VERİ YAPILARI VE ALGORİTMALAR

BLM2512 Gr.2

2020-2021 Bahar Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

GRAF ARAMA YÖNTEMLERİ

BFS, DFS

BFS Algoritması

```
BFS(V, E, s)
for each u ∈ V −{s}
   do d[u]←∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q != ∅
   do u ← DEQUEUE(Q)
      for each v ∈ Adj[u]
         do if d[v]=∞
            then d[v] ← d[u] + 1
                 ENQUEUE(Q, v)
```

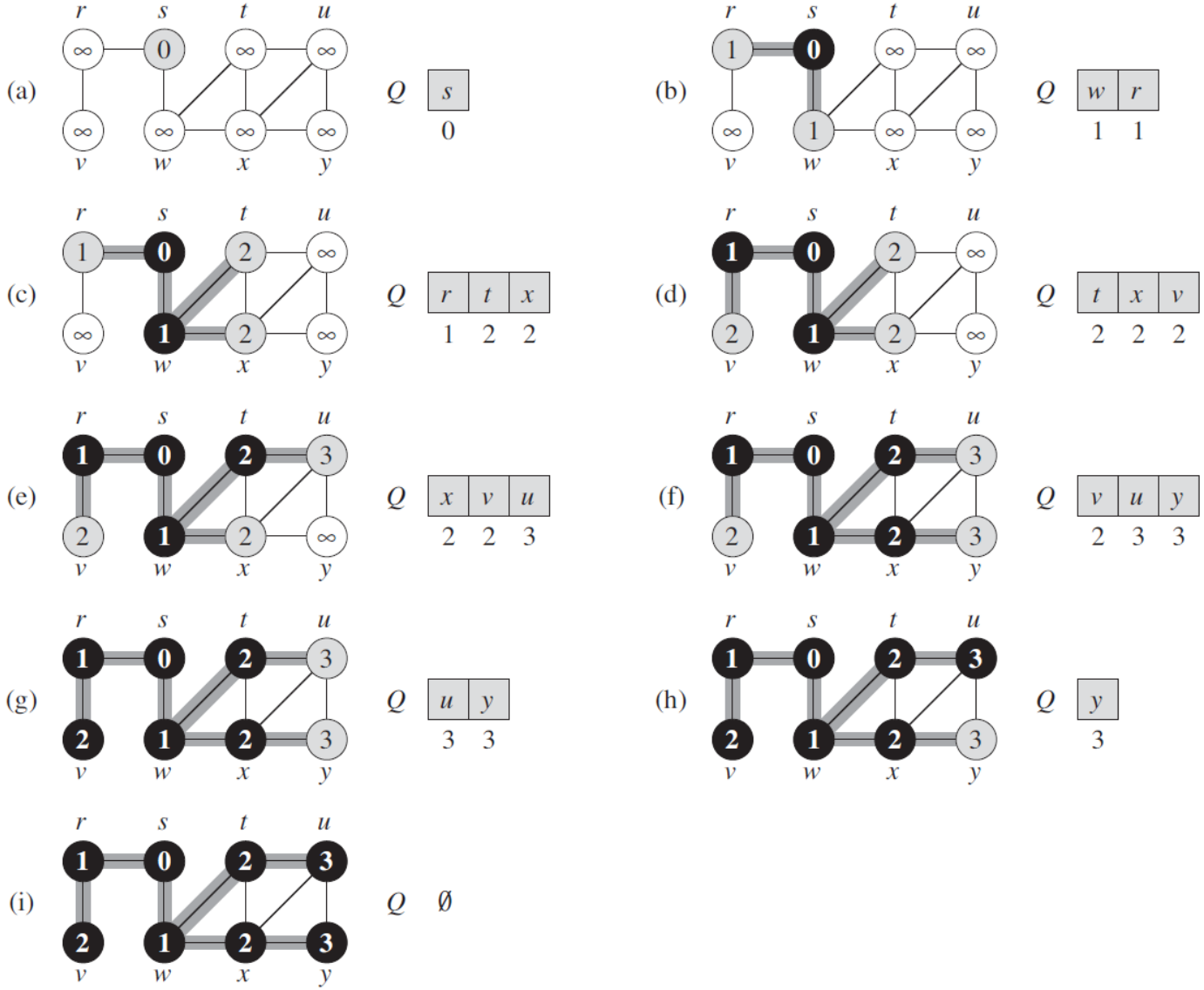
(Renkli) BFS Algoritması

- Düğümlerin durumlarını takip etmek isteyebiliriz.
  - Gezilmemiş: Beyaz
  - Gezilmiş Gri/Siyah
  - Siyahların komşuları siyah/gri olabilir. (siyahların tüm komşuları keşfedilmiştir)
  - Grilerin komşuları gri/beyaz olabilir.

(Renkli) BFS Algoritması

```
BFS (G,s)
for each vertex u ∈ G.V – {s}
   u.color = WHITE
   u.d = ∞
   u.p = NIL
s.color = GRAY
s.d = 0
s.p = NIL
Q = { } ;
ENQUEUE(Q,s)
while Q != { }
   u = DEQUEUE(Q)
   for each v ∈ G.Adj[u]
      if v.color == WHITE
         v.color = GRAY
         v.d = u.d + 1
         v.p = u
         ENQUEUE(Q,v)
    u.color = BLACK
```

(Renkli) BFS Algoritması



Önce Derinlemesine Arama (Depth First Search, DFS)

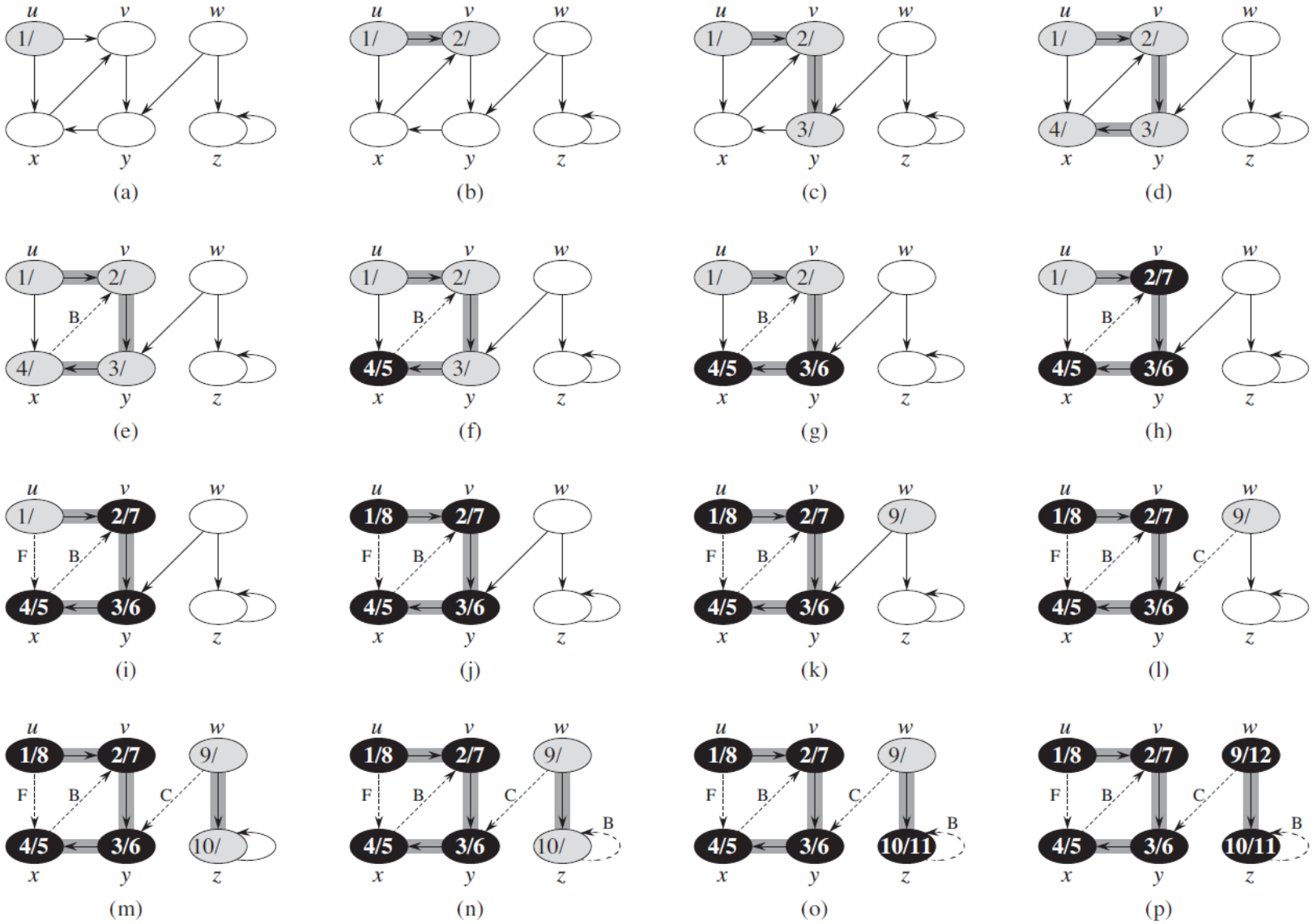
- Düğümleri rekürsif olarak ziyaret et.
- Bir düğümü al, işaretle.
- Tüm (işaretsiz) komşularını rekürsif olarak ziyaret et.

```
DepthFirstSearch(G,s)
   count=0;
   for each vertex u ∈ G.V
      marked[u]=FALSE
   dfs(G,s)

dfs(G,v)
   marked[v]=TRUE
   count++
   for each w ∈ G.adj[v]
      if (! marked[w])
         dfs(G,w)
```

