10/21/14 - Tuesday

I. Computer Architecture

Skip

II. American History I

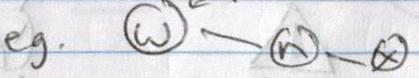
Skip

III. Data Structures

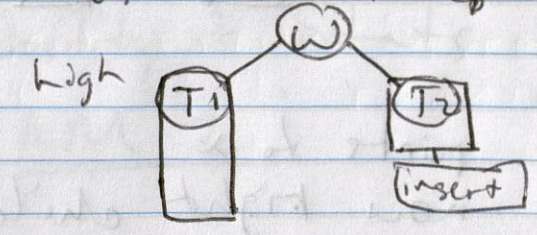
30 Mins Late

- AVL Trees

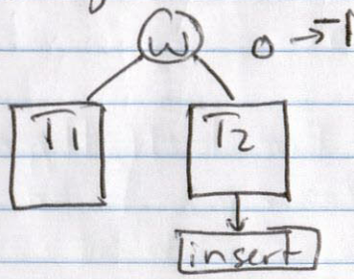
- The balancing factor of x's parent has changed

eg. 

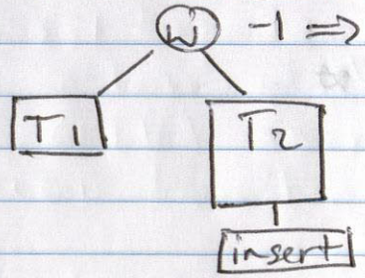
- If w was originally high in the opposite direction of x, then we have not changed the height of the tree, therefore, we simply change the balancing factor to level, and we are done!



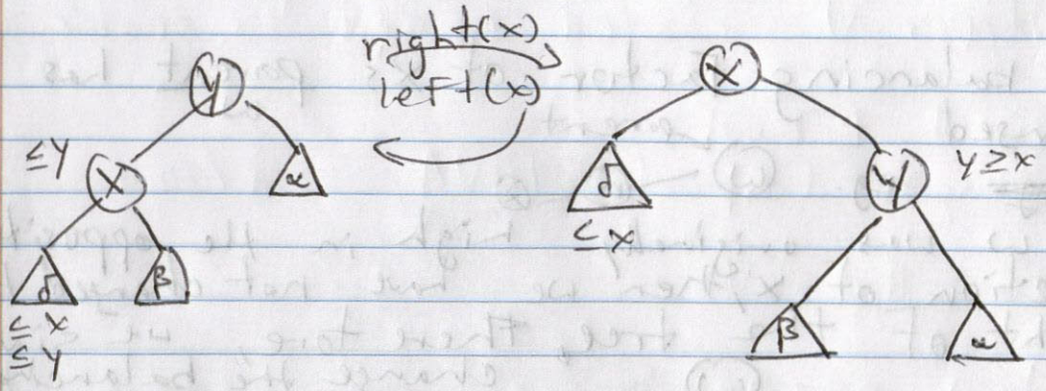
- If w was originally level-balanced, then we change its balancing factor to right-high. w's parent now has its height increased by 1 so we must ~~to~~ go to w's parent and repeat this procedure.



- If w was originally high in the same direction as x, we have increased the tree (height) and unbalanced w. We must therefore correct the tree using rotations.



✻Rotation Worksheet✻

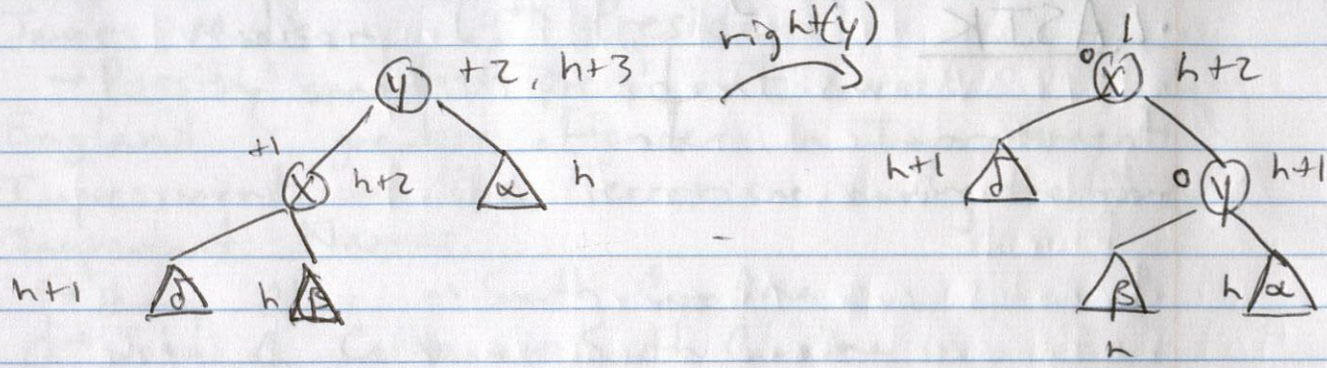


right-rotate

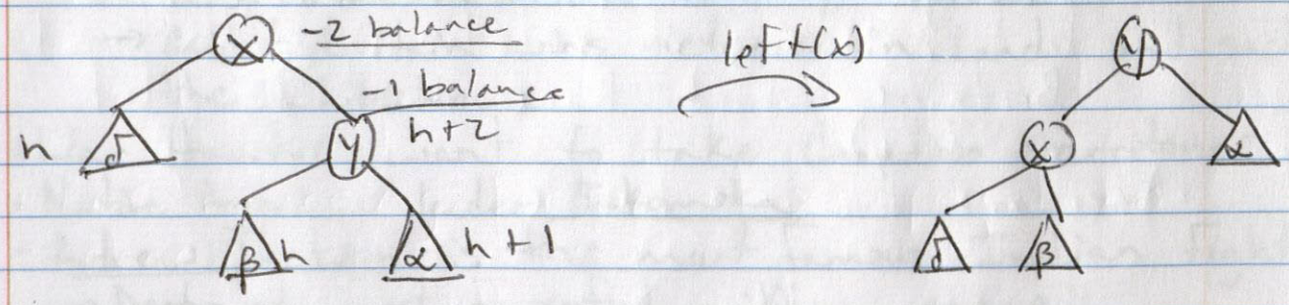
- parent of y now points to x
- y's left child is now right child of x
- x's right child is now y

