

cr:merenyan

Runtime Analysis

	<u>Best Case</u>	<u>Average Case</u>	<u>Worst Case</u>	<u>(Space Comp) Worst Case</u>
Quick Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n^2)$	$O(\log n)$
Merge Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$	$O(n)$
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$

<u>Graph Operations</u>	<u>Remove Edge</u>	<u>Storage</u>	<u>Add Vertex</u>	<u>Add Edge</u>	<u>Remove Vertex</u>	<u>Query</u>
Adjacency List	$O(E)$	$O(V + E)$	$O(1)$	$O(1)$	$O(V + E)$	$O(V)$
Incidence List	$O(E)$	$O(V + E)$	$O(1)$	$O(1)$	$O(E)$	$O(E)$
Adjacency Matrix	$O(1)$	$O(V ^2)$	$O(V ^2)$	$O(1)$	$O(V ^2)$	$O(1)$

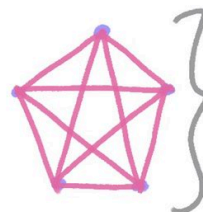
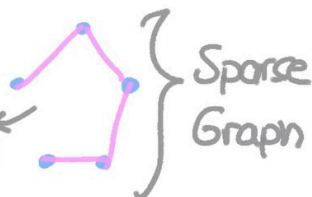
Average Case

Worst Case

<u>Data Structures</u>	<u>Access</u>	<u>Search</u>	<u>Insert</u>	<u>Delete</u>	<u>Access</u>	<u>Search</u>	<u>Insert</u>	<u>Delete</u>
Singly Linked List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Doubly Linked List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Binary Search Tree	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Notes:

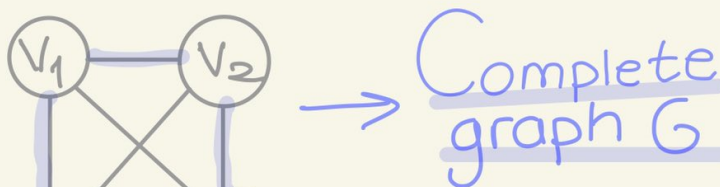
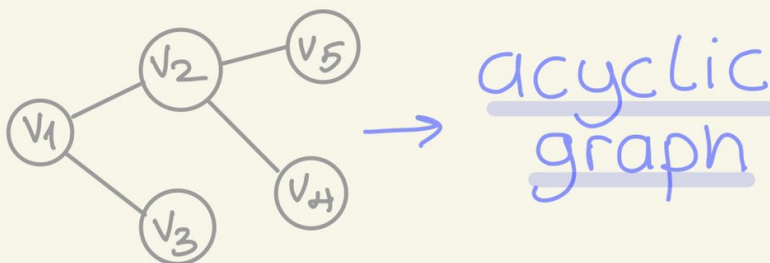
Use adjacency list to represent



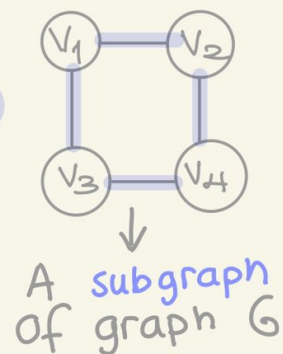
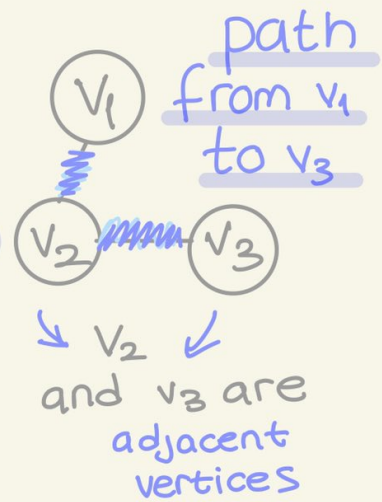
Use adjacency matrix to represent

Graphs

Given a graph $\rightarrow G = (V, E)$



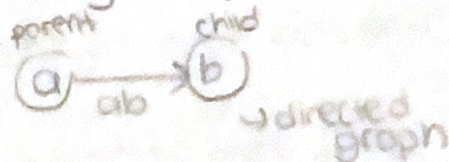
(There's a direct edge between each vertex)



Graph Representation

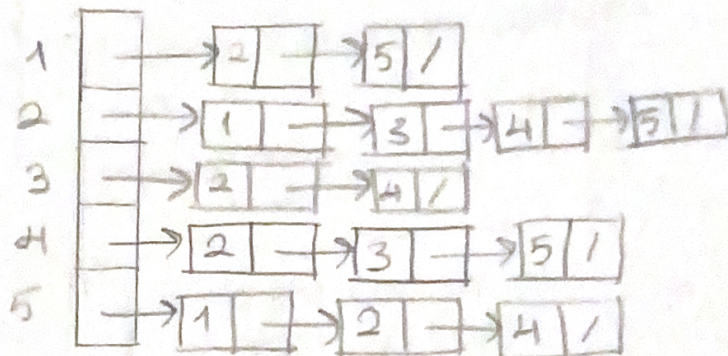
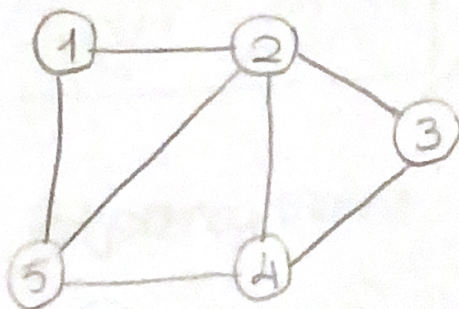
Given a graph $G=(V,E) \rightarrow$ express complexity using V & E

vertices $\swarrow$ edges $\searrow$



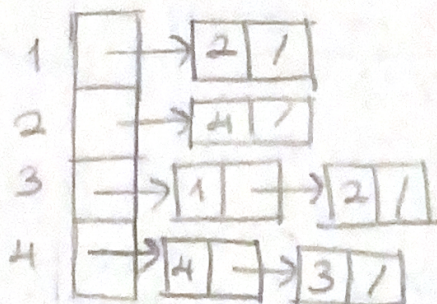
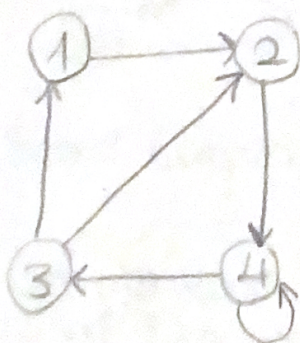
Adjacency List

For an undirected graph:



static data types (not ADT)

For a directed graph:



Space Requirement

$|V| \rightarrow$ size of V

$|E| \rightarrow$ size of E

array part $O(|V|)$ linked list part $O(|E|)$ } space comp.

Total complexity $\rightarrow O(|V|+|E|)$

Time Complexity

Assume you've taken linked list rep of adjacency list.
How do you draw graph from scratch?

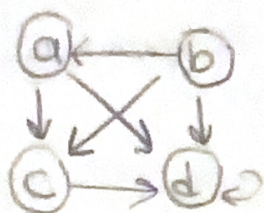
1) First go through array part & draw vertices $O(|V|)$

2) Draw edges, go over linked list $O(|E|)$

Total complexity: $O(|V|+|E|)$ (Directed graph)
(for drawing the graph)

Adjacency List (Contd.)

Time Complexity



$O(|V|)$ for directed graphs

Check if there's an edge between c & a.

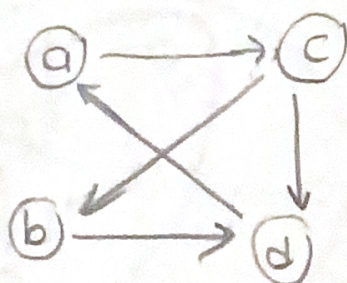
1) Go through the array part first. $O(1)$



2) Check if linked list contains a



Adjacency Matrix



there can be at most $|V|$ items. $O(|V|)$

Space Complexity = $O(|V|^2)$

Growth Rate (Time Comp) of Drawing the Graph = $O(|V|^2)$

	a	b	c	d
a	0	0	1	0
b	0	0	0	1
c	0	1	0	0
d	1	0	0	0

list was better

Checking the Edge = $O(1)$

matrix is better

★ But it's not wise to compare $|V|^2$ to $|V| + |E|$ we need to know how V relates to E.

Sparse Graphs

VS

Dense Graphs

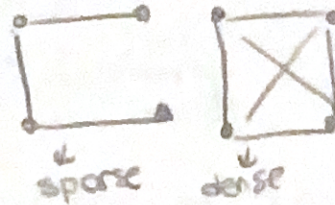
- Vertices & edges are linearly related
- $|V| \sim |E|$
- Number of edges can be at most $|V|-1$

- Edges are related to (vertices)²
 $|E| = \frac{|V|(|V|-1)}{2}$ $\rightarrow$ at most
 $|E| \sim O(|V|^2)$
 $\downarrow$
 worse case

2

Sparse vs Dense Graphs

- If graph is sparse, adjacency list rep is linear.
 - If graph is dense, it is same as adjacency matrix ($O(V^2 + |E|)$)
- use adj list if graph is sparse!



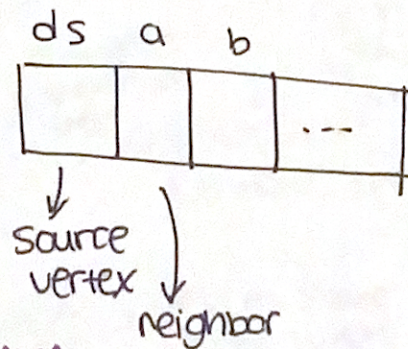
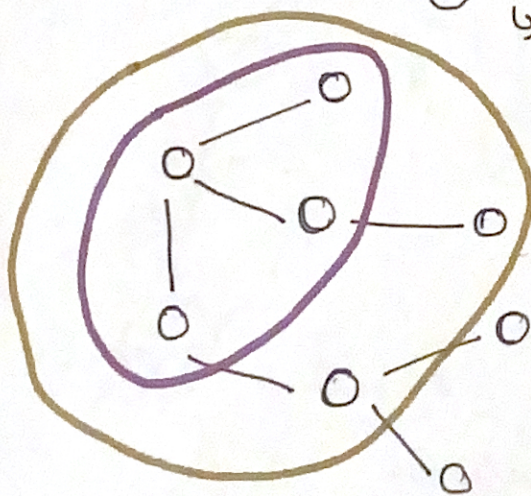
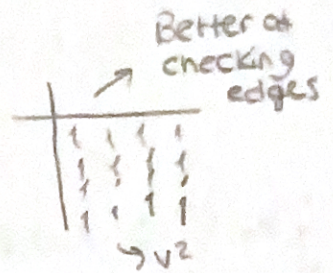
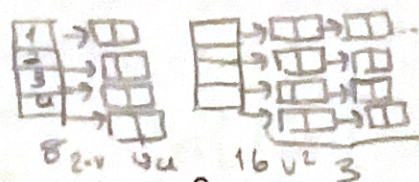
$|V|^2$