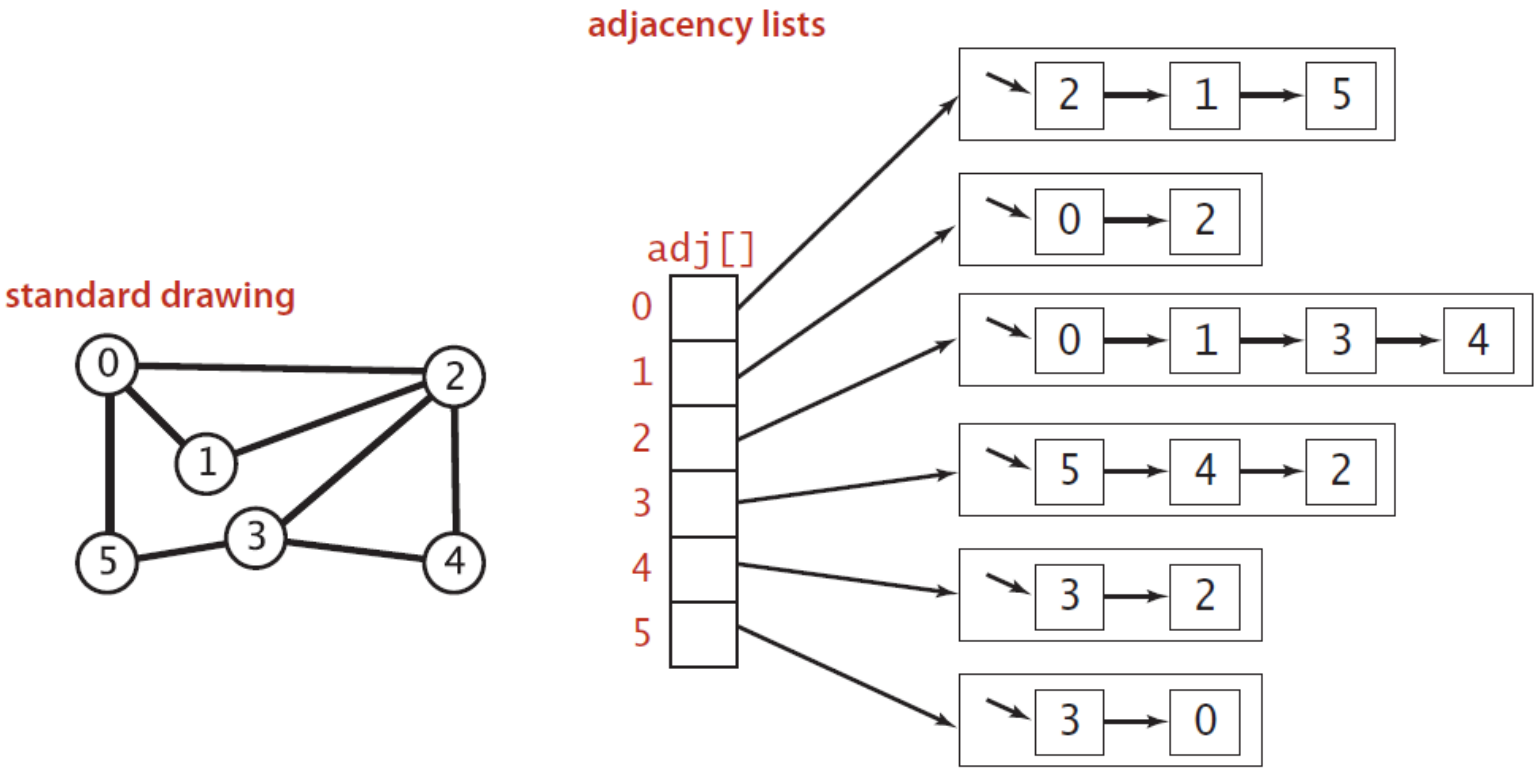
(Renkli) DFS Algoritması

```
DFS(V, E)
for each u ∈ V
   do color[u] ← WHITE
time ← 0
for each u ∈ V
   do if color[u] = WHITE
            then DFS-VISIT(u)

DFS-VISIT(u)
color[u] ← GRAY  // discover u
time ← time+1
d[u] ← time
for each v ∈ Adj[u]  // explore (u, v)
   do if color[v] = WHITE
            then DFS-VISIT(v)
color[u] ← BLACK
time ← time+1
f [u] ← time   // finish u
```

(Renkli) DFS Algoritması

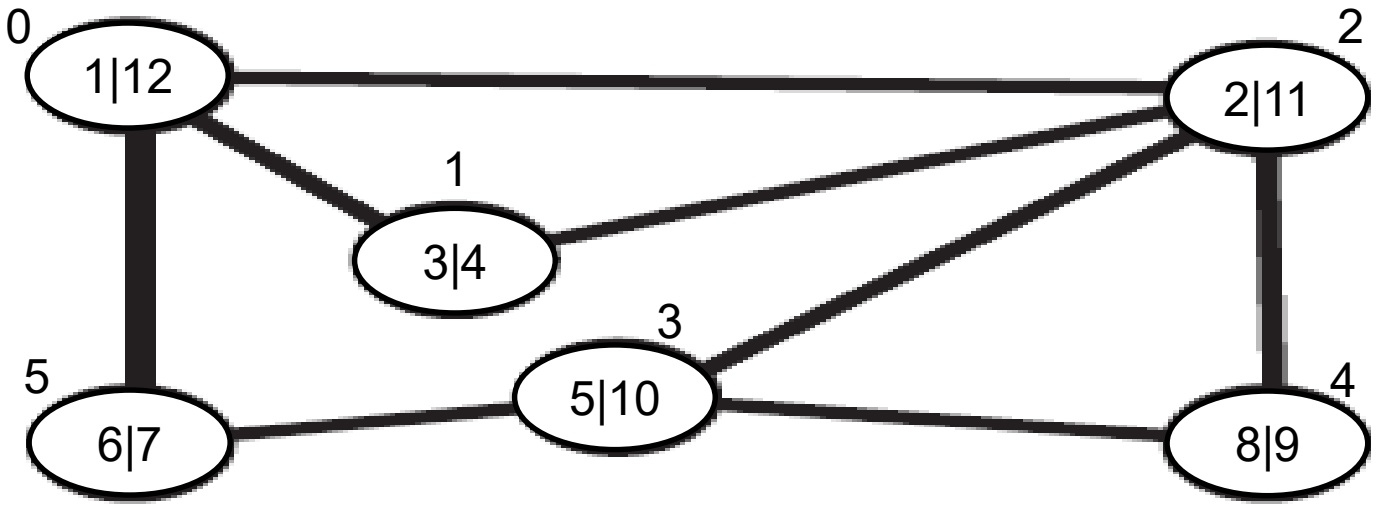


(Renkli) DFS Algoritması



(Renkli) DFS Algoritması

dfs(0)
2 1 5
dfs(2)
0 1 3 4
dfs(1)
0 2
dfs(3)
5 4 2
dfs(5)
3 0
dfs(4)
3 2



	d	f	color			Pre
0	1	12	W	G	B	
1	3	4	W	G	B	2
2	2	11	W	G	B	0
3	5	10	W	G	B	2
4	8	9	W	G	B	3
5	6	7	W	G	B	3

AÇGÖZLÜ ALGORİTMALAR

Greedy Algorithms

Greedy Algoritmalar

- Probleme, bir dizi seçimden geçerek parça parça çözüm oluştururlar. Bu seçimler:
  - yapılabilir (feasible)
  - yerel en iyi (locally optimal)
  - geri alınamaz (irrevocable)

olmalıdır.