↳ · no change in balance factors δ, β, α

· Simple Right Rotation
when the inserted item is in the left subtree of the left child of nearest ancestor with factor +2



· Simple Left-Rotation
when the inserted item is in the right subtree of the right child of the nearest ancestor with factor -2



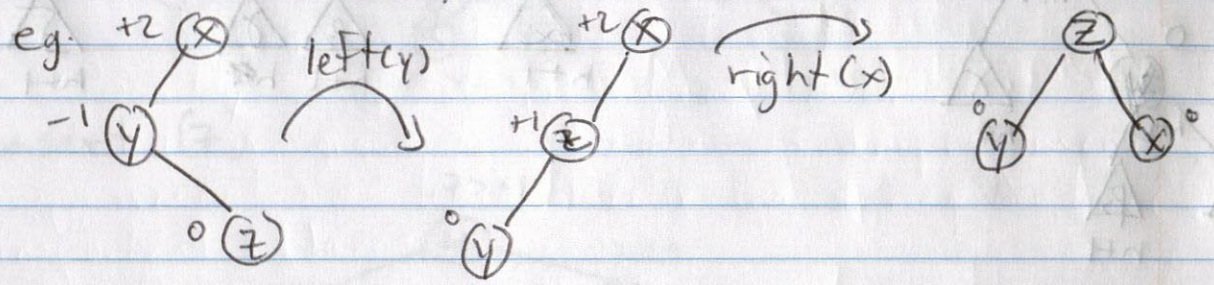
IV. Probability & Stats

EXAM 2!

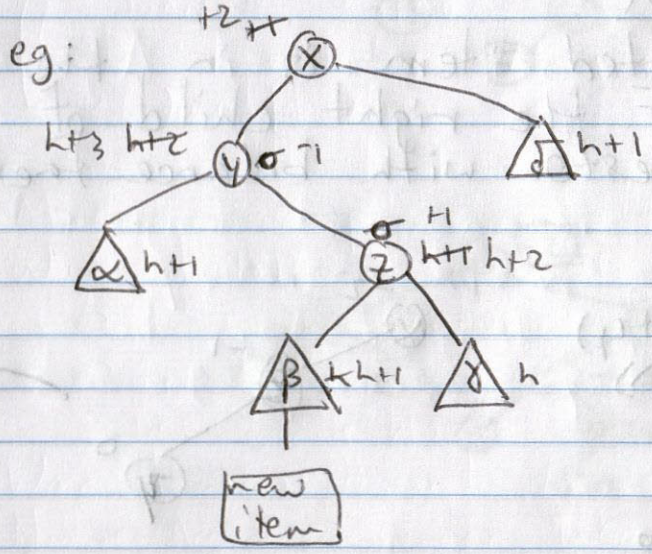
III. Data Structures

- AVL Trees
- Left-Right Rotation
  - when the inserted item is in the right subtree of the left child of the nearest ancestor with factor +2

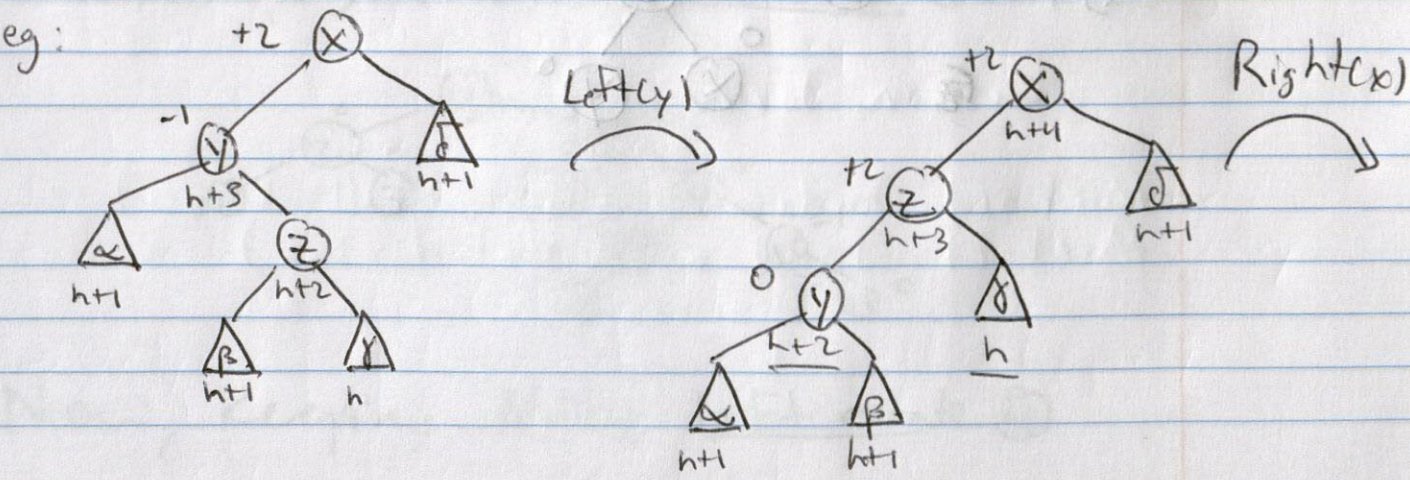
eg:

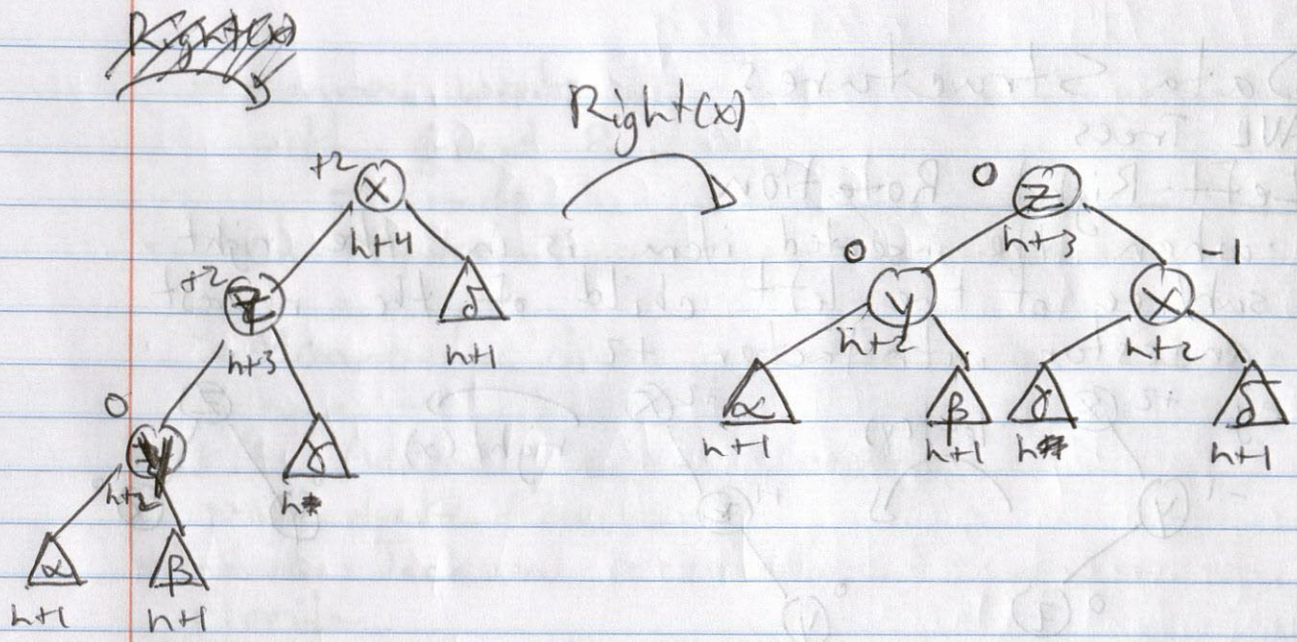


eg:



eg:

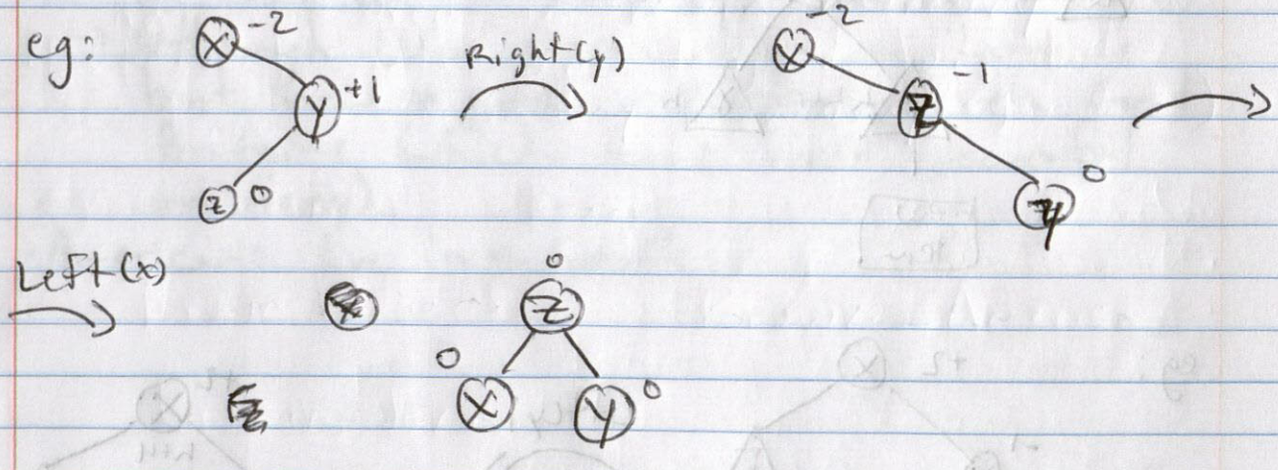




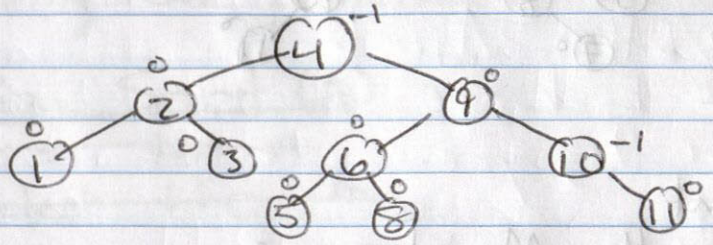
- Right-Left Rotation

  when the inserted item is in the left subtree of the right child of the nearest ancestor with balance factor -2.

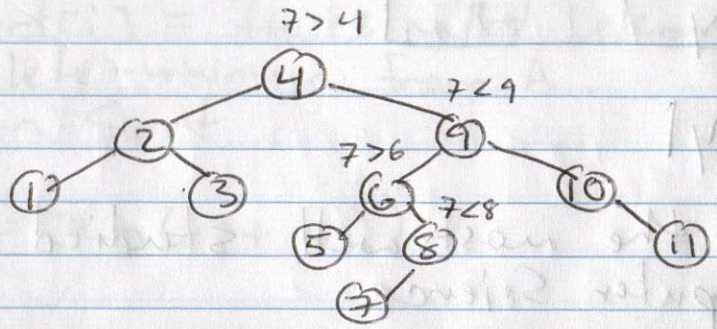
eg:

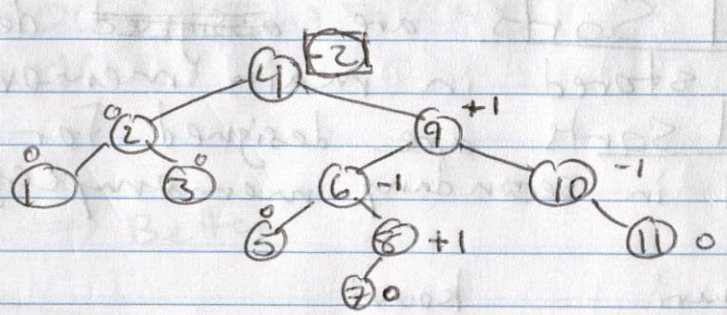


eg: Binary Search Tree? Yes
AVL Tree? Yes

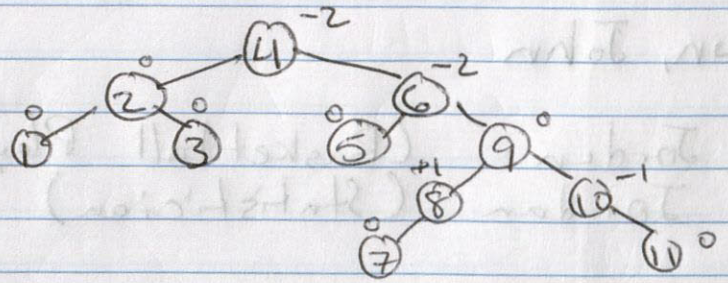


Insert (7)