Graph Traversals

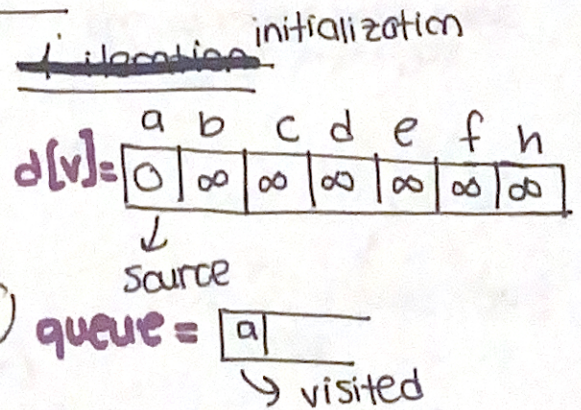
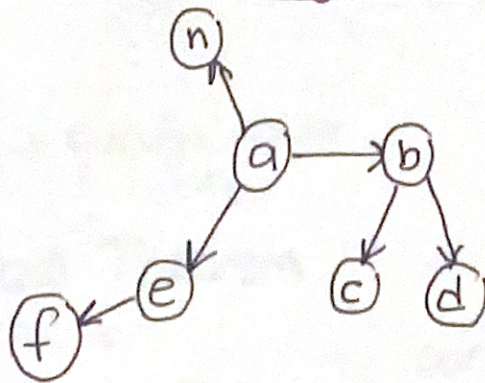
Breadth First Search

Input $\rightarrow G(V, E)$

Output $\rightarrow d[v]$ distance from s to v for all vertices of V
 $\rightarrow$ distance array



Algorithm



2. iteration

dequeue b.

check if b's neighbors = ∞

a	b	c	d	e	f	h
0	1	2	2	1	∞	1

$b+1$ $b+1$

1) iteration

check a's neighbors.

dequeue a.

add them if their distance is ∞

a	b	c	d	e	f	h
0	1	∞	∞	1	∞	1

[b, e, h]

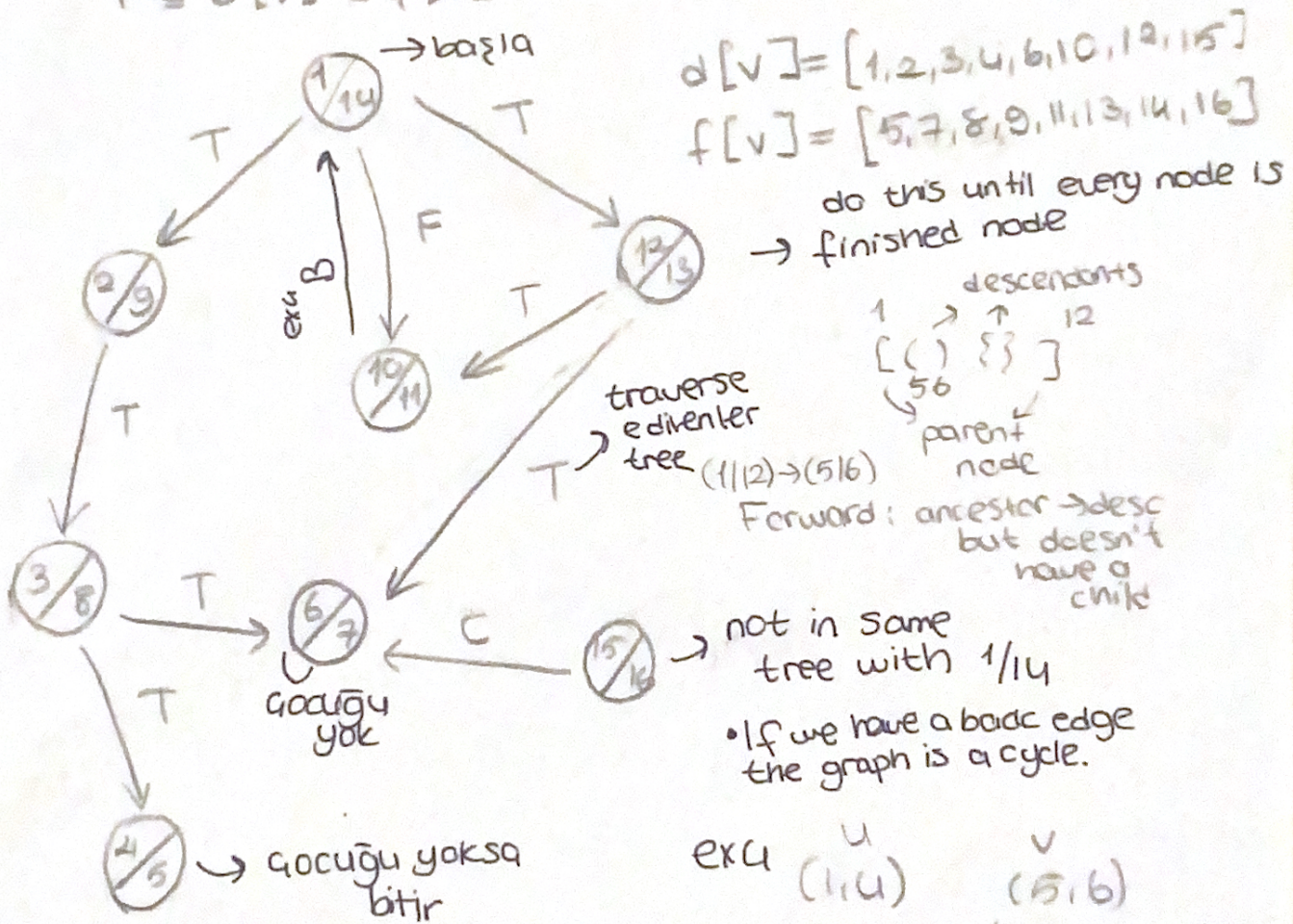
$\hookrightarrow$ enqueued (visited)

3. iteration $\rightarrow$ dequeue e, do the same

terminate when queue gets empty

Depth First Search

- Input: $G(V, E)$
- Output: $d[v]$ & $f[v]$
 - ↓ discovery times
 - ↘ finishing times
- Timestamps start at 1 & end at $2|V|$
- $1 \leq d[v] \leq f[v] \leq 2|V|$



Parenthesis Theorem

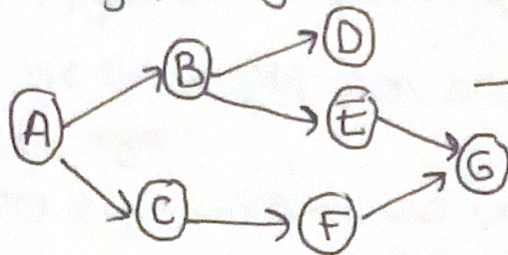
for u & v nodes, if:

- $d[u] < f[u] < d[v] < f[v] < d[u] < f[u] \rightarrow u \& v$ are not descendants of each other. (they're in different trees)
- $d[u] < d[v] < f[v] < f[u]$ (or vice versa) v is a child of u .
 u is a child of v .

ex: $u(1,1,2) \supset v(2,7)$ $d[u] = 1$ $d[v] = 2$ $f[v] = 7$ $f[u] = 12$
 $\hookrightarrow v$ is a child of u

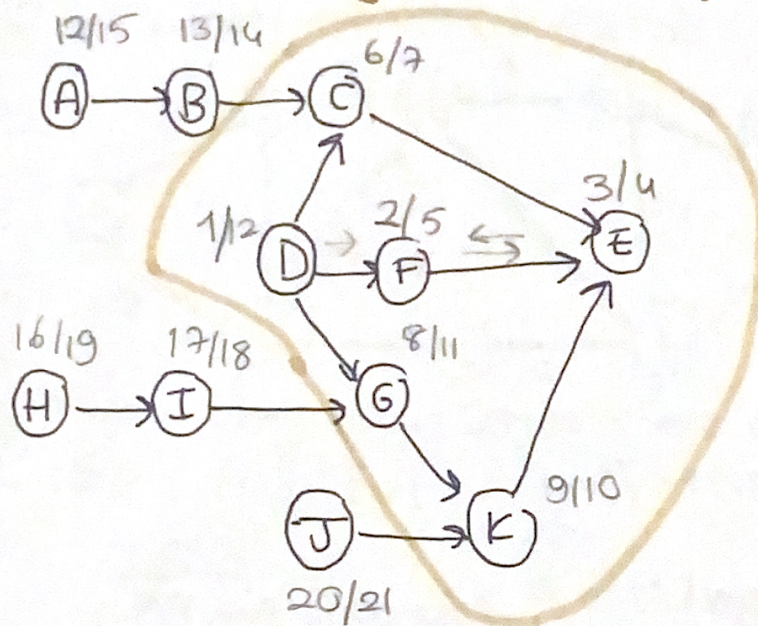
Directed Acyclic Graphs

- No back edge, no cycles



→ scheduling problems etc.

Topological Sorting (with DAGs)

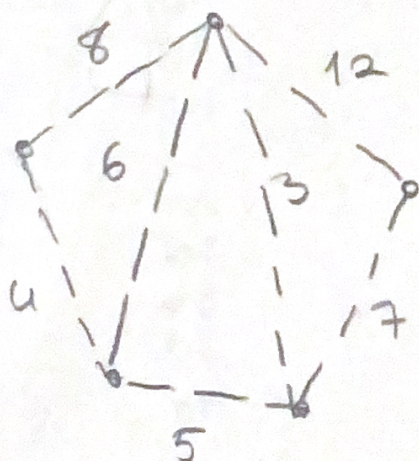


first tree

(my order)

A
B
H
I
J
D
C
F
K
E

Minimum Spanning Tree



→ Connect roads with min cost to connect all the houses

Input → $G(V, E)$
 weight $w(u, v)$
 ↓ ↓
 node node

T connects all the vertices (spanning tree)

$$\min w(T) = \sum w(u, v)$$

MST was:

- * no cycles
- * $|V| - 1$ edges (vertices - 1)
- * Might not be unique (another tree might have same value)

Minimum Spanning Tree

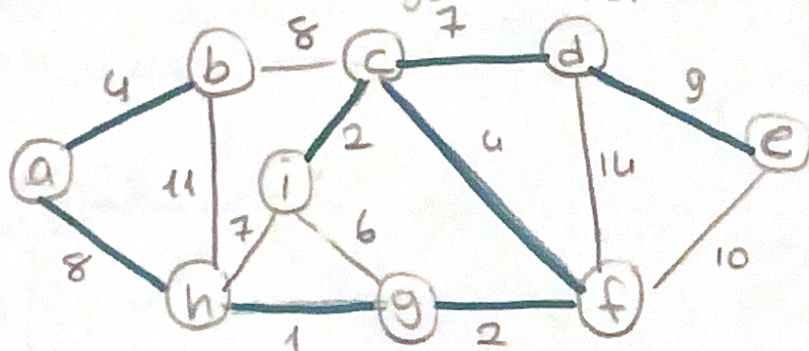
Kruskal's Algorithm ($\Theta(|E| \log |E|)$)

sparse graph $\rightarrow O(V \log V)$
 dense graph $\rightarrow O(V^2 \log V)$
 better than Prim
 worse than Prim

1. Sort all the edges from smallest to biggest in weights
2. Pick an edge
3. If the edge connects two vertices in disjoint subsets, merge the subsets, add the edge to F