- Bazı problemlerin tüm örnekleri için optimal çözüm sunarlar.
- Çoğu problem için bunu başaramasa da, hızlı yakınsama nedeniyle tercih edilirler.

Greedy Algoritma Örnekleri

- Optimal çözümler:
  - Bozuk para üstü vermek.
  - Minimum spanning tree (göreceğiz)
  - Single-source shortest paths
  - Basit çizelgeleme problemleri
  - Huffman kodlama
- Yaklaşımlar:
  - Gezgin satıcı problemi (Traveling salesman)
  - Sırt çantası problemi (Knapsack)
  - Diğer kombinatoryal optimizasyon problemleri

Bozuk Para Üstü Vermek

- Bozuk paralarımız: 1TL, 50Kr, 25Kr, 10 Kr, 5 Kr, 1 Kr ? :))
- 48 kuruş para üstü verelim.
  - 1 tane 25 Kr. (23 Kr kaldı)
  - 1 tane 10 Kr. (13 Kr kaldı)
  - 1 tane 10 Kr. (3 Kr kaldı)
  - 1 tane 1 Kr (2 Kr kaldı)
  - 1 tane 1 Kr (1 Kr kaldı)
  - 1 tane 1 Kr (0 Kr kaldı)

- Her adımda, kalan miktarı en az yapacak olan bozuk parayı ver.

Huffman Kodlama

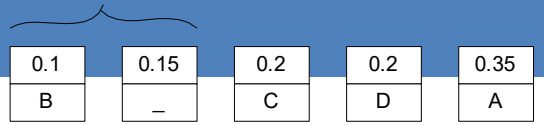
karakter	A	B	C	D	_
frekans	0.35	0.1	0.2	0.2	0.15

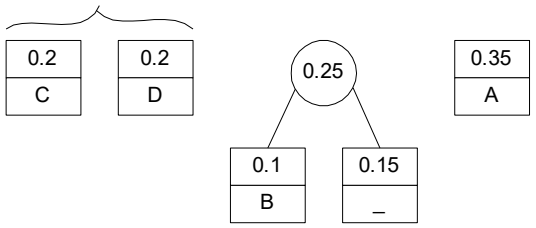
Kod 11 100 00 01 101

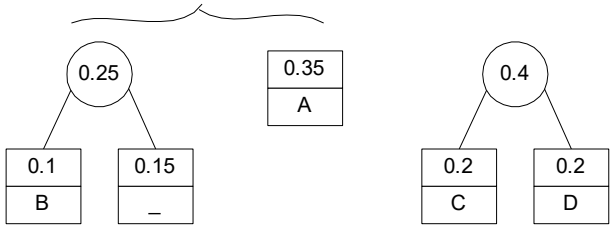
Karakter başına ortalama bit: 2.25

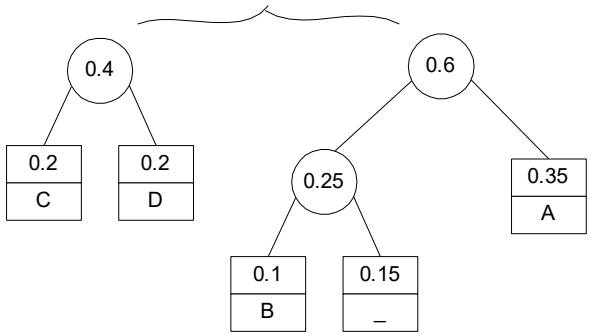
Sabit uzunlukl kodlama için: 3

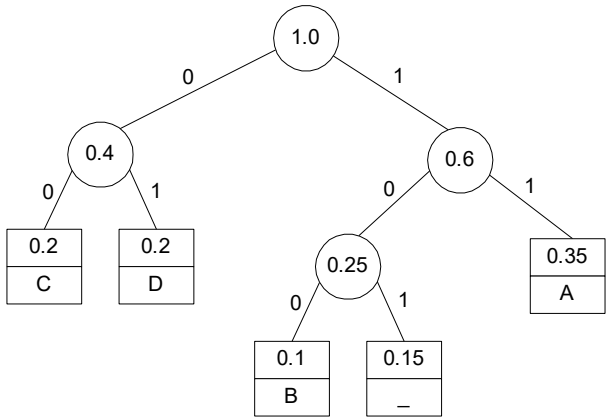
Sıkıştırma oranı: (3-2.25)/3*100% = 25%











ÖZYİNELEMELİ ALGORİTMALAR

Recursive Algorithms

Özyinelemeli Algoritmalar

- Örnek: Faktöriyel Hesabı

n!= n.(n-1).(n-2). … . 3.2.1
= n . (n-1)!
(n-1)! = (n-1).(n-2). … 3.2.1
2! = 2.1!=2
1! = 1.0!=1
0! = 0.(-1)! ???

n! = { n.(n-1)! if n>1
1 o/w

```
int fact(int n)
{
    if (n <= 1)
        return n;
    return n * fact(n - 1);
}
```

- fact(0); fact(1); fact(2); fact(10);
- 0 1 2*1 10*9*8*7*6*5*4*3*2*1

Özyinelemeli Algoritmalar

- Örnek: Fibonacci Sayıları
- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …
- fib(n) = fib(n-1) + fib(n-2)
- fib(5) = fib(4) + fib(3) …
- fib(1) = fib(0) + fib(-1) ???

- fib(n) = { fib(n-1)+fib(n-2) if (n>=2)
- 1 o/w

```
int fib (int n)
{
   if (n <= 1)
      return 1;
   return fib(n-1) + fib(n-2);
}
```

- fib(2) = fib(1) + fib(0)
- fib(4) = fib(3) + fib(2)

Özyinelemeli Algoritmalarda Big-Oh

- n=0 T(0) = c1
- n=1 T(1) = c2
- n>1 ise T(n)=c1+c2+T(n-1)+T(n-2) ➔ **T(n-1)+T(n-2)+c**
- Yineleme ilişkisi denklemini çözmek gereklidir.

- T(n-1) ~ T(n-2) kabul edebiliriz
  - aslında T(n-1) ≥ T(n-2) ama üstten sınırlayabiliriz (upper bound)