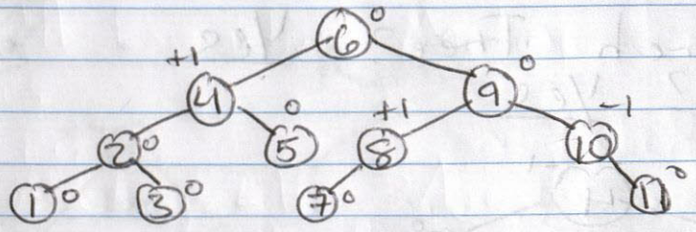




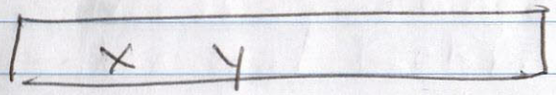
Now, keeping AVL, right rotate ⑨ (Right-Left)



Now, keeping AVL, left rotate ④



- Sorting
  - → Important
  - → Traditionally Mainframe



$|x - y|$

- Sorting is the most well studied Algorithm in computer Science

- Internal Sorts are ~~assigned~~ designed For data stored in main memory
- External Sorts are designed for data stored in secondary memory (eg., disk)

eg: Kouri kouri

Brown-Williams Brown_America

Brown, John

Michael Jordan (Basketball Player)
Michael Jordan (Statistician)

- Sorting algorithms that always leave equal items in the same relative order as in the original permutation are called stable

- Selection Sort (A)

~~For i=1 to n~~
~~select the ith~~

$O(1)$ Sort[ ]
$O(n)$ { For i=1 to n
| Sort[i] = the smallest element from A $T(L3)$
| Delete-minimum from A $T(L4)$
$O(1)$ Return Sort

$T(n) = O(n(L3 + L4))$

array
$T(L3) = O(n)$ } $T(n) = O(n^2)$
$T(L4) = O(1)$ }

binary tree
$O(\log n) \Rightarrow$ Better

binary heap
Best!

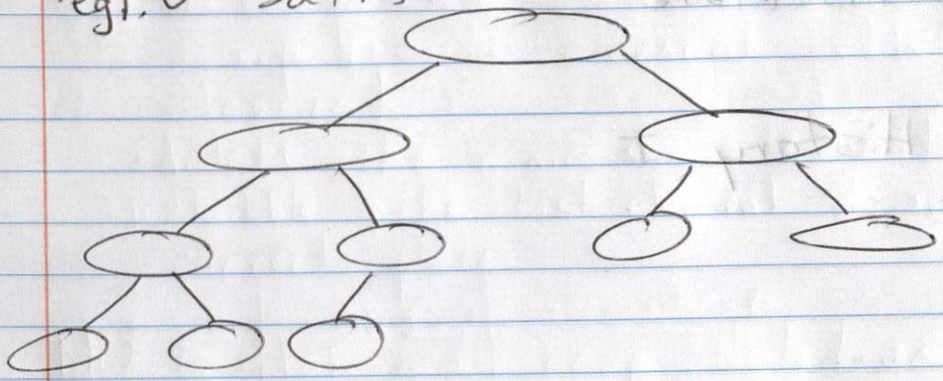
10/28/14 - Tuesday

I. Computer Architecture

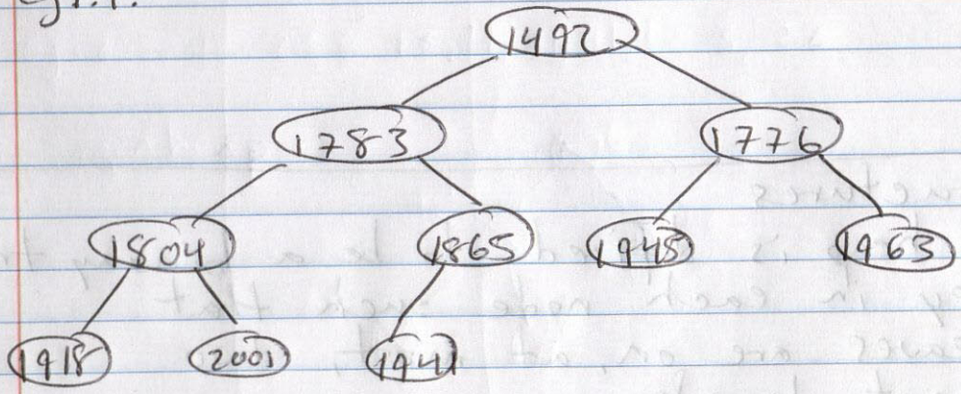
II. American History I
- EXAM 2!

III. Data Structures
- A binary heap is defined to be a binary tree with a key in each node such that
  1. all leaves are on, at most, two adjacent levels
  2. all leaves on the lowest level occur to the left, and all levels except the lowest level are completely filled.
  3. The key in any node dominates the keys in its children
     - → min-heap: a node dominates its children by containing a smaller key than they do
     - → max-heap: a node dominates its children by containing a larger key than they do.

eg1: ✓ Satisfies Conditions ① + ②



eg1.1:



eg1.2:

0	1492
1	1783
2	1776
3	1804
4	1865
5	1945
6	1963
7	1918
8	2001
9	1941

the left child of k sits in position $2k+1$ and the right child in position $2k+2$