If all subsets are merged $\rightarrow$ STOP

exa



1. Assume every single vertex is the subset of the graph
2. $S = \{\{a\}, \{b\}, \dots, \{n\}\} \rightarrow$ every node.

edges

hg

gf

ci

cf

ab

gi

cd

hi

ah

bc

dc

cf

bn

$\rightarrow$ Pick this

1st
it

$F = \{hg\}$

2nd
it

$F = \{hg, ci\}$

3rd
it

$F = \{hg, ci, gf\}$

subset

$S = \{\{a\}, \{b\}, \{i, c\}, \{d\}, \{h, g, f\}, \{e\}\}$

$\rightarrow$ subset

4th
it

$F = \{hg, ci, gf, ab\}$

$S = \{\{ab\}, \{i, c\}, \{d\}, \{h, g, f\}, \{e\}\}$

5th
it

$F = \{hg, ci, gf, ab, cf\}$

$S = \{\{ab\}, \{i, c, f, g, h\}, \{d\}, \{e\}\}$

6th
it

$F = \{hg, ci, gf, ab, cf, cd\}$

$S = \{\{ab\}, \{i, c, f, g, h, d\}, \{e\}\}$

7th
it

$F = \{hg, ci, gf, ab, cf, cd, ah\}$

$S = \{\{ab, c, f, g, h, d\}, \{e\}\}$

8th
it

$F = \{hg, ci, gf, ab, cf, cd, ah, de\}$

$S = \{a, b, c, d, e, f, g, h, i\}$

Minimum Spanning Tree (Contd)

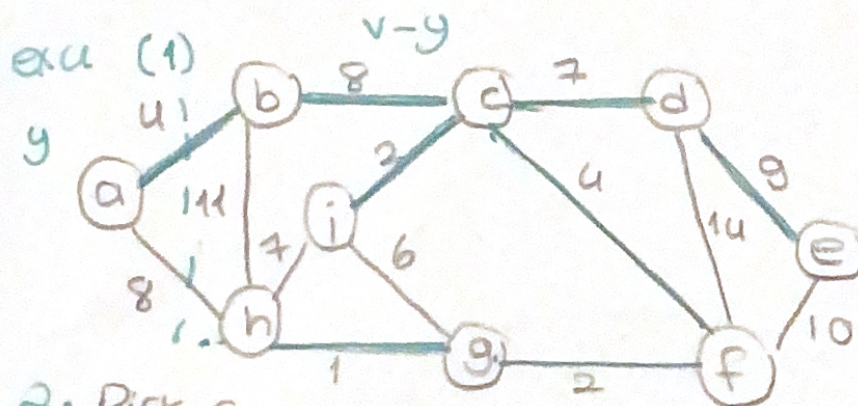
Greedy Choice $\rightarrow$ Can be based on selecting an edge or vertex at a time

Prim's Algorithm (vertex based) $\rightarrow \Theta |V|^2$

$F \rightarrow$ edge set (None at the beginning)

$Y = V_1 \rightarrow$ vertex set

1. Y 'ye en yakın ve Y 'de olmayan bir vertex seç.
2. Vertex'i Y 'ye edge'i F 'ye ekle.
3. $Y = V$ oluncaya kadar devam et.



First step:

$$F = \emptyset$$

$$Y = \{a\}$$

1. Pick b $\rightarrow$ shortest to a (Y) from ($V-Y$)

$$F = \{ab\}$$

$$Y = \{a, b\}$$

2. Pick c

(h 'ye de seçebiliriz, rastgele seçtik)

$$Y = \{a, b, c\}$$

$$F = \{ab, bc\}$$

3. Pick i $\leftarrow$ MST

$$F = \{ab, bc, ci\}$$

$$Y = \{a, b, c, i\}$$

4. Pick f

$$F = \{ab, bc, ci, cf\}$$

$$Y = \{a, b, c, i, f\}$$

5. Pick g

$$F = \{ab, bc, ci, cf, fg\}$$

6. Pick h

$$F = \{ab, bc, ci, cf, fg, gh\}$$

7. Pick d

$$F = \{ab, bc, ci, cf, fg, gh, cd\}$$

8. Pick e

$$F = \{ab, bc, ci, cf, fg, gh, cd, de\}$$

$$Y = \{a, b, c, d, e, f, g, h, i\}$$

Sorting

Items with $\leq$ relation

1) If we always keep order in unsorted list $\rightarrow$ Stable
otherwise $\rightarrow$ Unstable

2) Inplace / not inplace
if we need extra space (temp)

3) Internal / external
if data is larger you have to do partial sorting

4) Comparison based or Linear sort

$\downarrow$
 $\mathcal{O}(n \log n)$

$\rightarrow$ Bucket sort
Radix sort
Counting sort

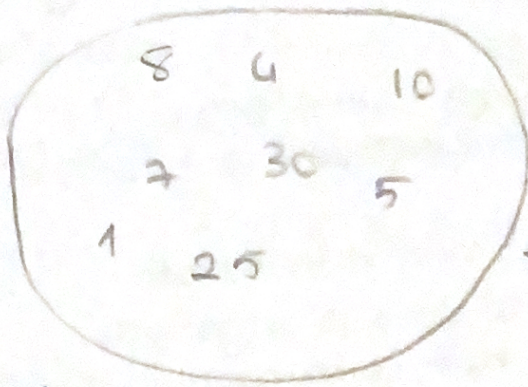
Selection $\rightarrow$ Brute force, # swaps is limited to $\mathcal{O}(n)$, in place
$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \in \Theta(n^2)$$

Bubble Sort $\rightarrow$ Brute force, in place

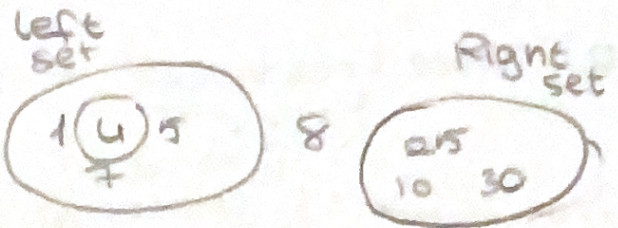
Insertion Sort $\rightarrow$ Brute force, in place

Merge sort > Divide & conquer, not in place
Quicksort

Quicksort

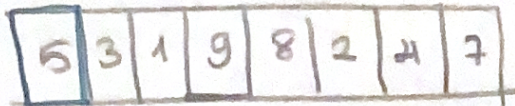


1. Select a pivot $v \rightarrow 8$
(Random selection)

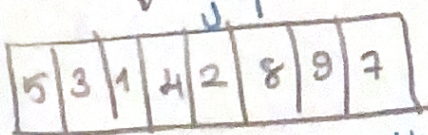
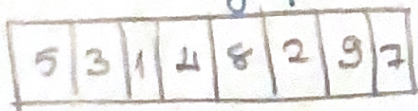
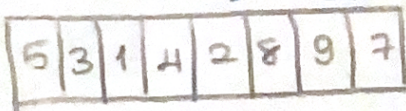
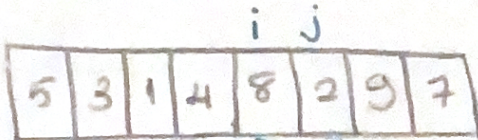


pivot

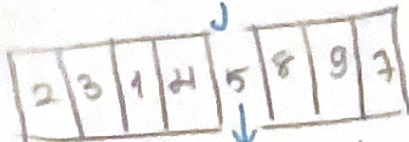
j stops here



i look for items ≤ 5
j stops here
look for items ≤ 5



↓ swap j with pivot (pivot heparatada kalsin)



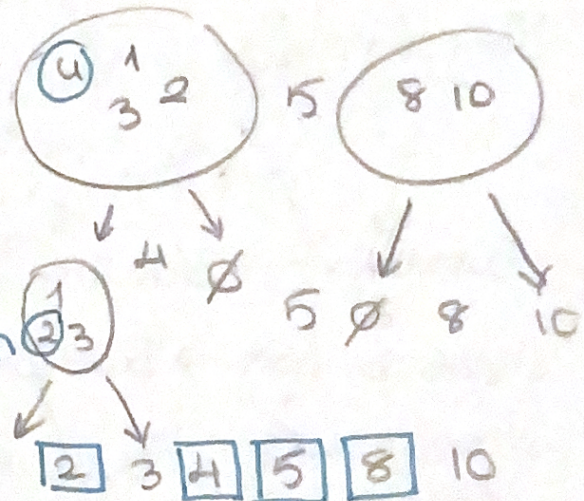
call quicksort for this

doesn't move again

call for this

Quicksort(A, p, r)

if $(p < r)$ {
 $q = \text{Partition}(A, p, r)$
 Quicksort(A, p, q-1)
 Quicksort(A, q+1, r)
}



Partition(A[L..r])

$P = A[L]$

$i = L$

$j = r+1$

do

do $i = i+1$ until $A[i] \geq P$

$j = j-1$ until $A[j] \leq P$

swap $(A[i], A[j])$

until $i \geq j$

swap $(A[i], A[j]) \rightarrow$ undo last swap

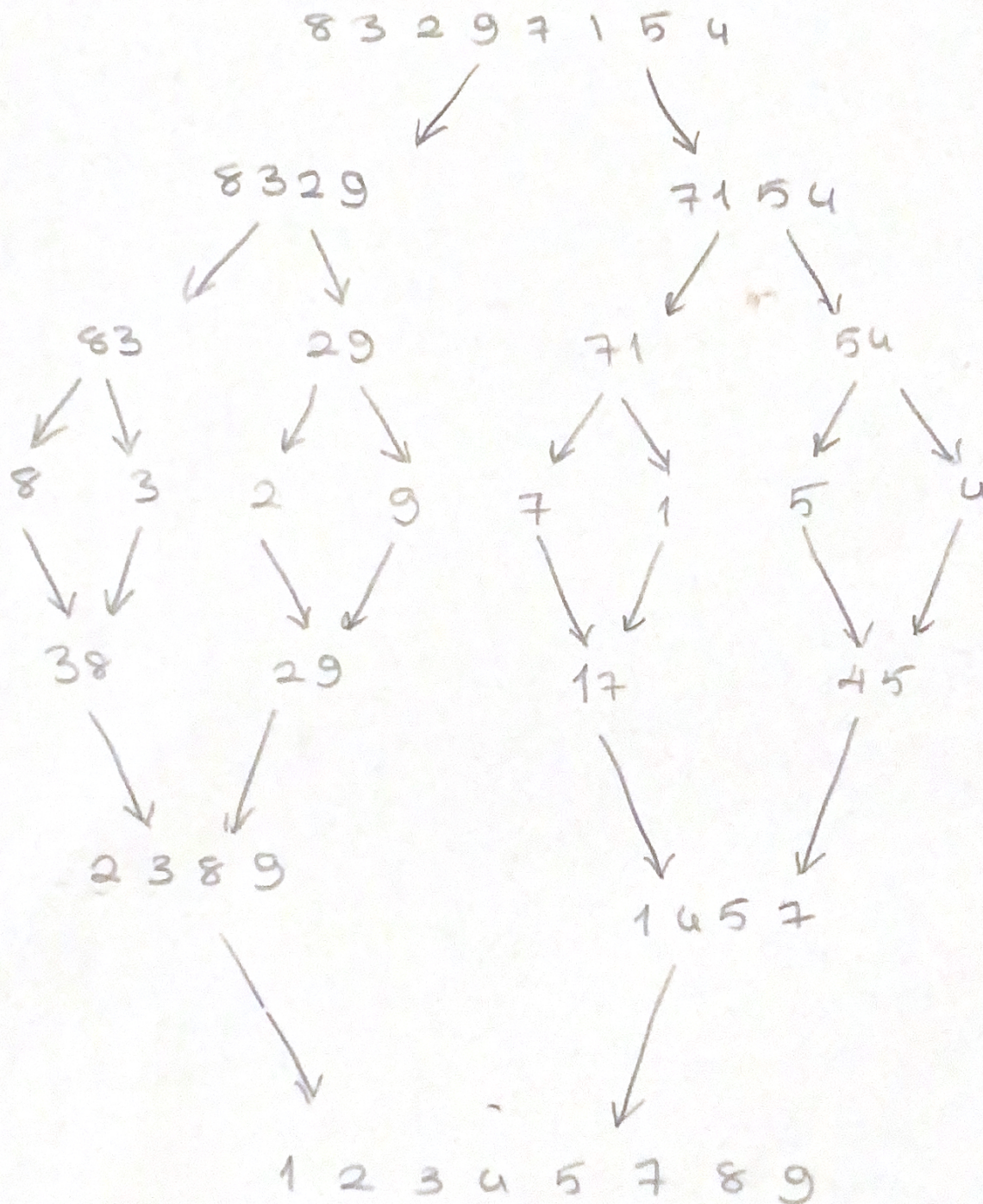
swap $(A[L], A[j])$

return j

pivot-on buyuk element bul

Merge Sort

- * Sorts array $A[0 \dots n-1]$ by recursive mergesort
- * Split array $A[0 \dots n-1]$ in two for equal halves (B & C)
- * Sort arrays B & C recursively



- * $\Theta(n \log n) \rightarrow$ no worst case
- * $\Theta(n) \rightarrow$ not in place

Bubble Sort

Repeatedly swap adjacent elements if they are in wrong order.