Özyinelemeli Algoritmalarda Big-Oh

$T(n) = T(n-1) + T(n-2) + c$

$= 2T(n-1) + c$ // $T(n-1) \sim T(n-2)$ yaklaşımı ile

$= 2*(2T(n-2) + c) + c$

$= 4T(n-2) + 3c$

$= 8T(n-3) + 7c$

$= 2^k * T(n-k) + (2^k-1)*c$

$n-k=0$, **$k=n$**

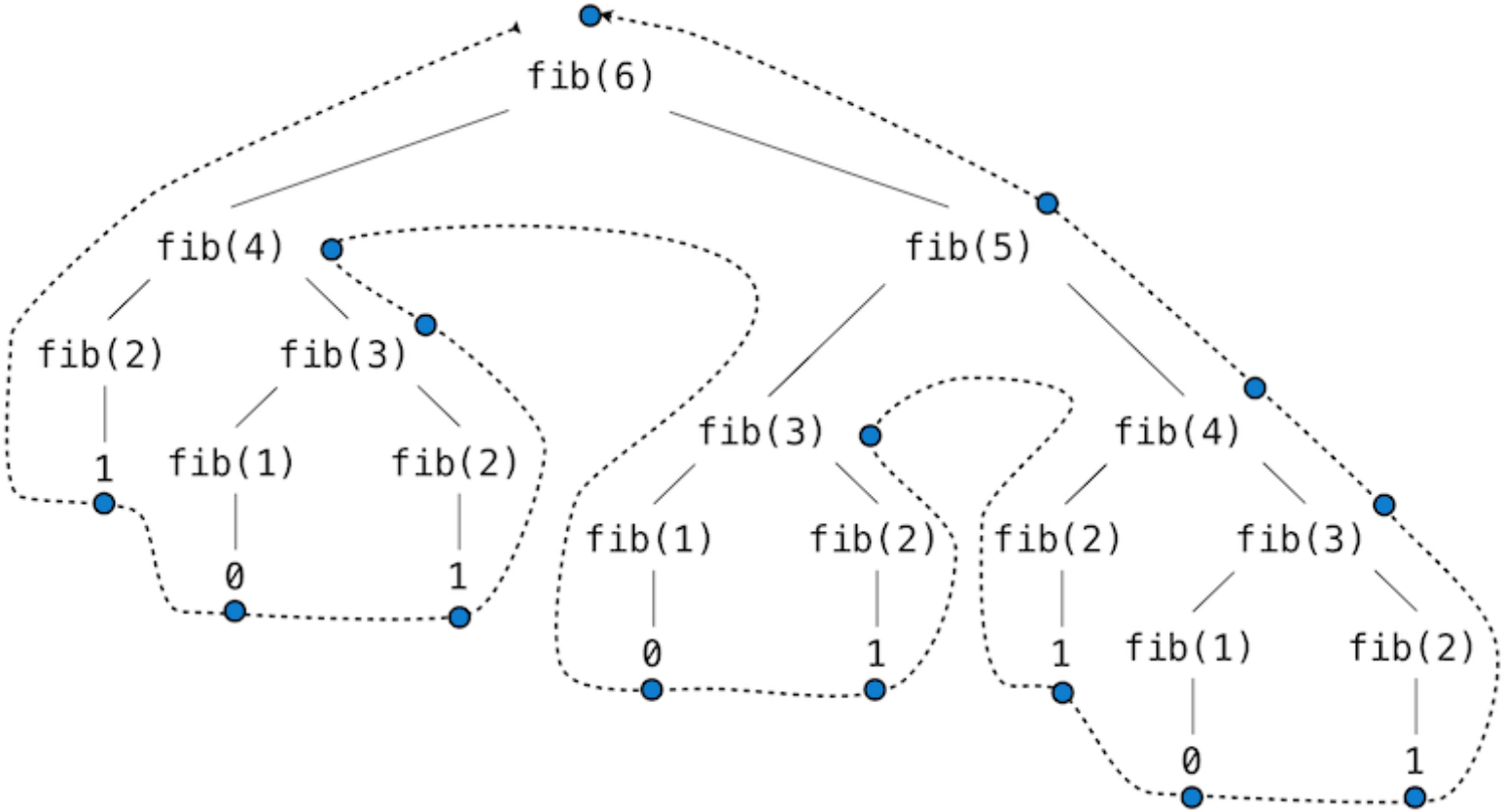
$T(n) = 2^n * T(0) + (2^n - 1) * c$

$= 2^n * (1 + c) - c$

$T(n) \sim 2^n$

- ➔ Algoritmanın büyüme oranı fonksiyonu **$O(2^n)$** olur.

Özyinelemeli Algoritmalarda Big-Oh



BÖL&YÖNET

Divide-and-Conquer

Böl&Yönet

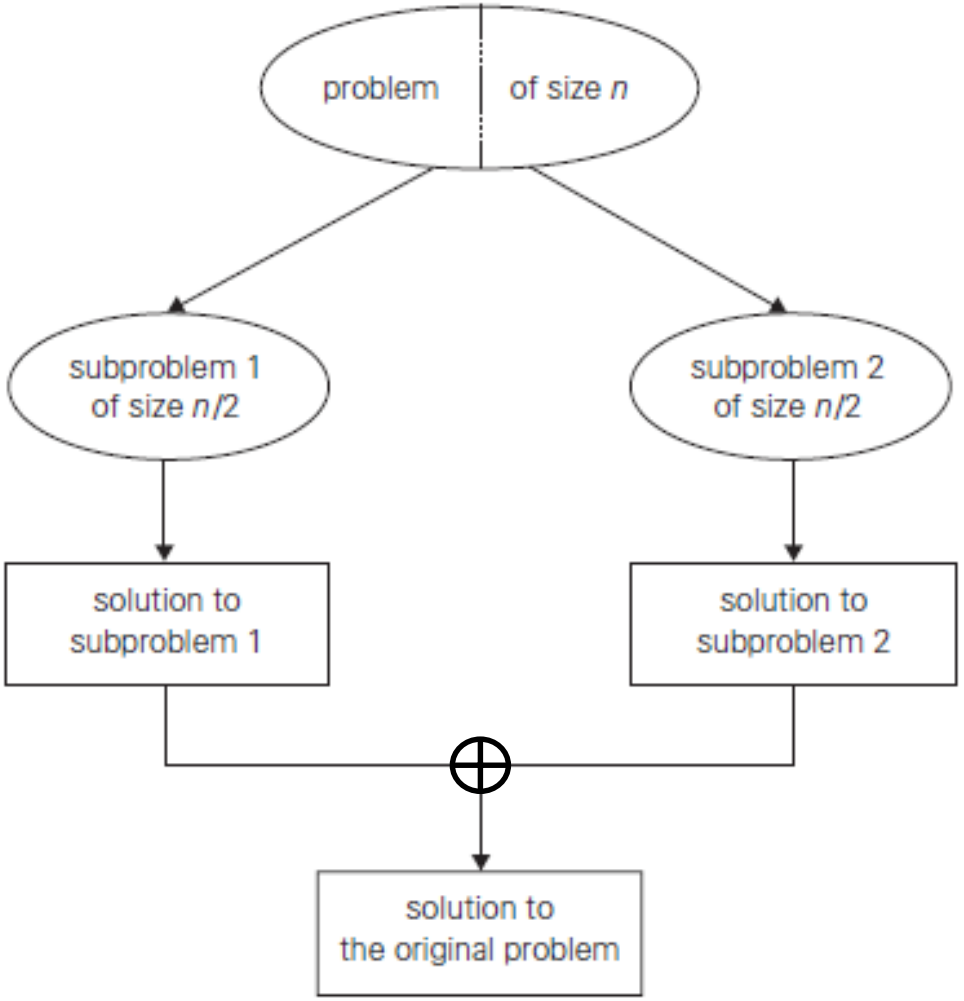
- Şimdiye kadar gördüğümüz algoritmik çözüm yöntemleri:
  - **Brute-Force:** Başlangıçtan dümdüz ilerleyerek çözüme doğru git.
    - Üs Alma a^n=a*a*….*a*a (n kere)
    - Selection Sort
    - Bubble Sort
  - **Greedy (Açgözlü):** Her adımda, o an için en iyi olan çözümü (local optimum) uygula
    - Huffman Encoding
  - **Decrease-and-Conquer:** Problemi küçülterek çözüme doğru ilerle.
    - Üs Alma $a^n=a^{n-1}*a$
    - Insertion Sort
    - Josephus Problem (Circular Linked List)

Böl&Yönet

- Problem (genellikle eşit boyutlu) aynı tipte küçük problemlere ayrıştırılır.
- Alt problemler çözülür.
  - Genellikle rekürsif olarak daha küçük problemler haline getirilir.
  - Bölünemeyecek kadar küçük problem bir yöntem ile çözülür.
- Gerektiğinde, alt problemlerin çözümleri bir araya getirilerek orijinal probleme ait nihai çözüm oluşturulur.

- Her zaman brute-force yaklaşımdan daha efektif çözüm olacak diye bir şart yoktur.
- Kolaylıkla *paralel hesaplama* şeklinde gerçekleştirilebilir.

Böl&Yönet


problem
of size n
subproblem 1
of size n/2
subproblem 2
of size n/2
solution to
subproblem 1
solution to
subproblem 2
solution to
the original problem

MERGESORT

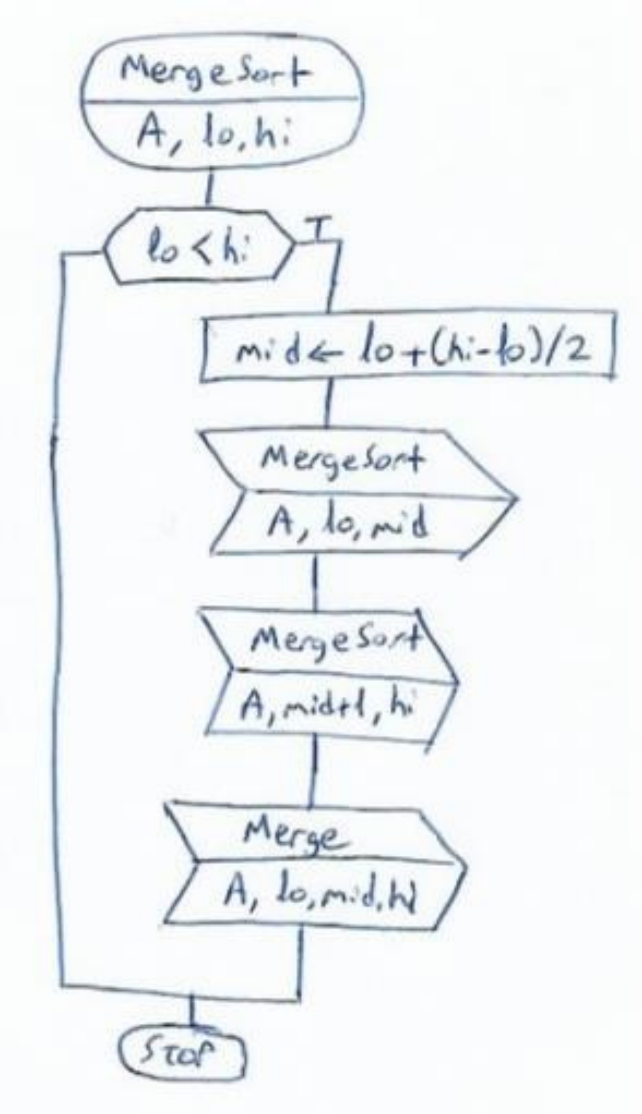
Merge Sort

- Elimizde A[0..n-1] dizisi var.
- Bu diziyi böl&yönet ile sıralayabilir miyiz?

