**

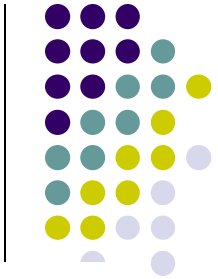
  k'ıncı en küçük elemanı bulmak için listeyi iki alt gruba ayırabiliriz.
  - p'den küçük olanlar
  - p'den büyük ya da p'ye eşit olanlar

$$\underbrace{a_{i_1} \ldots a_{i_{s-1}}}_{\leq p} \quad \mathrm{p} \quad \underbrace{a_{i_{s+1}} \ldots a_{i_n}}_{\geq p}$$

  - Quicksort'un prensipi!
  - Bu liste parçalamanın avantajı var mıdır?

ALGORITHM *LomutoPartition*$(A[l..r])$

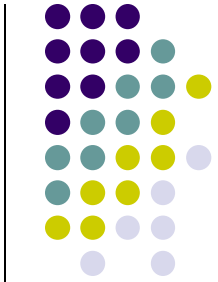
//Partitions subarray by Lomuto's algorithm using first element as pivot
//Input: A subarray $A[l..r]$ of array $A[0..n-1]$, defined by its left and right
// indices l and r $(l \le r)$
//Output: Partition of $A[l..r]$ and the new position of the pivot
$p \leftarrow A[l]$
$s \leftarrow l$
for $i \leftarrow l+1$ **to** r **do**
  if $A[i] < p$
    $s \leftarrow s+1$; swap$(A[s], A[i])$
swap$(A[l], A[s])$
return s



ALGORITHM *Quickselect*$(A[l..r], k)$

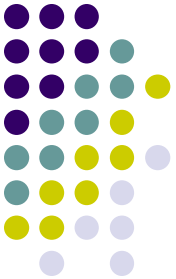
//Solves the selection problem by recursive partition-based algorithm
//Input: Subarray $A[l..r]$ of array $A[0..n-1]$ of orderable elements and
// integer k $(1 \le k \le r - l + 1)$
//Output: The value of the kth smallest element in $A[l..r]$
$s \leftarrow$ *LomutoPartition*$(A[l..r])$ //or another partition algorithm
if $s = k - 1$ **return** $A[s]$
else if $s > l + k - 1$ *Quickselect*$(A[l..s-1], k)$
else *Quickselect*$(A[s+1..r], k - 1 - s)$

Selection Problemi



- *s* listenin pivota göre ayrılma noktası olsun. (k. En küçük için)
- Aynı zamanda pivotun da pozisyonunu veriyor.
- If **`s=k-1`**
  - Pivot p selection probleminin çözümüdür
- If **`s>k-1`**
  - Parçalanmış array'in sol tarafında aramaya devam
- If **`s<k`**
  - Arrayin sağ kısmında (k-s-1)'inci en küçüğü aramaya devam.

Selection Problem Örnek



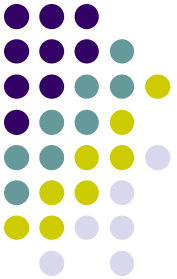
- Örnek:
  - Aşağıdaki listenin ortanca elemanını bulma:
    4, 1, 10, 9, 7, 12, 8, 2, 15

    - $k = \lceil 9/2 \rceil = 5$
    - Listedeki 5'inci en küçük elemanı bulma
    - Listedeki elemanların 0 dan 8'e indexlendiğini varsayıyoruz.
    - İlk elemanı pivot olarak alıyoruz.

Selection Problemi Örnek

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s					*i*	
4	1	2	8	7	12	9	10	15
		s						*i*
4	1	2	8	7	12	9	10	15
2	1	**4**	8	7	12	9	10	15

Selection Problemi Örnek



0	1	2	3	4	5	6	7	8
			s	*i*				
			8	7	12	9	10	15
				s	*i*			
			8	7	12	9	10	15
				s				*i*
			8	7	12	9	10	15
			7	**8**	12	9	10	15

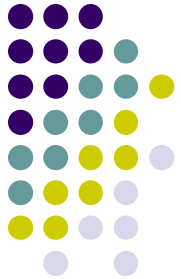
Selection Problemi

- **Analiz:**
  - Quicksorttan daha maliyetli
  - En iyi durumda ilk seferde bulur.

$$C(n) \in \Theta(n)$$

  - Ortalama durumda
    - C(*n*) = C(*n*/2)+(*n*+1) C(*n*) ∈ Θ(*n*)

Selection Problemi

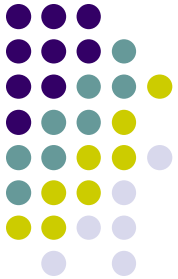


- **Analiz:**
  - En kötü durumda, sadece bir eleman ve geriye kalan elemanlar olarak parçalanabilir.
  - Yani,

$$C(n) = C(n-1) + (n+1)$$

$$C(n) \in \Theta(n^2)$$

Selection Problem



- **Tartışma:**
  - Ortalama zaman karmaşıklığı lineerdir.
  - Tüm durumlarda lineer çalışan başka bir algoritma da bulunmuştur.
    - Pratik uygulamalarda kullanmak için çok karmaşıktır.
  - Paçalama tabanlı problem istenilenden fazlasını çözmektedir.
  - Listedeki k'ıncı en küçük ve n-k'ıncı en büyük elemenı bulmaktadır.
    - Sadece k'ıncı en küçük elemanın değerini değil.