1st pass: $\underline{5} 1 4 2 8 \rightarrow 1 \underline{5} 4 2 8 \rightarrow 1 4 \underline{5} 2 8 \rightarrow 1 4 2 \underline{5} 8$

2nd pass: $\underline{1} 4 2 5 8 \rightarrow 1 \underline{4} 2 5 8 \rightarrow 1 2 \underline{4} 5 8 \rightarrow 1 2 4 \underline{5} 8$

3rd pass: A pass with no swap ends the algorithm.

Insertion Sort

Sorting playing cards.

- 1) Iterate over first element til last in array. Do:
- 2) Compare current element to its predecessor, if it is, compare it to elements before. move greater elements one position up to make space for the swapped element.

4 3 2 10 12 1 5

1: $\underline{4} 3 2 10 12 1 5$

2: $\underline{3} 4 2 10 12 1 5$

3: 2 3 4 10 12 1 5 (Stays same for 2 iterations)

4: $\underline{2} 3 4 10 12 1 5$

5: 1 2 3 4 10 12 5

6: 1 2 3 4 5 10 12

Selection Sort

Scan the array to find its smallest element and swap it with first element.

On pass i ($0 \leq i \leq n-2$) find the smallest element $A[i..n-1]$ and swap it with $A[i]$.

7 3 2 6 1 9 5

1 3 2 6 7 9 5

1 2 3 6 7 9 5

1 2 3 5 6 7 9

for $i = 0$ to $n-2$ do

$min = i$

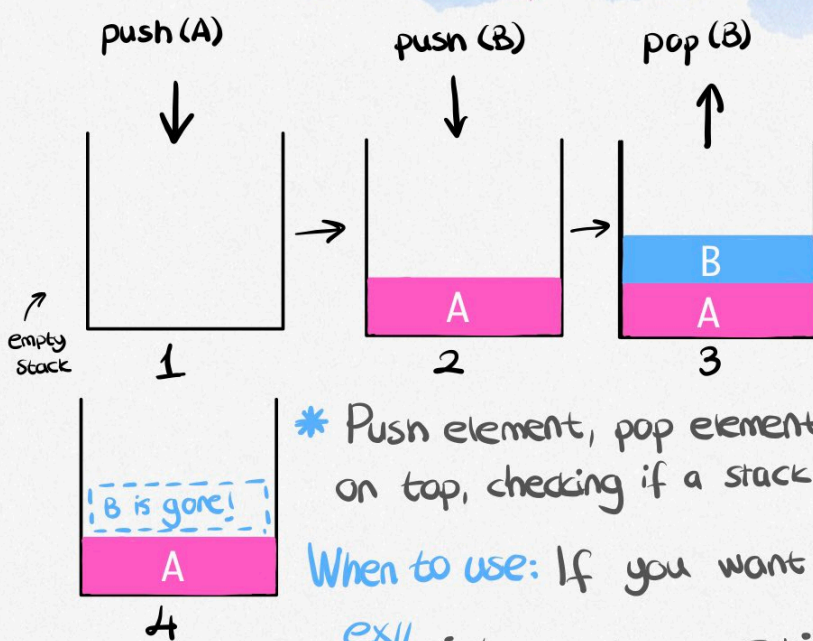
 for $j = i+1$ to $n-1$ do

 if $A[j] < A[min]$

$min = j$

 swap $A[i]$ and $A[min]$

Stacks



* First in, last out

• **Push (A):** To add an element(A) on top, push.

• **Pop (A):** To remove an element from the top, pop

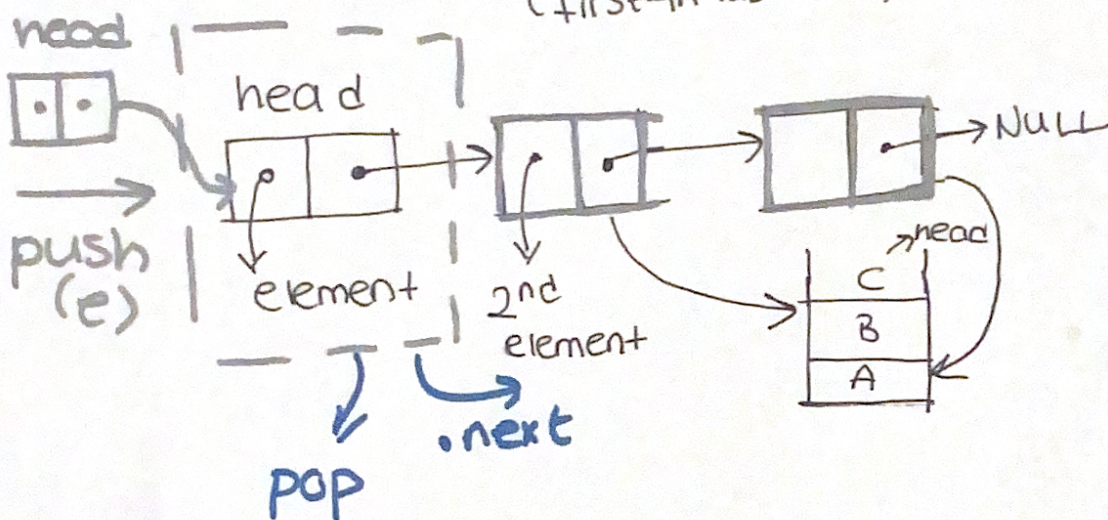
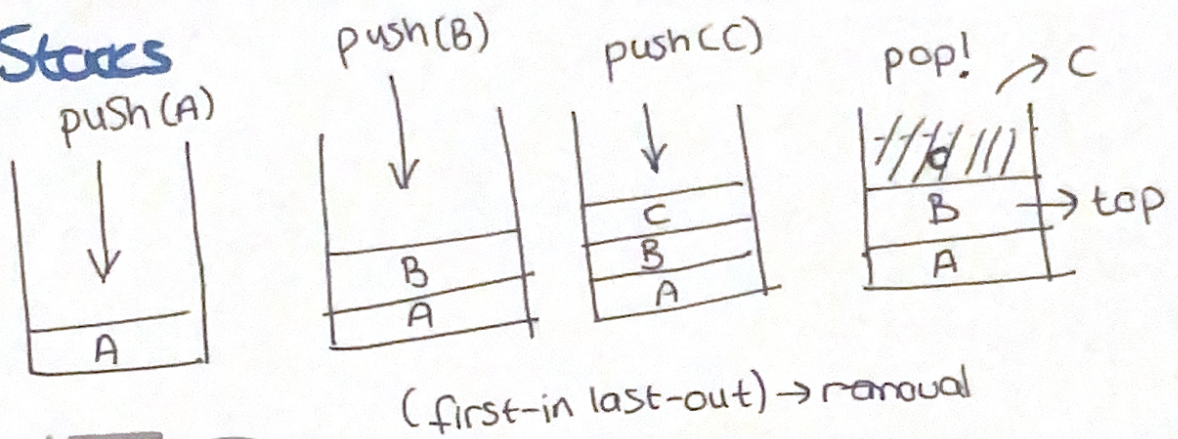
* Push element, pop element, returning the element on top, checking if a stack is empty, all runs in $O(1)$

When to use: If you want to return the last element

ex||

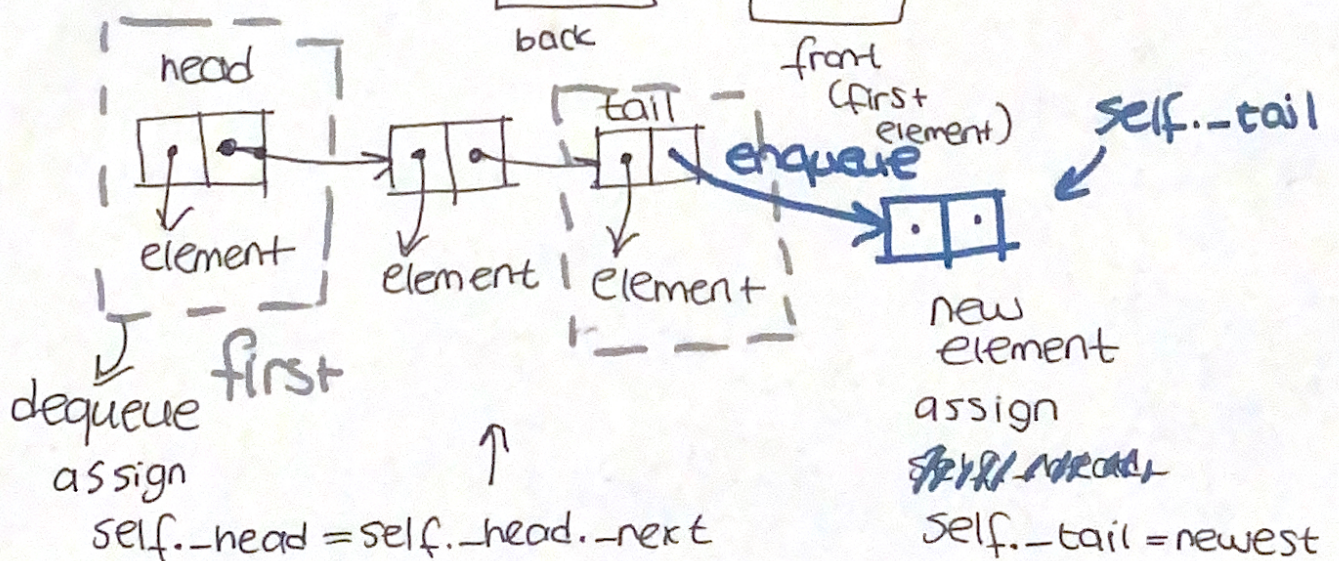
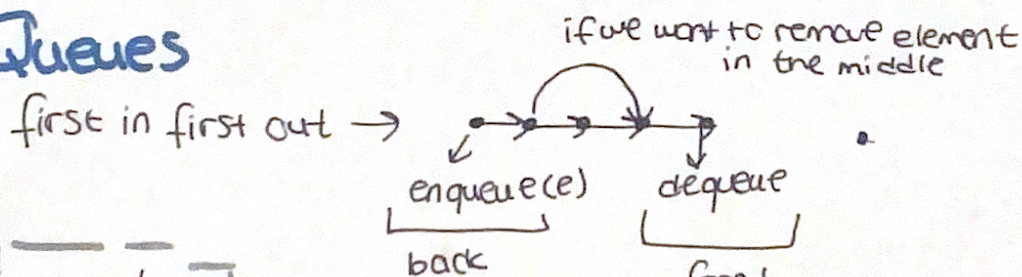
internet browser history,
infix to postfix, parentheses matching in
code editors.