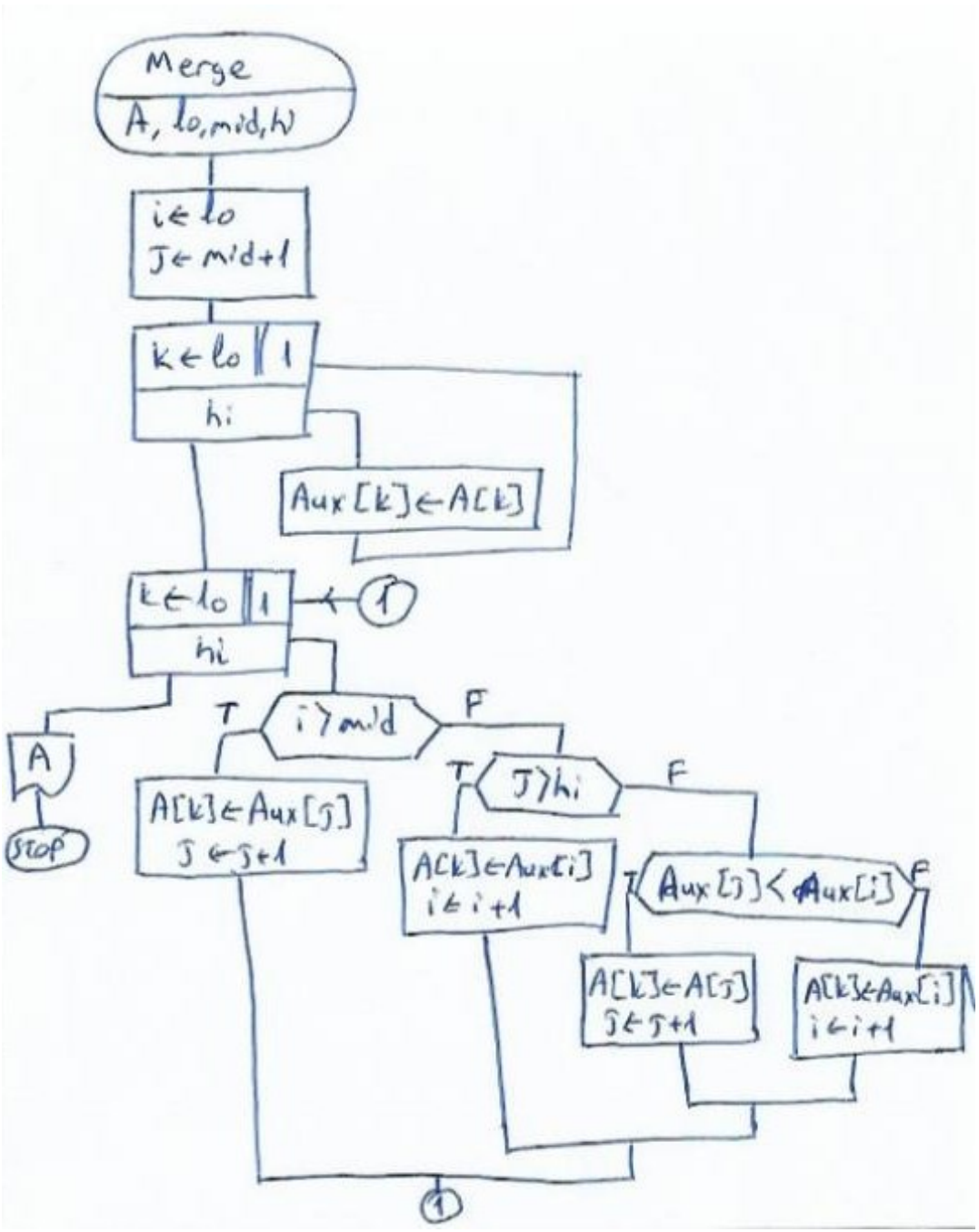
- Diziyi ikiye böl, her parçayı kendi içinde sırala.
  - Rekürsif olarak parçaları kendi içinde ikiye bölmeye devam et.
  - Algorithm MergeSort
- Tek elemanlık parçalar, zaten sıralı demektir.

- Sonra, her sıralı alt-alt parçayı sıralı olarak birleştir.
- Parçalar birleştikçe onları da sıralı olarak birleştir.
  - Algorithm Merge

Merge Sort (Top-Down MergeSort)



Merge


Merge
A, lo, mid, hi
i ← lo
j ← mid+1
k ← lo | 1
hi
Aux[k] ← A[k]
k ← lo | 1
hi
1
i > mid
T
F
A
stop
A[k] ← Aux[j]
j ← j+1
j > hi
T
F
A[k] ← Aux[i]
i ← i+1
Aux[j] < Aux[i]
T
F
A[k] ← A[j]
j ← j+1
A[k] ← Aux[i]
i ← i+1
1

Abstract in-place merge trace

		a[]										aux[]									
	k	0	1	2	3	4	5	6	7	8	9	i	j	0	1	2	3	4	5	6	7	8	9
Input		E	E	G	M	R	A	C	E	R	T			-	-	-	-	-	-	-	-	-	-
Copy		E	E	G	M	R	A	C	E	R	T			E	E	G	M	R	A	C	E	R	T
												0	5										
	0	A										0	6	E	E	G	M	R	A	C	E	R	T
	1	A	C									0	7	E	E	G	M	R		C	E	R	T
	2	A	C	E								1	7	E	E	G	M	R			E	R	T
	3	A	C	E	E							2	7		E	G	M	R			E	R	T
	4	A	C	E	E	E						2	8			G	M	R			E	R	T
	5	A	C	E	E	E	G					3	8			G	M	R				R	T
	6	A	C	E	E	E	G	M				4	8				M	R				R	T
	7	A	C	E	E	E	G	M	R			5	8					R				R	T
	8	A	C	E	E	E	G	M	R	R		5	9									R	T
	9	A	C	E	E	E	G	M	R	R	T	6	10										T
Merged		A	C	E	E	E	G	M	R	R	T												

input
copy
Abstract in-place merge trace
merged result

MergeSort

ALGORITHM *Mergesort(A*[0*..n* − 1]*)*

//Sorts array *A*[0*..n* − 1] by recursive mergesort

//Input: An array *A*[0*..n* − 1] of orderable elements

//Output: Array *A*[0*..n* − 1] sorted in nondecreasing order

if *n* > 1

  copy *A*[0*..n/*2 − 1] to *B*[0*..n/*2 − 1]

  copy *A*[*n/*2*..n* − 1] to *C*[0*..n/*2 − 1]