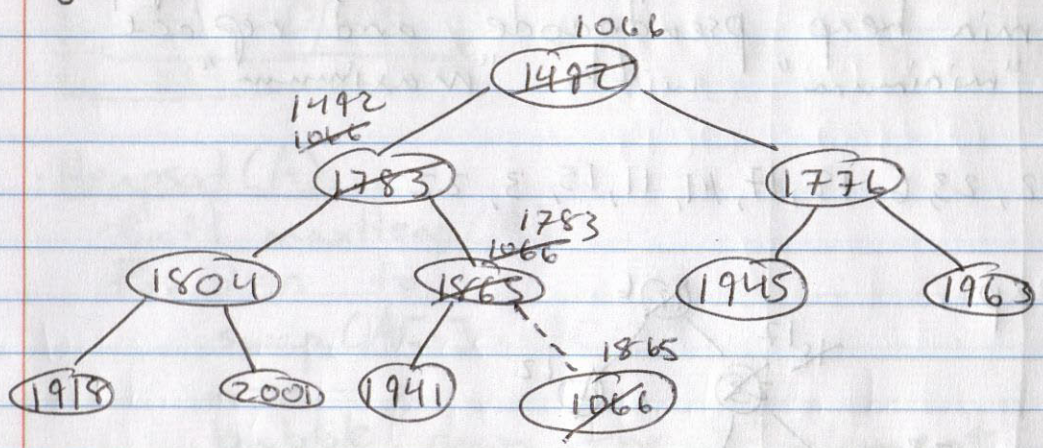
parent of k is in position $\frac{k-1}{2}$?

max

Basic Heap Operation

- Construct
- Check if empty
- insert
- retrieve the max element
- delete the max element

eg.13: Insert 1066



Bubble AKA Percolate

- heap insertion
  - Put new element in the (n+1)st position
  - "Bubble" the new item up to the correct place

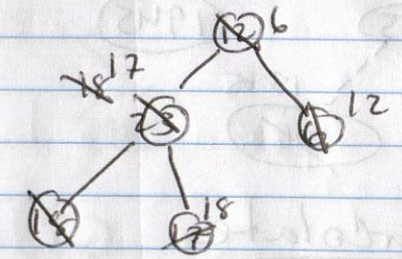
eg. min heap deletion

- Swap the last element with the minimum element.
- delete the minimum element
- "Bubble" the last element down to the correct place.

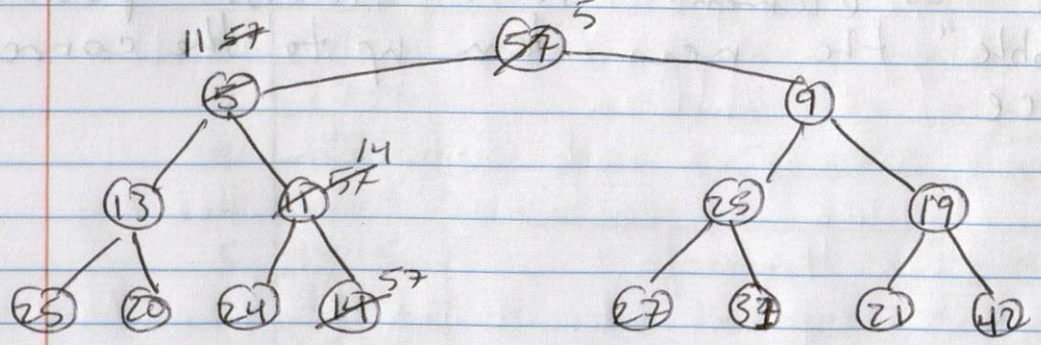
NOTE: For max heap deletion, simply take min heap pseudocode and replace "minimum" with "maximum"

eg. 2: 12, 23, 6, 18, 17, 14, 11, 15, 3, 22

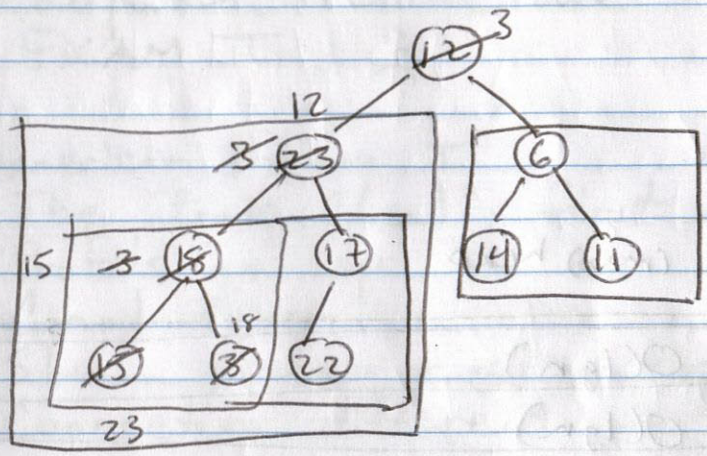
minheap



eg. 3: A = [5, 13, 11, 25, 20, 24, 14]
B = [9, 25, 19, 27, 31, 21, 42]



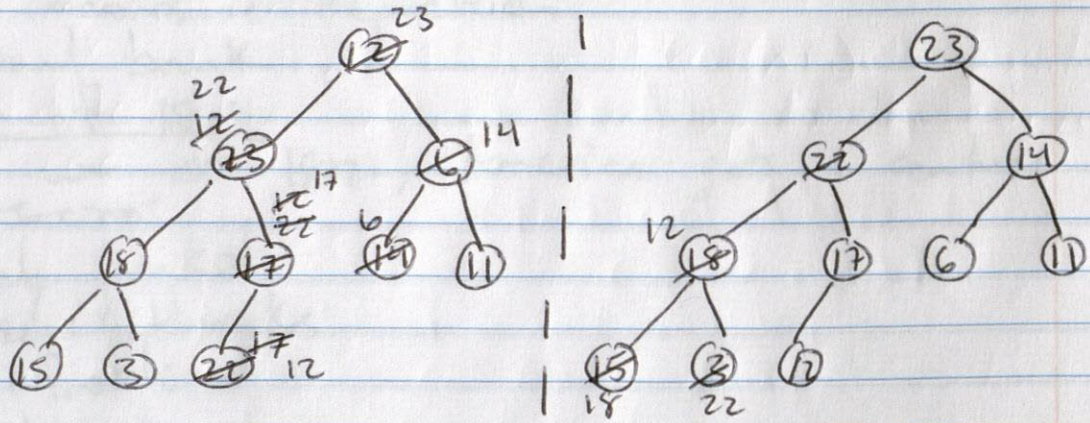
eg. 21: 12, 23, 6, 18, 17, 14, 11, 15, 3, 22