Stacks



All operations (push, pop, is-empty, top) runs in $O(1)$ time.

Queues



Infix to Postfix

$$\left(\left((A/B)^{**}C \right) + (D^*E) - (A+C) \right) \rightarrow \text{In-fix}$$

$\downarrow \quad \downarrow \quad \downarrow$
 Operands Operators

In-fix $\rightarrow$ Operand Operator Operand
 $A \quad + \quad B$

Post-fix $\rightarrow$ Operand Operand Operator
 $A \quad B \quad +$

1. Parantezleri yaz
2. Operator'leri içinde bulundukları ifadenin sağ parantezine yaz.
- En içteki parantezlere uygula, dışarı doğru git.
3. Parantezleri sil

Rules

$+, -, /, *$ left to right $\rightarrow (A+B)+C \rightarrow AB+C+$

$+ = - , * = / , (* /) > (+ -) \rightarrow \text{precedence}$

examples

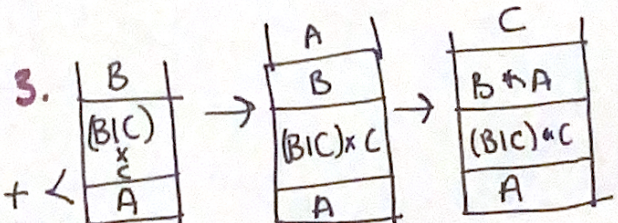
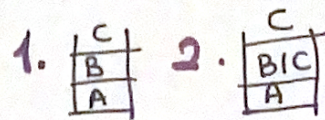
$$(A + (B * C)) + D \rightarrow \underline{A B C * +} D +$$

$$A - ((B+C) * D) \rightarrow A B C + D * -$$

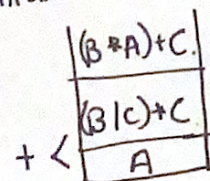
$$A + ((B/C) * C) + (B * A) + C \rightarrow A B C / C * \uparrow B A * C + +$$

To validate:

1. Put operand to stack (Iterate over the expression)
2. If you encounter an operator, pop last two elements, apply operation.
3. Keep going until no element is left. (Eleman kaldysa işaret unutmuşsunur!)



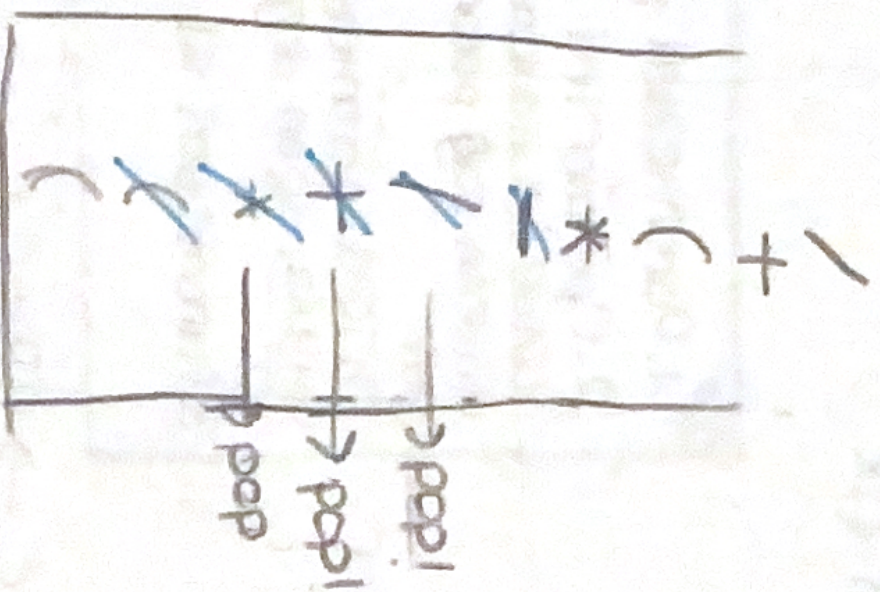
$+ <$
unuttum



geri
denmek
için

Infix to Postfix

$$((A * (B + D) / E) - F * (G + H / K)))$$

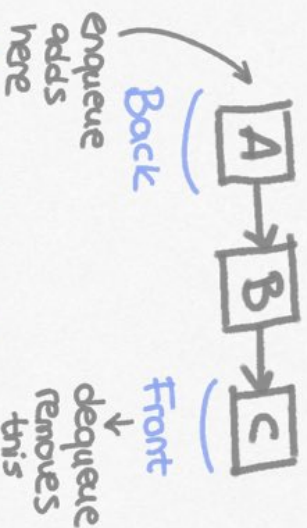


1. If operand $\rightarrow$ write
2. If operator $\rightarrow$ push
3. If operator $>$ op on stack, push
4. If op on stack $>$ next operator, pop all operators
5. If (, push on stack
6. If), pop all of them until (shows up

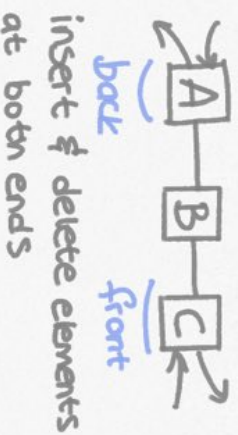
$$ABD + E / * \pi G I K / + * -$$

Queues

First in, first out



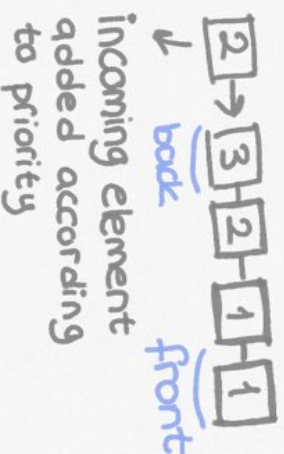
Double Ended Queue



enqueue(e): add element to the
front $\nearrow$ back of the queue

dequeue: remove the element in the
front of the queue

Priority Queue

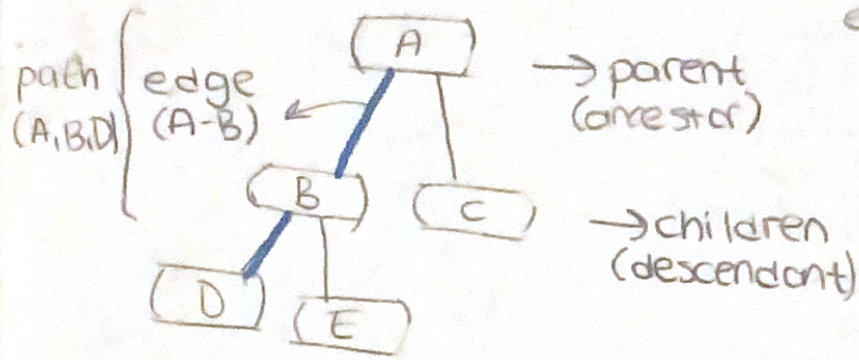


Time complexity

enqueue
dequeue
get front
get back } $O(1)$

Search $\rightarrow O(n)$

Trees



ordered tree
(Göçüklerini 1. 2. 3. diye ayırt edebiliyoruz)

Abstract Data Type

$p.\text{element}()$ → returns the element stored at position p .

$T.\text{root}()$ → root of tree

$T.\text{is-root}(p)$ → if p is the root

$T.\text{parent}(p)$ → parent of p

$T.\text{children}(p)$ → children of p

$T.\text{is-leaf}(p)$ → True if p is leaf

→ bunlar Tree ABC'sinin abstract method'ları.

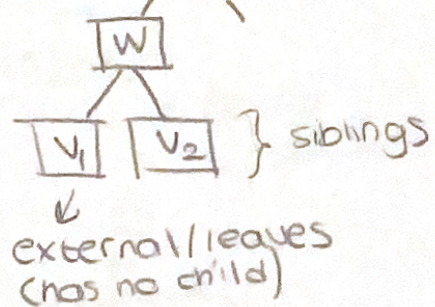
Yeni tree subclass'ı oluştururken override ediyoruz.

tree T : set of nodes storing elements s.t.

- nodes have parent-child relationships

- If T is nonempty, it has a root node (has no parent)

- Every node v of T different from the root has a unique parent w , every node with a parent w is a child of w .



Depth of p : p 'den önceki node'ların sayısı, p hariç. root'un depth'i 0.

$\text{depth } p = 1 + \text{depth}_{\text{parent of } p}$

$O(dp + 1)$, performs

constant time recursive step for each ancestor of p .

(worst case'i $O(n)$)

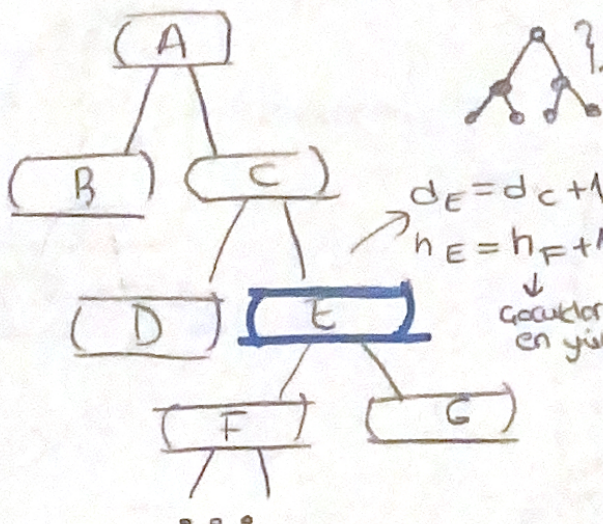
Height of p : maximum of depths of its leaf positions. (taşı katlı?)

T ağacının height'i root'un height'ine eşit.