  Mergesort(B[0*..n/*2 − 1]*)*

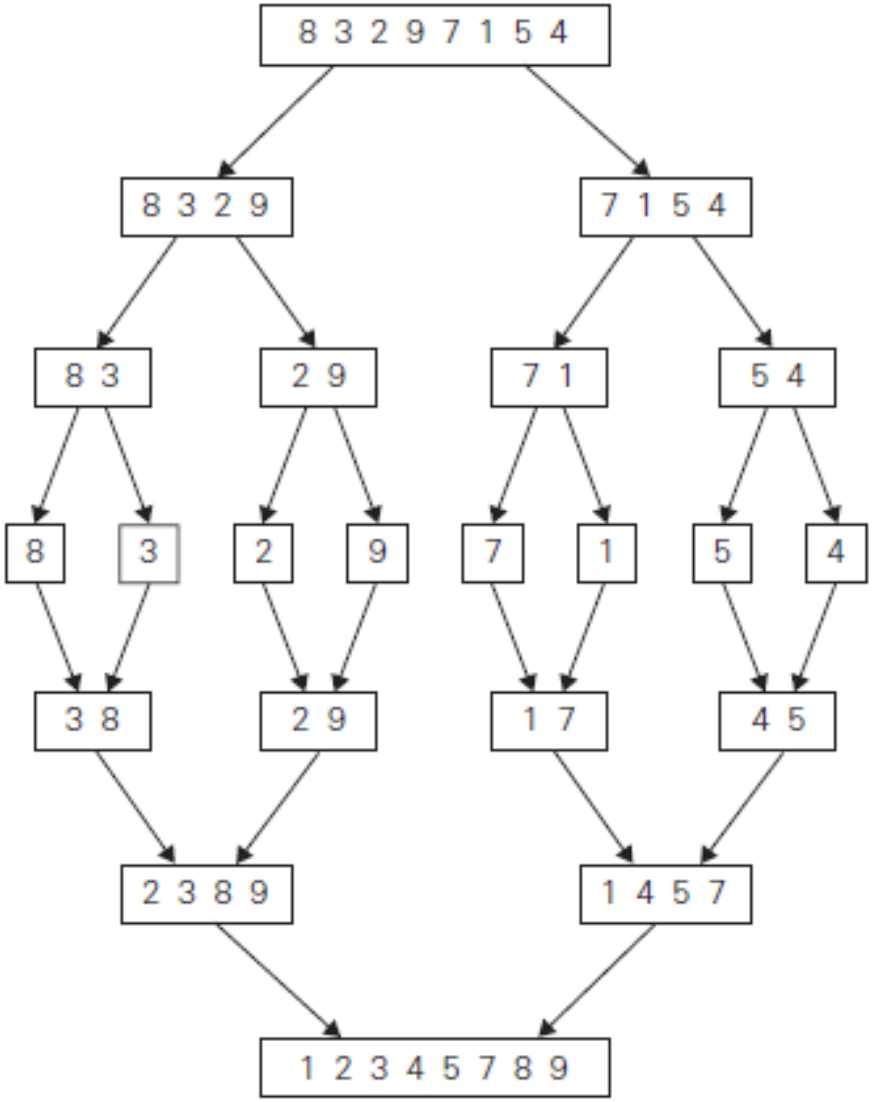
  Mergesort(C[0*..n/*2 − 1]*)*

  Merge(B, C, A)

Merge

ALGORITHM *Merge(B*[0..*p* − 1], *C*[0..*q* − 1], *A*[0..*p* + *q* − 1]*)*
//Merges two sorted arrays into one sorted array
//Input: Arrays *B*[0..*p* − 1] and *C*[0..*q* − 1] both sorted
//Output: Sorted array *A*[0..*p* + *q* − 1] of the elements of *B* and *C*
i ←0; *j* ←0; *k*←0
while *i* <*p* **and** *j* <*q* **do**
  if *B*[*i*]≤ *C*[*j*]
    A[*k*]←*B*[*i*]; *i* ←*i* + 1
  else
    A[*k*]←*C*[*j*]; *j* ←*j* + 1
  k←*k* + 1
if *i* = *p*
  copy *C*[*j*..*q* − 1] to *A*[*k*..*p* + *q* − 1]
else
  copy *B*[*i*..*p* − 1] to *A*[*k*..*p* + *q* − 1]

Merge Sort Örnek-1



MergeSort Örnek-2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
1	E	M														
2			G	R												
3	E	G	M	R												
4					E	S										
5							O	R								
6					E	O	R	S								
7	E	E	G	M	O	R	R	S								

MergeSort Örnek-2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
E	E	G	M	O	R	R	S								
8								E	T						
9										A	X				
10								A	E	T	X				
11												M	P		
12														E	L
13												E	L	M	P
14								A	E	E	L	M	P	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

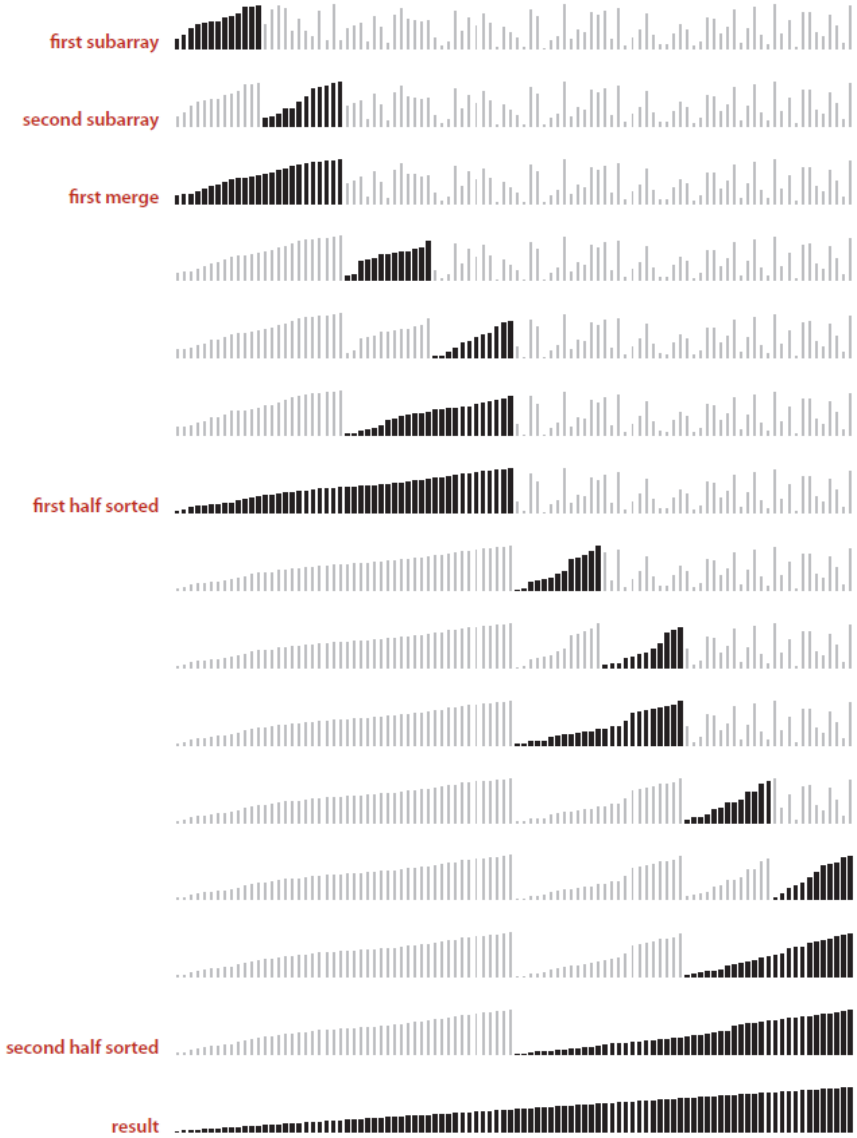
MergeSort Örnek-2 Analiz

- MergeSort(A, 0, 15)
  - MergeSort(A, 0, 7)
    - MergeSort(A, 0, 3)
      - MergeSort(A, 0, 1)
        - Merge(A, 0, 0, 1)
      - MergeSort(A, 2, 3)
        - Merge(A, 2, 2, 3)
      - Merge(A, 0, 1, 3)
    - MergeSort(A, 4, 7)
      - MergeSort(A, 4, 5)
        - Merge(A, 4, 4, 5)
      - MergeSort(A, 6, 7)
        - Merge(A, 6, 6, 7)
      - Merge(A, 4, 5, 7)
    - Merge(A, 0, 3, 7)
  - MergeSort(A, 8, 15)
    - MergeSort(A, 8, 11)
      - MergeSort(A, 8, 9)
        - Merge(A, 8, 8, 9)
      - MergeSort(A, 10, 11)
        - Merge(A, 10, 10, 11)
      - Merge(A, 8, 9, 11)
    - MergeSort(A, 12, 15)
      - MergeSort(A, 12, 13)
        - Merge(A, 12, 12, 13)
      - MergeSort(A, 14, 15)
        - Merge(A, 14, 14, 15)
      - Merge(A, 12, 13, 15)
    - Merge(A, 8, 11, 15)
  - Merge(A, 0, 7, 15)

MergeSort Karmaşıklık Analizi

- O(N lgN)
- A[0..15]
- A[0..7] A[8..15]
- A[0..3] A[4..7] A[8..11] A[12..15]
- A[0,1] A[2,3] . . . . . A[14,15]
- N Seviye, LogN adım
- k=0..n-1
- k.seviyede 2^k alt dizi, uzunlukları 2^{n-k}
- 2^{n-k} karşılaştırma
- 2^k . 2^{n-k} = 2^n işlem (her n için)
- $n.2^n$ ➔ O(N logN)
  - (6NlogN dizi erişimi: Her merge'de 2N kopya, 2N geri taşıma, 2N karşılaştırma)