O(n) to build a heap

- Heapsort(A)
  - Build maxHeap(A)
  - For i=n to 1
    - swap(A[1], A[i])
    - n=n-1
    - bubble down the 1st element

eg. 22: 12, 23, 6, 18, 17, 14, 11, 15, 3, 22



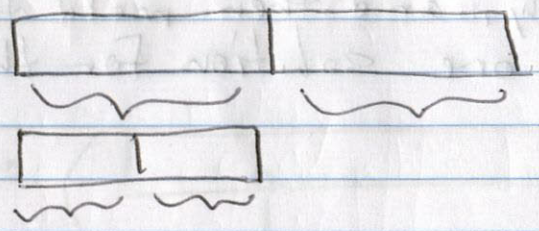
max

- Priority Queue Container
  - Construct
  - insert
  - delete-max
  - find-max

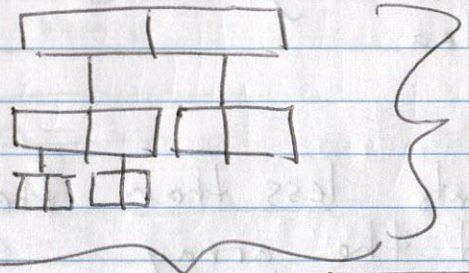
eg. 4:	Balanced BST	(max) heap
Insert	$O(\lg n)$	$O(\lg n)$
delete-max	$O(\lg n)$	$O(\lg n)$
Find-max	$O(\lg n)$	$O(1)$

III. Data Structures

- MergeSort



eg 1:



Total work: $\boxed{O(n \lg n)}$

- Pseudo code:

```
void mergesort(S, low, high) { // Recursive
    if (low < high) {
        middle = (low + high) / 2

        mergesort(S, low, middle)
        mergesort(S, middle+1, high)
        mergesort(S, low, middle, high)
    }
}
```

eg 2: A = {5, 7, 12, 19}
B = {4, 6, 13, 15}

merged = {4, 5, 6, 7, 12, 13, 15}

// assuming both A & B are sorted already

divide-and-conquer Algorithms

- divide the problem into smaller subproblems, solve each recursively, and then meld the partial solutions into one solution for the full problem.

Quicksort

- quicksort uses partitioning about a pivot as its main idea.

main idea

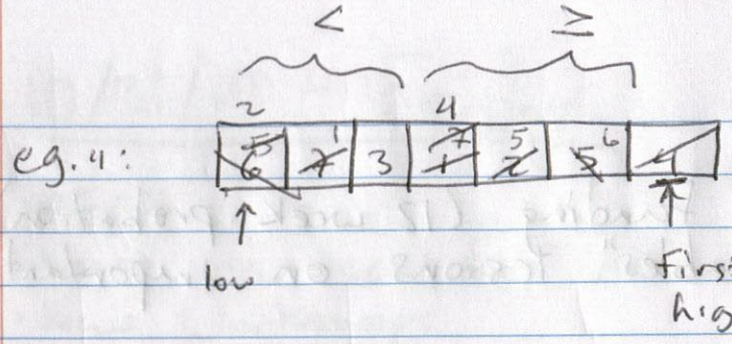
- place all elements less than the pivot in the left part of the array
- place all elements greater than or equal to the pivot in the right part of the array
- the pivot fits in the slot between them

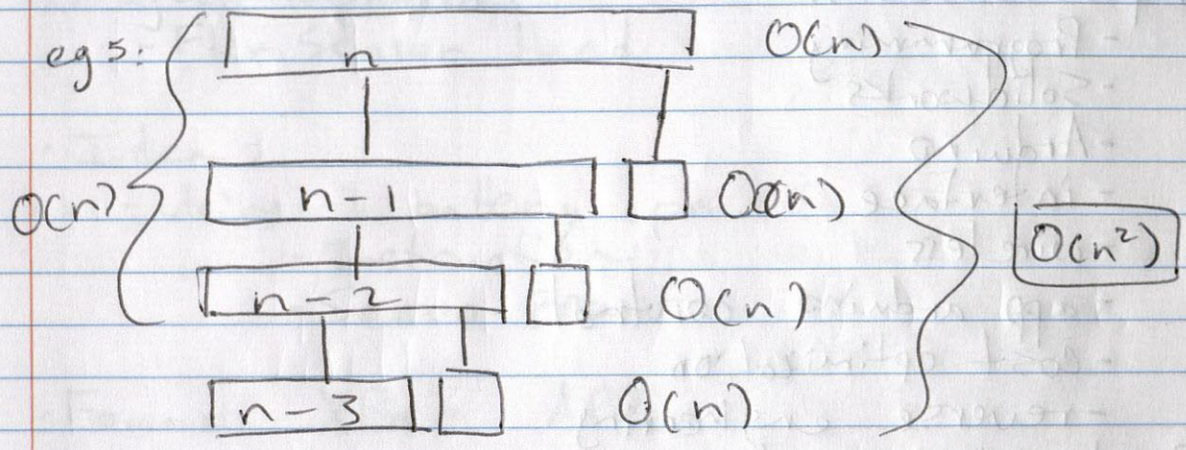
eg.3: {6, 7, 3, 1, 2, 5, 4}

[{3, 1, 2} {4} {6, 7, 5}]

```
quicksort(A, n)
    quicksort(A, 0, n-1)

quicksort(A, low, high)
    if (low < high)
        pivotLocation = Partition(A, low, high)
        quicksort(A, low, pivotLocation-1)
        quicksort(A, pivotLocation+1, high)
```





- If we choosing deterministically
  "If you give me random input data, quicksort runs in expected O(nlgn) time"
- Choose a random ~~point~~ pivot.
  "With high probability, randomized quicksort runs in O(nlgn) time"

11/4/14 - Tuesday

I. Computer Architecture

II. American History I

III. Data Structures

eg.

E	A	T
0	1	01

0 1 0 1
T E A

one way to ensure that no bit string corresponds to more than one sequence of letters is to encode letters so that the bit string for a letter never occurs as the first part of a bit string for another letter.

prefix codes

eg.