→ p leaf node's a height'i

0 değilse ($n_{\text{children}} + 1$)

→ worst case $O(n)$ → recursive



Complexity of Linked Binary Tree

len	left	add-root	add-left	attach	} $O(1)$
is-empty	right	is-root	add-right		
root	sibling	is-leaf	replace		
parent	children	num-children	delete		

depth(p) $\rightarrow O(\text{dp} + 1)$

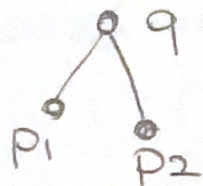
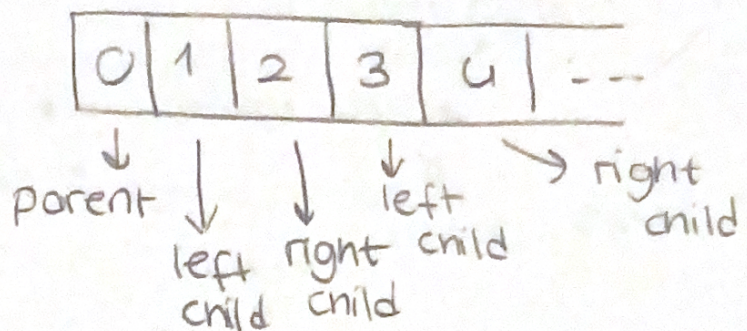
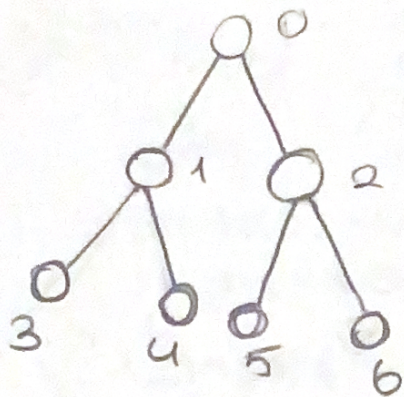
height $\rightarrow O(n)$

$\rightarrow$ space usage : $O(n)$

Complexity of Linked General Tree

Same with children(p) = $O(\text{cp} + 1)$
 space usage = $O(n)$

Array Based Binary Tree



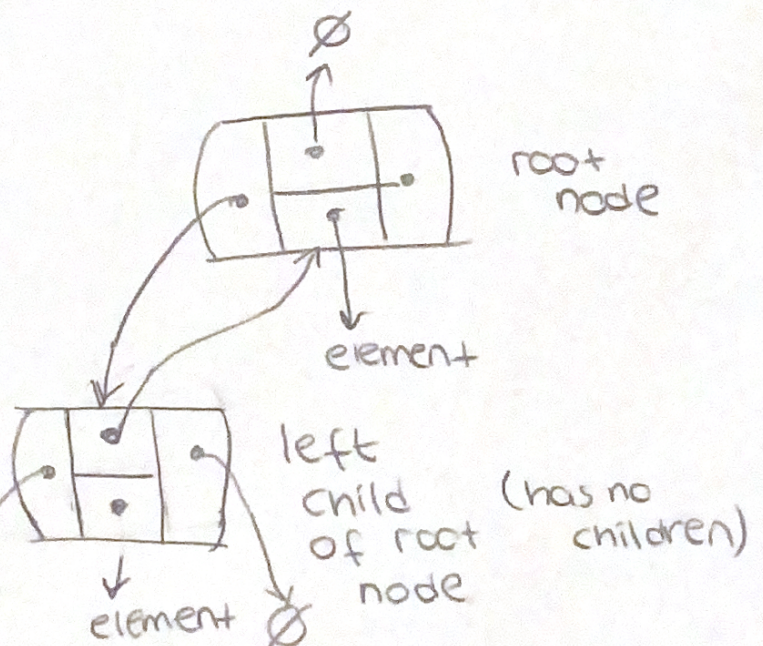
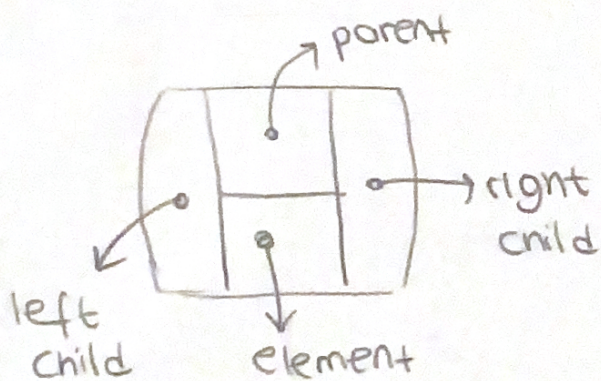
$$f(p_1) = 2f(q) + 1$$

$$f(p_2) = 2f(q) + 2$$

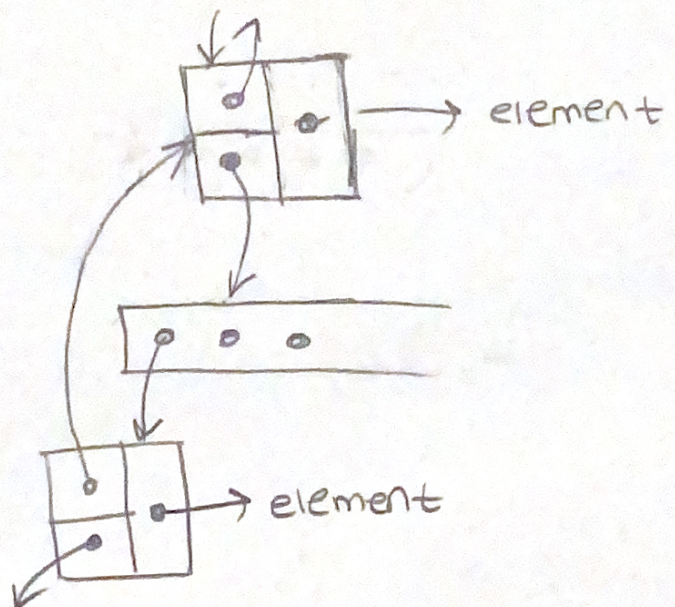
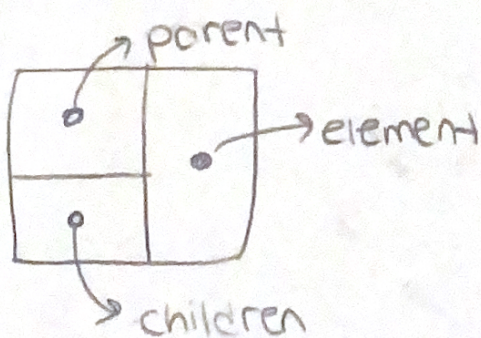
$$f(q) = \lfloor (f(p) - 1) / 2 \rfloor$$

→ aṣaḡi yuwarla

Link Based Binary Tree



Link Based General Tree



Trees

Binary Tree

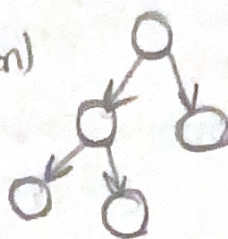


D	n
0	1
1	3
2	7
3	15
4	31
depth	nodes

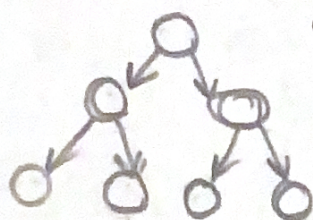
logn (insertion)

Strictly Binary Tree

0 or 2 child



Complete Binary Tree



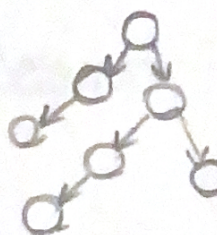
$$\text{depth} = \log_2(N+1) - 1$$

vertices

$$\text{vertices} = 2^{k+1} - 1$$

depth

Balanced Tree

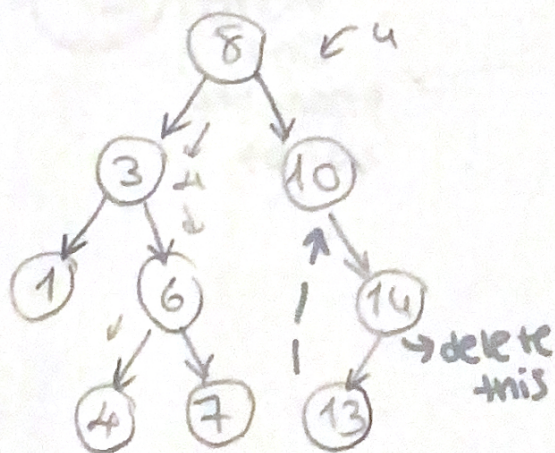


depth differ by 1
right = left

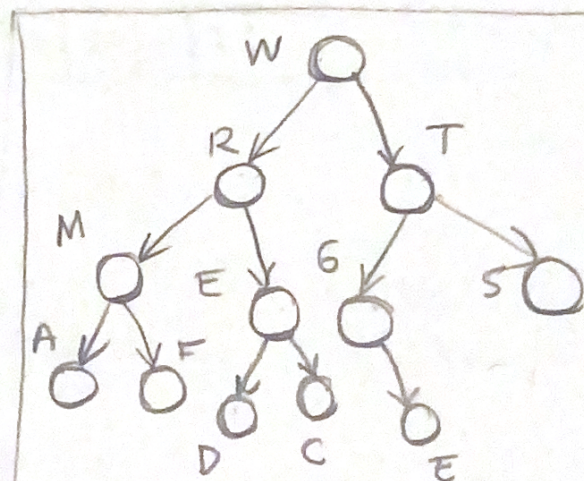
Binary Search Trees

Left subtree < Node key

Node key < Right subtree



Almost complete tree lede deletion da insertion da rep sondakine. uyguur



W	R	T	M	E	G	S	A	F	D	C	E
---	---	---	---	---	---	---	---	---	---	---	---

$$\text{parent}(i) = \lfloor i/2 \rfloor$$

→ azoqi yuvorla

Insertion in BST

1. Test key to be inserted against root. If it's smaller, insert it on the left, otherwise (2) insert it on the right.
2. Search until subtree is empty. (Link to a new node with respective parent)

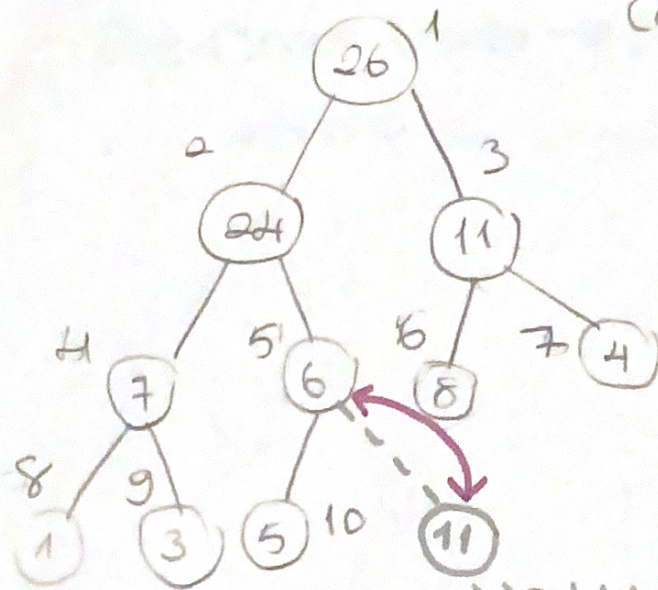
Deletion

If leaf, disconnect.

If it has a left child but no right child, connect parent to that child.

Otherwise find it's LNR successor

Heaps

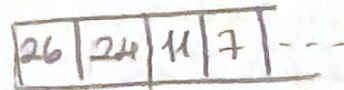


Completely balanced binary tree with heap property.