Top-Down MergeSort Görselleştirme



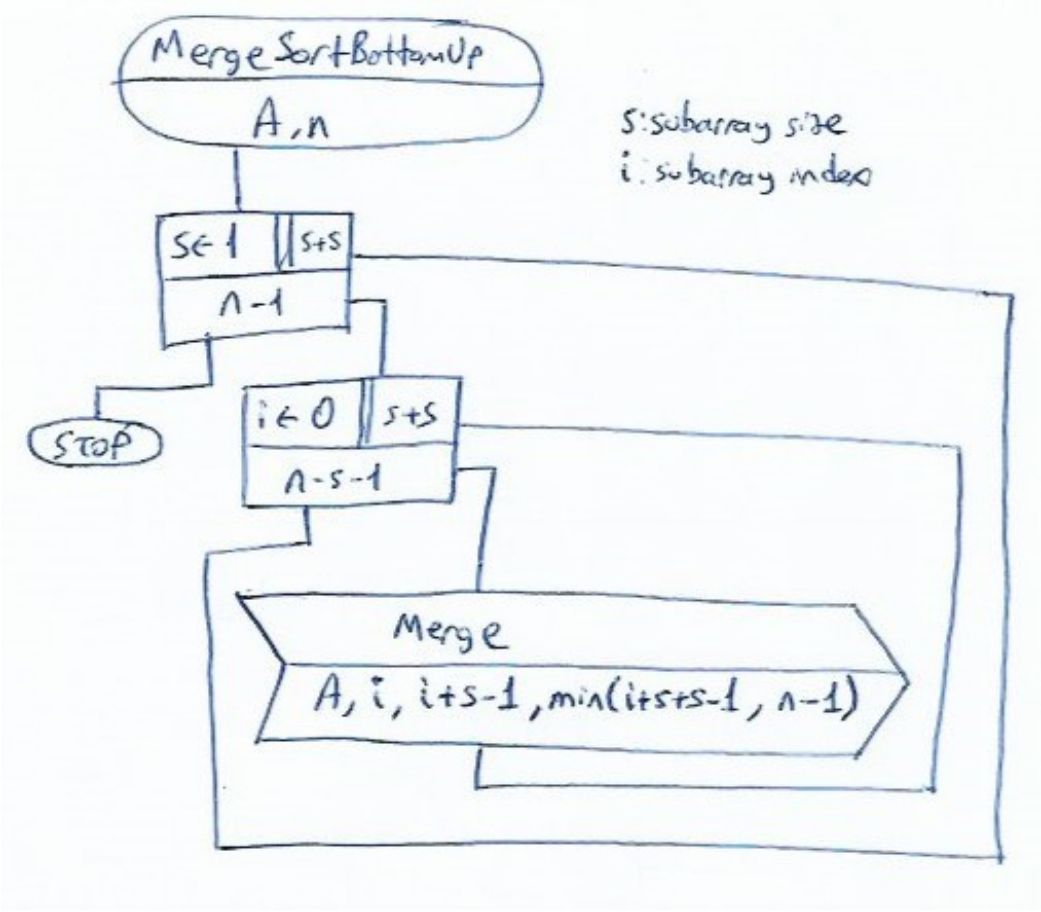
Bottom-Up MergeSort

```
void mergeSort(int a[], int lo, int r){
    int q;
    if(lo < r){
        q = (lo + r) / 2;
        mergeSort(a, lo, q);
        mergeSort(a, q+1, r);
        merge(a, lo, q, r);
    }
}

void merge(int a[], int lo, int q, int r){
    int b[5];   //same size of a[]
    int i=lo, j=q+1, k=0;

    while(i <= q && j <= r){
        if(a[i] < a[j]) b[k++] = a[i++];    // same as b[k]=a[i]; k++; i++;
        else b[k++] = a[j++];
    }
    while(i <= q) b[k++] = a[i++];
    while(j <= r) b[k++] = a[j++];

    for(i=r; i >= lo; i--)
       a[i] = b[--k];  // copying back the sorted list to a[]
}
```

```
int main(){
    int arr[] = {32, 45, 67, 2, 7};
    int len = sizeof(arr)/sizeof(arr[0]);

    mergeSort(arr, 0, len - 1);
    return 0;
}
```

Merge Sort (Bottom-Up MergeSort)

- 1-1, 2-2, 4-4, …N/2-N/2 birleştir
- 1/2NLgN – 6NlgN arası karşılaştırma, 6NlgN dizi erişimi



MergeSort Bottom-Up Örnek

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
1	E	M													S=1	
2			G	R												
3					E	S										
4							O	R								
5									E	T						
6											A	X				
7													M	P		
8															E	L

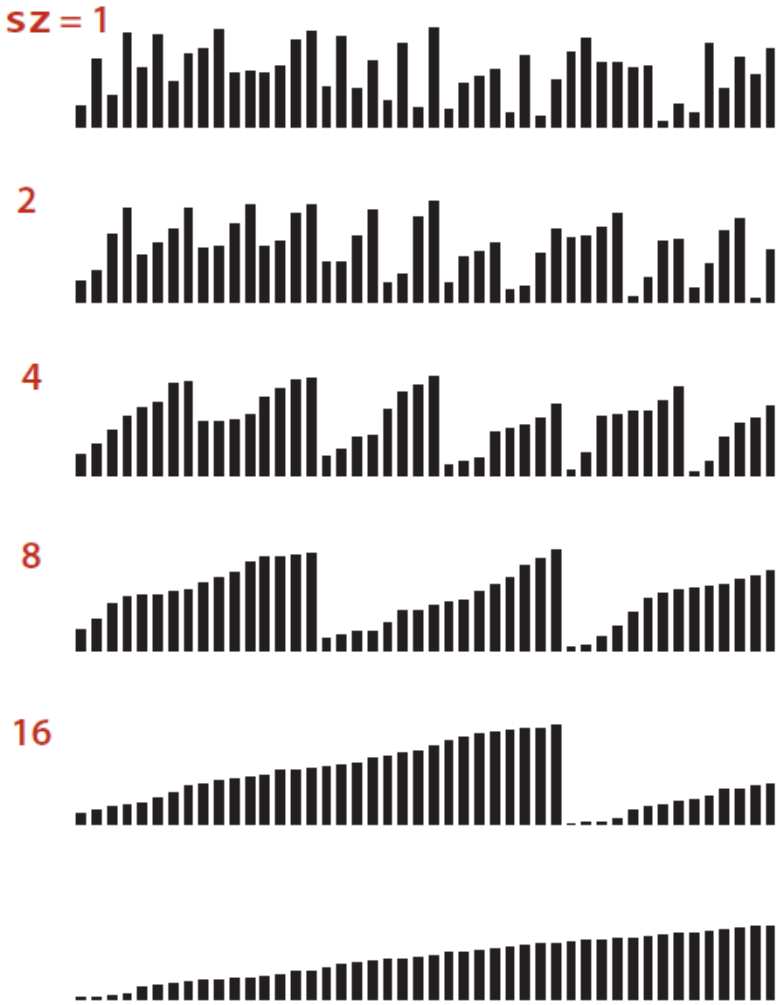
MergeSort Bottom-Up Örnek

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
9	E	G	M	R											S=2	
10					E	O	R	S								
11									A	E	T	X				
12													E	L	M	P
13	E	E	G	M	O	R	R	S							S=4	
14									A	E	E	L	M	P	T	X
															S=8	
15	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Bottom-Up MergeSort Analiz

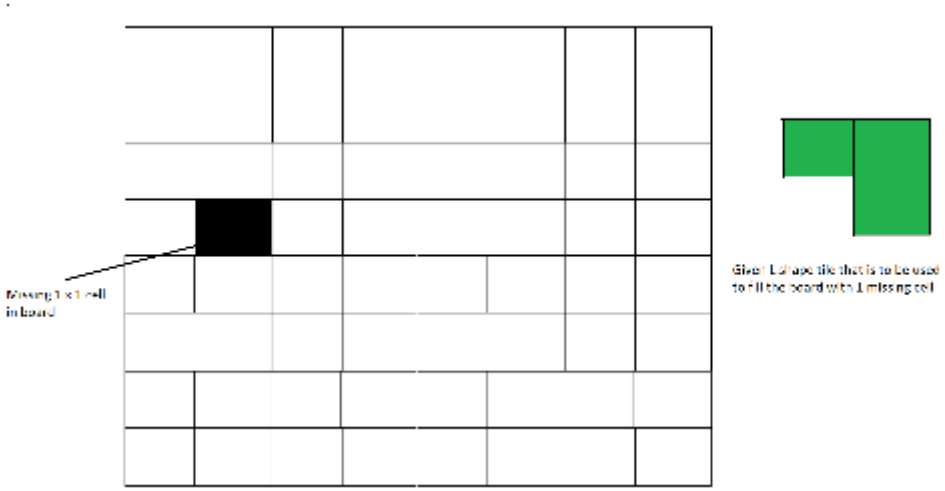
	a[i]															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
S=1																
merge(a, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 4, 4, 5)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 6, 6, 7)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 8, 8, 9)	E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
merge(a, 10, 10, 11)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, 12, 12, 13)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, 14, 14, 15)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
S=2																
merge(a, 0, 1, 3)	E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
merge(a, 4, 5, 7)	E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L
merge(a, 8, 9, 11)	E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L
merge(a, 12, 13, 15)	E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P
S=4																
merge(a, 0, 3, 7)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
S=8																
merge(a, 0, 7, 15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Bottom-Up MergeSort Görselleştirme