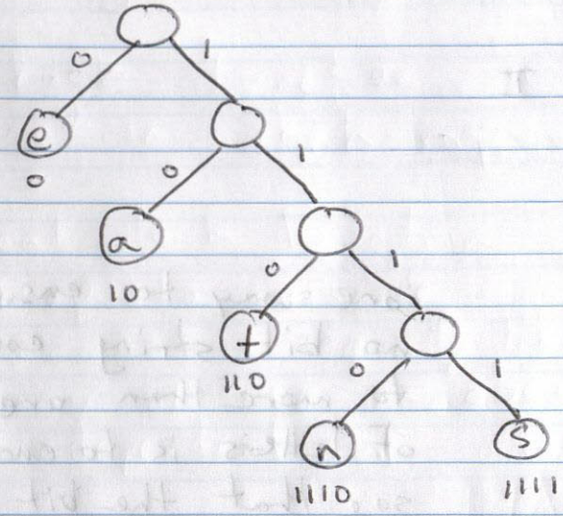
E	A	T
0	10	11

10 11 0
A T E

- a prefix code may be represented as a ~~st~~ binary tree where
  - the characters are the labels of the leaves of the tree
  - the edges are labeled so that an edge leading to a left child is assigned a 0 and an edge leading to a right child is assigned a 1.
- the bit string used to encode a character is the sequence of labels of the edges in

the unique path from the root to the leaf that has this character as its label.

eg.



1111 10 1110 0
s a n e

Huffman Coding:

input symbols a_i with frequencies w_i

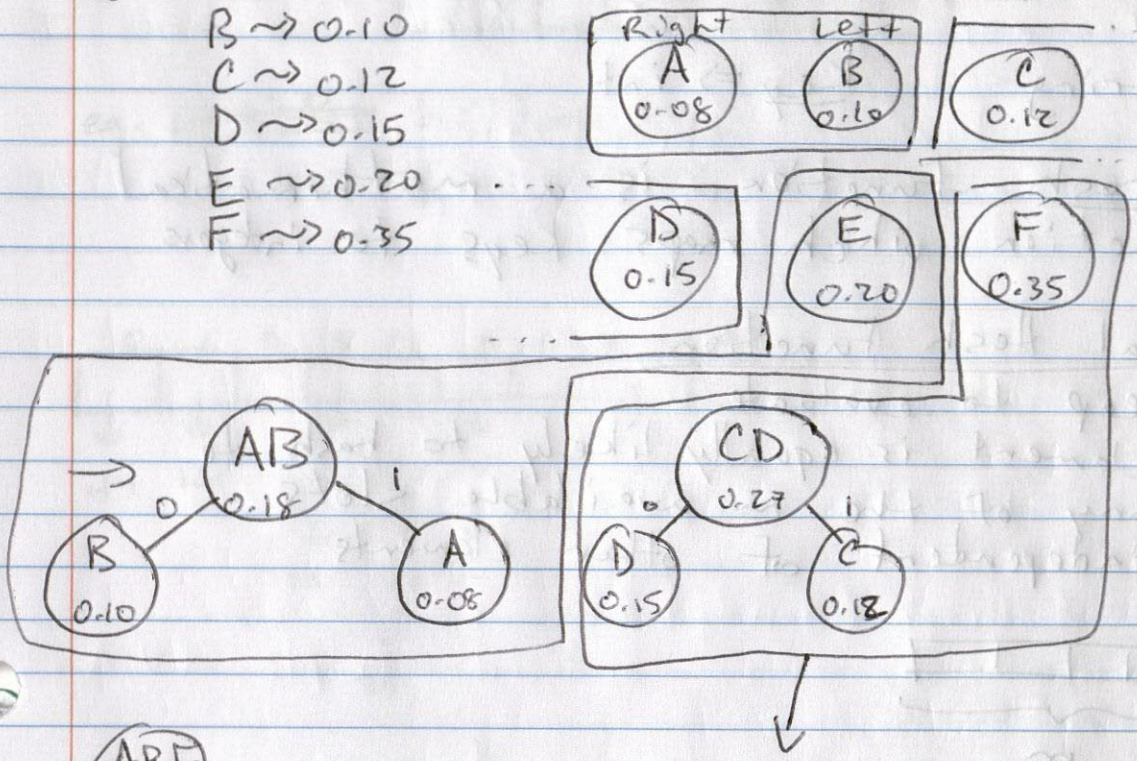
f = forest of n rooted trees, each consisting of a single vertex a_i and weight w_i

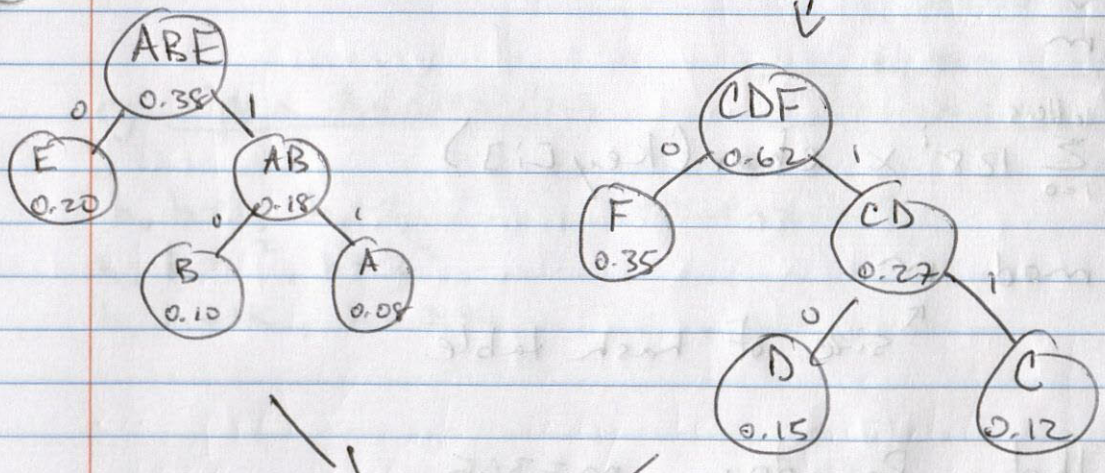
while (F is not a tree)
- replace rooted trees T and T' of least weights from F with $w(T) \geq w(T')$ with a tree having a new root that has T as its left subtree and T' as its right subtree.