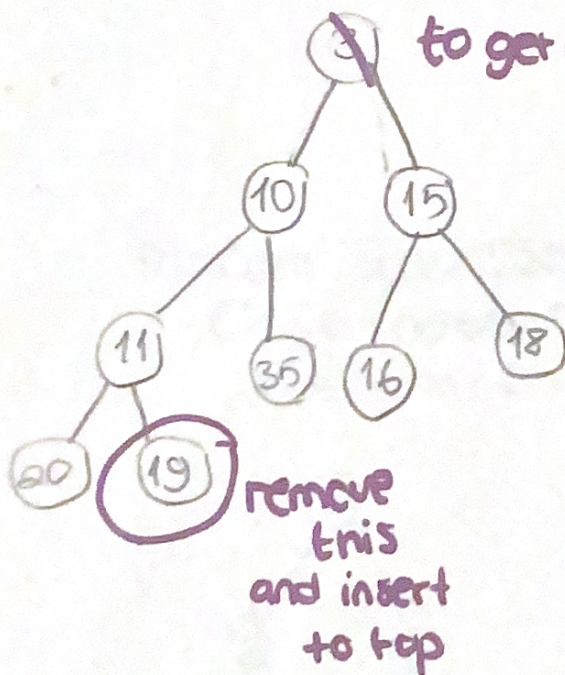
Heapify $\rightarrow O(\log n)$

Insert to last element and do heapify

(k, v)

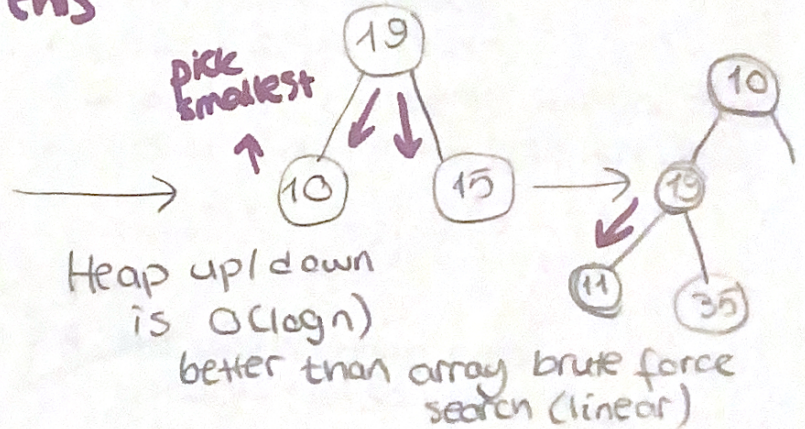


Deletion



to get rid of this

pick smallest



Heap up/down is $O(\log n)$

better than array brute force search (linear)

Heapify:

$\lfloor \frac{n}{2} \rfloor \rightarrow$ first parent from end

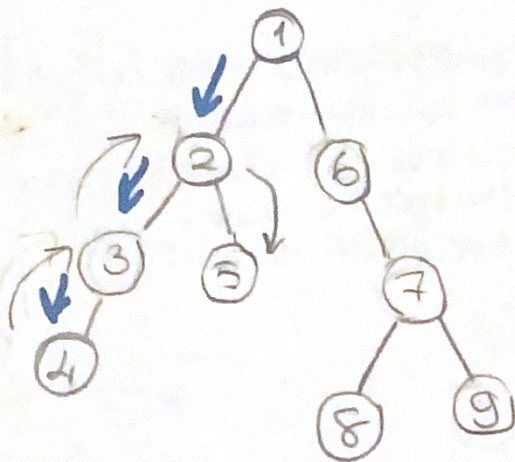
(11 here) then everything before that

Tree Traversal

Pre-Order (node-left-right)

traverse-nlr(t-node) {

1. Recursion base case (en uytaki node'lar için) çocuğu yoksa value döndür.
2. Çocuğu varsa
kendisini döndür $\xrightarrow{\text{traverse}} n.\text{leftchild}$
sol çocuk için traverse-nlr
sağ çocuk için traverse-nlr



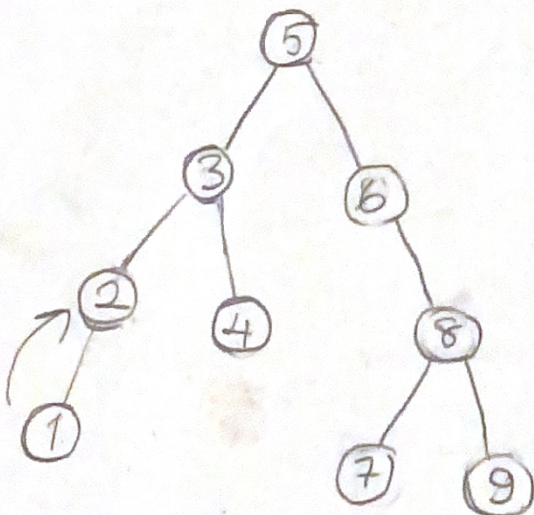
• İterasyonlar

1 $\rightarrow$ n
1 2 $\rightarrow$ traverse 1.lchild (return n)
1 2 3 $\rightarrow$ traverse 2.lchild (return n)
1 2 3 4
1 2 3 4 (3'e döndü)
1 2 3 4 (2'ye döndü)
1 2 3 4 5 $\rightarrow$ traverse 2.rchild
1 2 3 4 5 6 $\rightarrow$ traverse 1.rchild
--

Inorder Traversal (left-node-right)

traverse Inr(t-node) {

1. Recursion base case:
Çocuğu yoksa kendisini döndür.



2. Çocuğu varsa;
traverse-Inr(sol çocuk)
visit+tree(t-node)
traverse-Inr(sağ çocuk) }

Post Order Traversal (Left-right-node)

traverse-lrn(t-node) {

 acugu yoksa value'yu
 dondur (yaprak node)

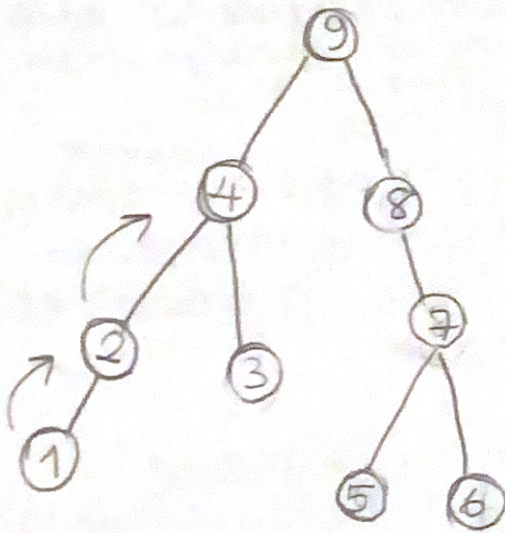
 acugu varsa {

 traverse-lrn(t-node.lchild)
 (sol acugu traverse et)

 traverse-lrn(t-node.rchild)

 visittree(t-node)
 (kendisini dondur)

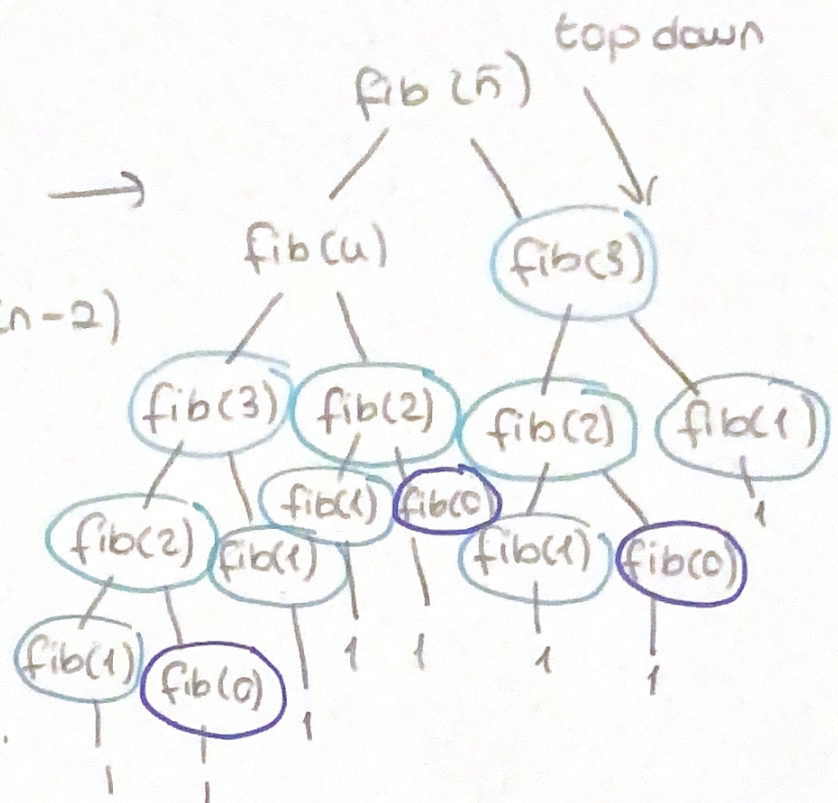
 }



Dynamic Programming

Fibonacci

$fib(n)$
 if $n=0$ or $n=1$
 return 1
 else $fib(n-1) + fib(n-2)$



Overlapping Subproblems?
 Use memorization!
 Store smaller subproblems.

0	1	2	...
1	1	2	...

do bottom up approach!

$Fib(n)$
 $F[0] = 1$
 $F[1] = 1$

$F(n) = \begin{cases} 1 & \text{if } n=0 \\ & n=1 \\ F[n-1] + F[n-2] & \text{otherwise} \end{cases}$
 Recurrence Equation

$F[0] = 1$
 $F[1] = 1$
 $i = 2$
 while $i < n$
 $F[i] = F[i-1] + F[i-2]$
 return $F[i]$

sadece n
 kez hesapladı
 (linear time)

↑ (not efficient with space)

Longest Common Subsequence

Subsequence $\rightarrow$ ACEGIJK
DFGHJK

Not subsequence $\rightarrow$ DAGH

a b b c d c
d b b a c

$> |LCS| = 3$

önceli
karakterler
de aynıdır
1 arttır.

If they don't match:

Check $LCS[j, k-1]$ or $LCS[j-1, k]$

$$X = \overbrace{ACDBAC}^{x'} \bar{A}$$

$$Y = \underbrace{ABCADDF}_{y'} \bar{A}$$
 if same, check earlier

if same,
check
earlier

LCS Algorithm

LCS [x, y]
 $\downarrow$ $\downarrow$
 n m

Running Times $\rightarrow$ Brute force: $2^n \times m$
 Dynamic programming: $O(nm)$

for $i=1$ to $n-1$, do;

$L[i, -1] = 0$

for $j=0$ to $m-1$, do

$L[-1, j] = 0$

for $i=0$ to $n-1$, do

for $j=0$ to $m-1$, do

if $x_i = y_j$, then

$L[i, j] = L[i-1, j-1] + 1$

else

$L[i, j] = \max\{L[i-1, j], L[i, j-1]\}$

return L

ex

$i=0$
 A B C D D F $\neq$

$j=0$
 C E D D G $\neq$ F

$F(5, 6)$

4, 5

$F(6, 5)$

5, 4

+1

L	-1	0	1	2	3	4	5	6
-1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	1	1	1	1	1	1	1
3	0	1	2	2	2	2	2	2
4	0	1	2	3	3	3	3	3
5	0	1	2	3	3	3	3	4
6	0	1	2	3	3	3	4	4

Output of LCS

```
def LCS-output(X, y, L):  
    solution = []  
    j, k = len(X), len(y)  
    while L[j][k] > 0:  
        if X[j-1] == y[k-1]:  
            solution.append(X[j-1])  
            j -= 1  
            k -= 1  
        elif L[j-1][k] >= L[j][k-1]:  
            j -= 1  
        else:  
            k -= 1  
    return "".join(reversed(solution))
```

	-1	A	B	B	C	B	A
-1	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1
C	0	0	1	1	2	2	2
B	0	0	1	2	2	3	3
A	0	1	1	2	2	3	4

Huffman Codes and Text Compression

Representing letters $\rightarrow$

A	B	C	D	$\emptyset$
000	001	010	011	100

 if we represent these with binary code how many digits should we use?