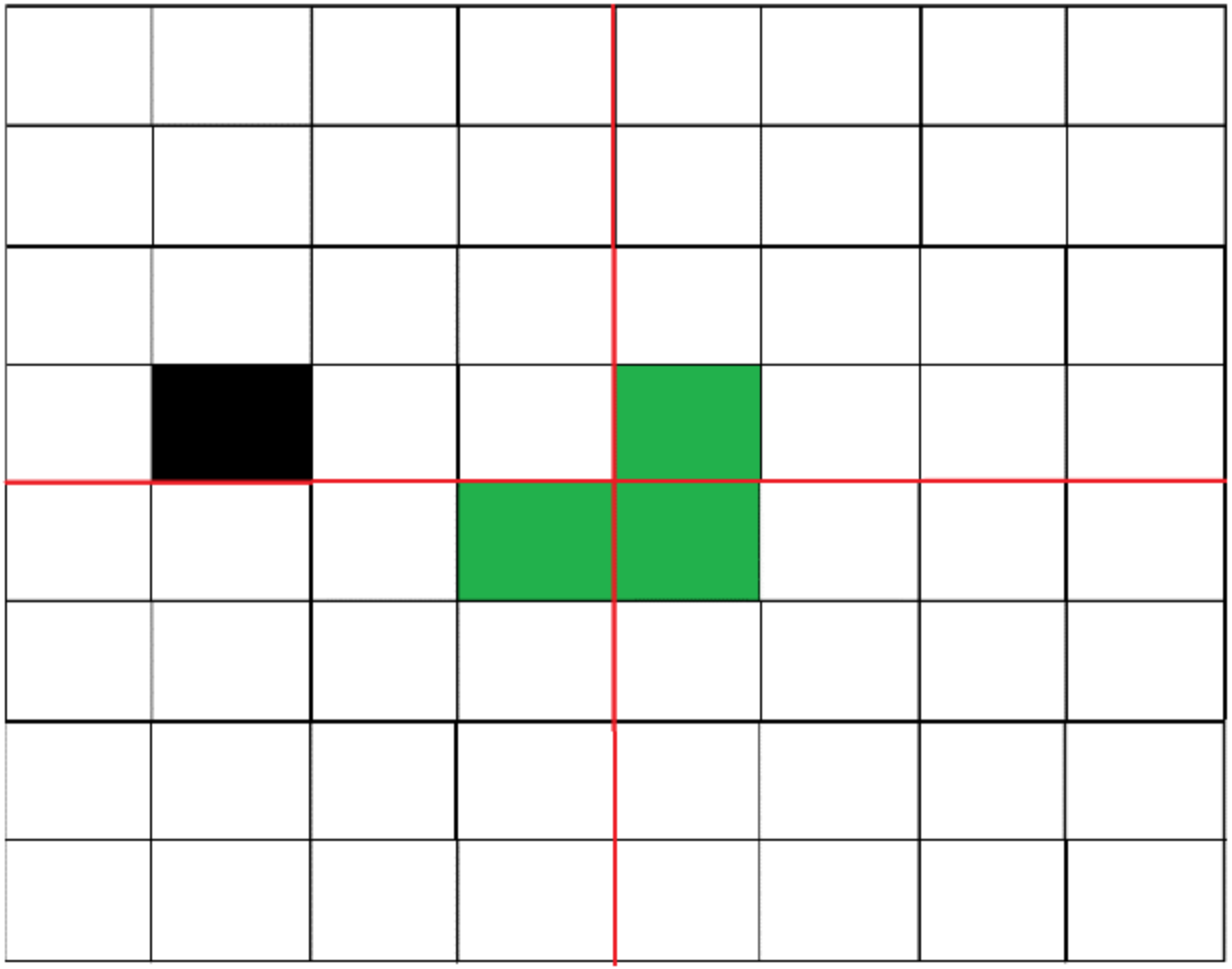


Böl&Yönet – Tromino Puzzle

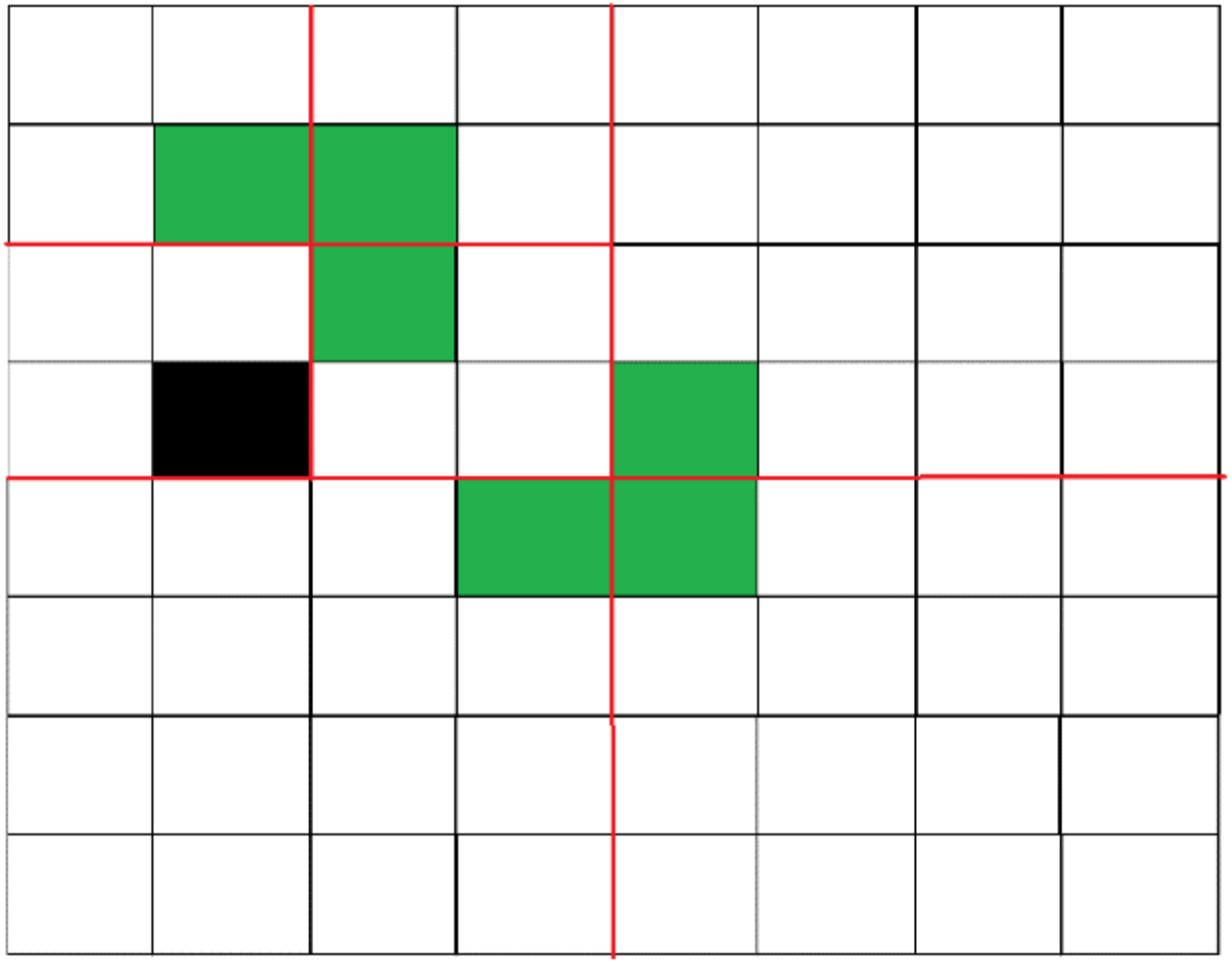
- Elimizde $2^n x 2^n$ boyutlarında bir kare pano var. Sadece tek bir gözü dolu.
- 3 tane 1x1 kareden oluşan (L şeklinde) karolarımız var.
- Karolar tüm yönlere döndürülerek yerleştirilebilir.
- Tüm panoyu karolar ile kaplayarak döşeyin.



Tromino Puzzle



Tromino Puzzle



Tromino Puzzle