ABCDABCBBA = m

$\rightarrow$ 000001010011000 ... = 3x m bits

$\rightarrow$ encoding table

$$2^m > 5$$

$\rightarrow$ use 3 digits to represent 5 chars

Variable Sized Encoding $\rightarrow$

d = number of chars
n = number of items
total comp = $n + d \log d$

A $\rightarrow$ 0

B $\rightarrow$ 10 $\rightarrow$ 0 ile başlayamaz

C $\rightarrow$ 110 $\rightarrow$ 10 ve 0 ile başlayamaz

D $\rightarrow$ 111 $\rightarrow$ 10, 110, 0 ile başlayamaz

ABCCABBA
01011011000100

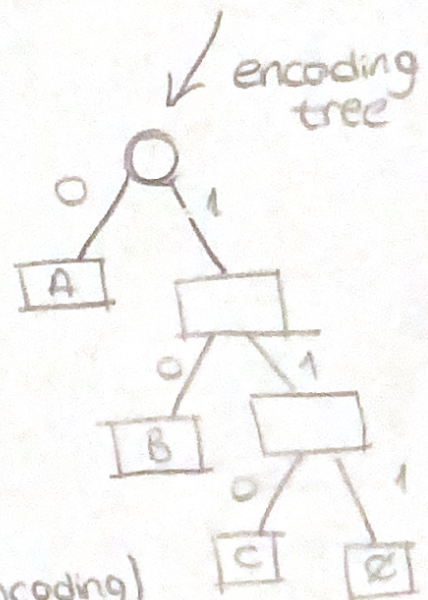
exu

A	B	C	D	-
0.35	0.1	0.2	0.2	0.15

 } letter frequencies

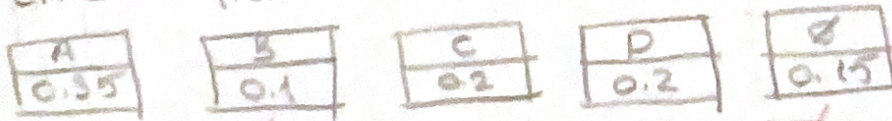
Fixed Encoding
100 letter text $\rightarrow$

we need 3 bits
 $3 \times 100 = 300$ bits



If we can build up a variable encoding tree we can decrease the space (Huffman encoding)

- En küçük frekansları recursive olarak birleştir.



* complexity of construction $d \log d$

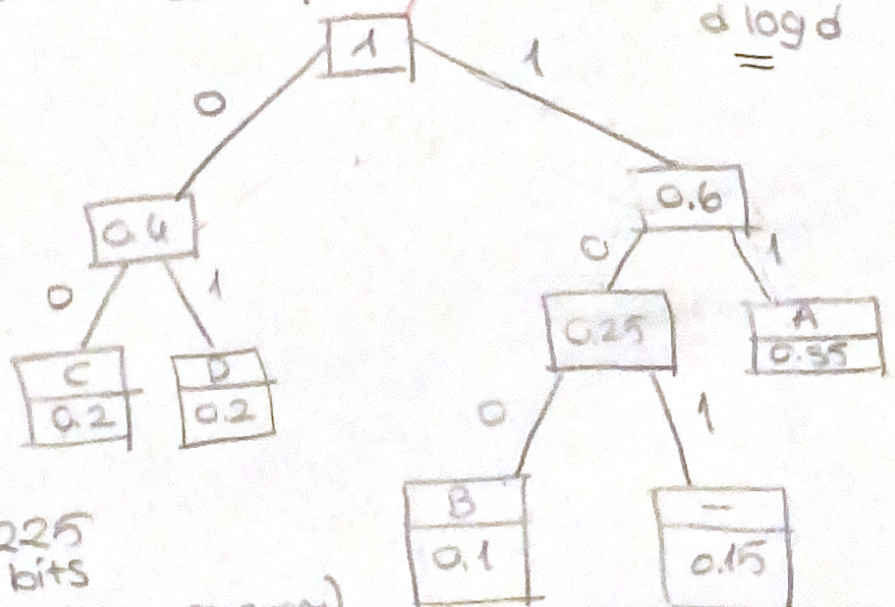
A $\rightarrow$ 11

B $\rightarrow$ 100

C $\rightarrow$ 00

D $\rightarrow$ 01

$\emptyset \rightarrow$ 101



100 letter $\rightarrow$ 35×2 A
 10×3 B
 20×2 C
 20×2 D
 15×3 $\emptyset$

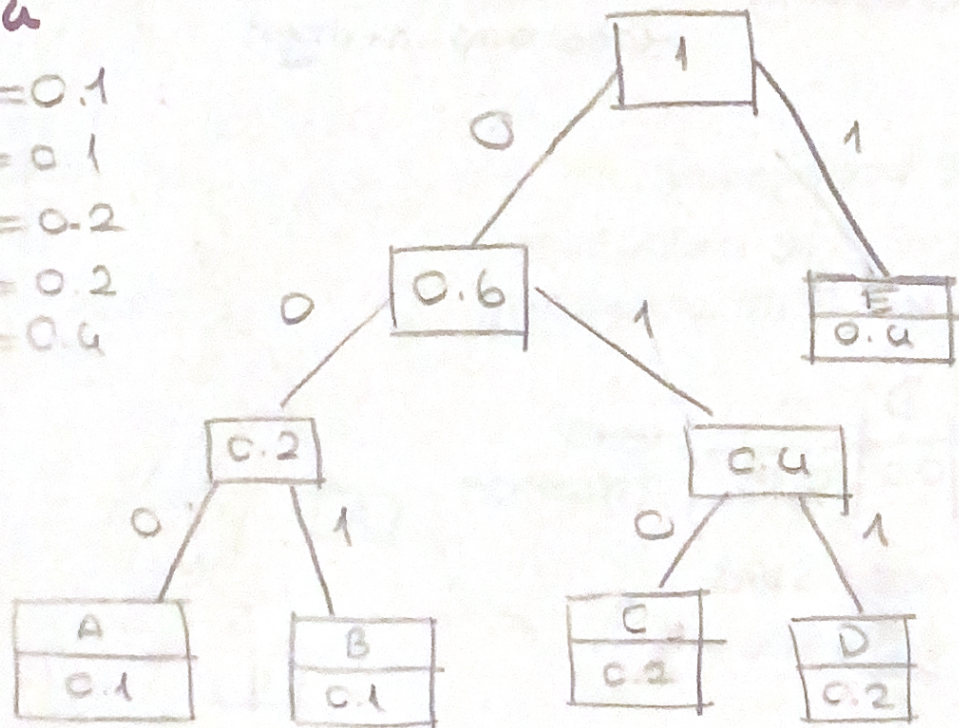
$\rightarrow$ 225 bits
(25% improvement)

Huffman's Algorithm

1. Initialize n one node trees and label them with letters and ~~sequences~~ frequencies
2. Repeat this step until every node is combined in a single tree. Combine them and make left & right subtree of a root which has the combined frequency.

ex 4

A = 0.1
B = 0.1
C = 0.2
D = 0.2
E = 0.4



A = 000
B = 001
C = 010
D = 011
E = 1

100 letters

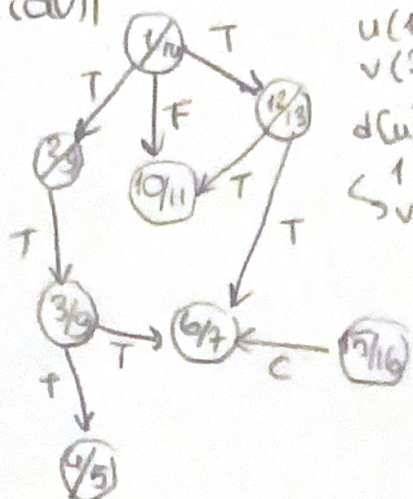
A = 3×10
B = 3×10
C = 3×20
D = 3×20
E = 1×40

Fixed Encoding $\rightarrow 3 \times 100 = 300$ bits

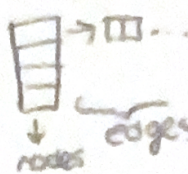
Variable Encoding $\rightarrow 220$ bits

220 bits

- DFS $\rightarrow$ Input $G(V, E)$
 $\text{time}(O(V+E))$ Output $d[u]$ & $f[u]$
 $\text{space}(O(V))$



$u(1,1u)$
 $v(2,7)$ $\rightarrow$ ancestor
 $d[u] < d[v] < f[v] < f[u]$
 $\hookrightarrow$ child of u



adj list

space $\rightarrow O(V+E)$

$\hookrightarrow$ pick if sparse

time $\rightarrow O(V+E)$

(drawing)

$O(V) \rightarrow$ check edge

adj matrix

$V \rightarrow$ nodes space $\rightarrow O(V^2)$
 $E \rightarrow$ edges time $\rightarrow O(V^2)$

$\hookrightarrow$ pick if dense

check edge $\rightarrow O(1)$

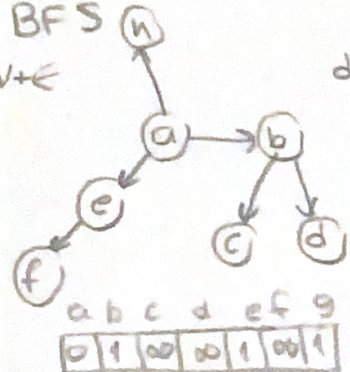
sparse $\rightarrow V \sim E$ (linear)

dense $\rightarrow E = \frac{V(V-1)}{2}$

worst $E \sim O(V^2)$

BFS

$V \in E$



$d[u] = \frac{a b c}{b a c} \dots$

queue

= [a]

$\hookrightarrow$ add visited
 dequeue when traversing neighbors

a	b	c	d	e	f	g
0	1	0	0	1	0	1

[b, e, g] $\rightarrow$ dequeue b, look at b's neighbors

MST

Prim ($O(V^2)$) (node selection)

$F = \emptyset \rightarrow$ edges

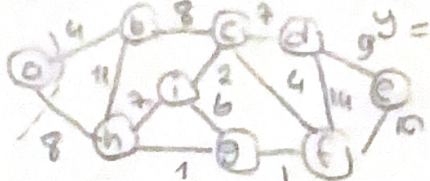
$Y = \{a\}$

Pick b

$F = \{a, b\}$

$Y = \{a, b\}$

finish
 all nodes on Y



Kruskal

$S = \{s_1, s_2, \dots, s_n\}$

$F = \{f_1, f_2, \dots, f_n\}$

$F = \{f_1\} \rightarrow S = \{s_1, \dots, s_n\}$

edges

hg 1

gf 2

ci 2

finish when all are one set

$\Theta(E \log E)$

LCS

ABCDDEFZ

CEDDGEZF

L	-1	A	B	C	D	E	F	Z
-1	0	0	0	0	0	0	0	0
C	0	0	0	1	1	1	1	1
E	0	0	0	1	1	1	1	1
D	0	0	0	1	2	2	2	2
D	0	0	0	1	2	3	3	3
G	0	0	0	1	2	3	3	3
E	0	0	0	1	2	3	3	3
F	0	0	0	1	2	3	3	3

Preorder $\rightarrow$ NLR

Inorder $\rightarrow$ LNR

Postorder $\rightarrow$ LRN

- Huffman $\rightarrow$ en karakterleri birleştirir, ağaç oluştur (comp = $n + d \log d$)

Fixed encoding = $2^n \times 5$

3 olmalı, 5 harf sayısı

Selection Sort (n^2)

Find smallest and swap with first (then second)

7 3 2 6 1 9 5

1 3 2 6 7 9 6

1 2 3 6 7 9 5

Bubble Sort

Swap adjacent if wrong order

5 1 4 2 8 $\rightarrow$ 1 5 4 2 8

A swap with no pass ends the algo