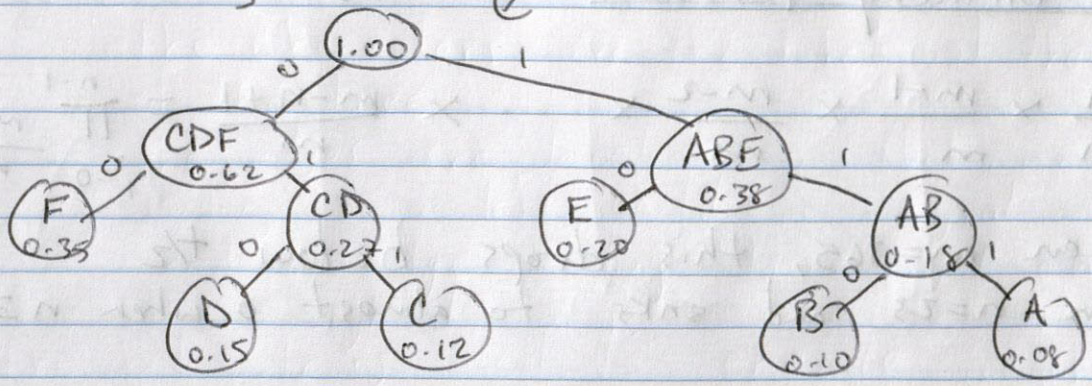
// Note: light = right
// heavy = left
- label the new edge to T with 0 & the new edge to T' with 1
- assign $w(T) + w(T')$ as the weight of the new tree

eg. A ⇝ 0.08
B ⇝ 0.10
C ⇝ 0.12
D ⇝ 0.15
E ⇝ 0.20
F ⇝ 0.35

Step1: Create Forest

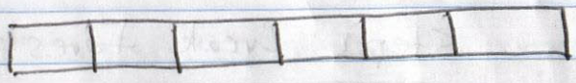






A B C D E F

0.08×3 + 0.1×3 + 0.12×3 + 0.15×3 + 0.2×2 + 0.35×2 = 2.45

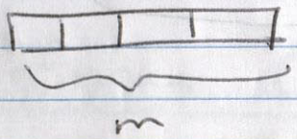
eg. 

String $\xrightarrow{\text{Key}}$ int

- a hash Function is a mathematical Function which maps keys to integers

Ideal Hash Function

1. cheap to evaluate
2. element is equally likely to hash to any of the m available slots
3. independent of other elements



$$h(x) = \sum_{i=0}^{\text{letters}} 128^i \times \text{char}(\text{key}[i])$$

$h(x) \bmod m$

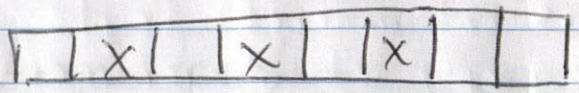
↖ size of hash table

The Birthday Paradox $m = 365$

$$\frac{m}{m} \times \frac{m-1}{m} \times \frac{m-2}{m} \times \cdots \times \frac{m-n+1}{m} = \prod_{i=0}^{n-1} \frac{m-i}{m}$$

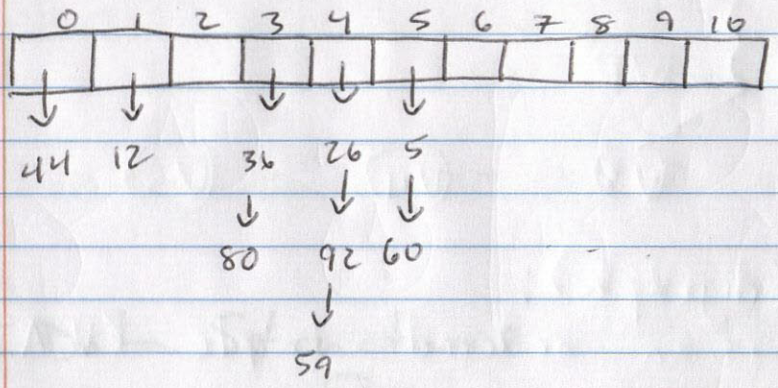
when $m = 365$, this drops below $1/2$ when $n = 23$ and sinks to almost 0 when $n \geq 50$

Chaining

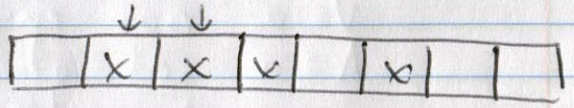


eg. ~~h(i)%11~~ $h(i) = i \% 11$

26, 42, 5, 40, 44, 92, 59, 76, 12



eg. Open Addressing



h, h+1, h+2, ... sequentially

$h, h+1^2, h+2^2, \ldots$

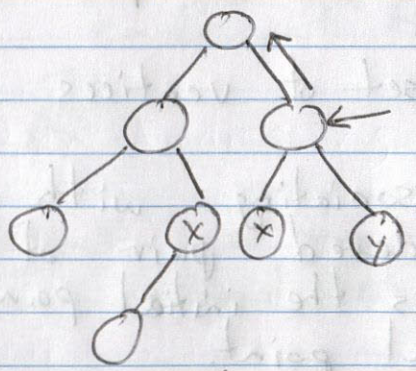
h, h+1k, h+2k

11/6/14 - Thursday

I. Computer Architecture
- Quiz!

II. American History I

III. Data Structures



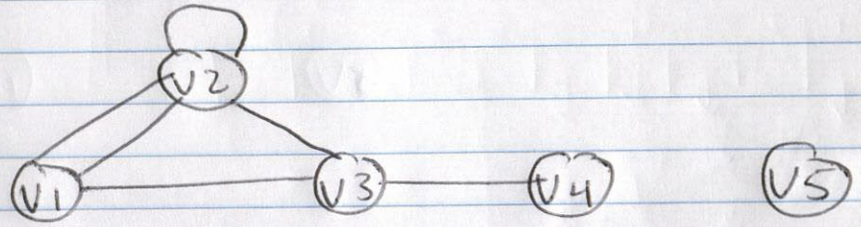
- successor / predecessor

eg. Open Addressing

For Review 3. Exam

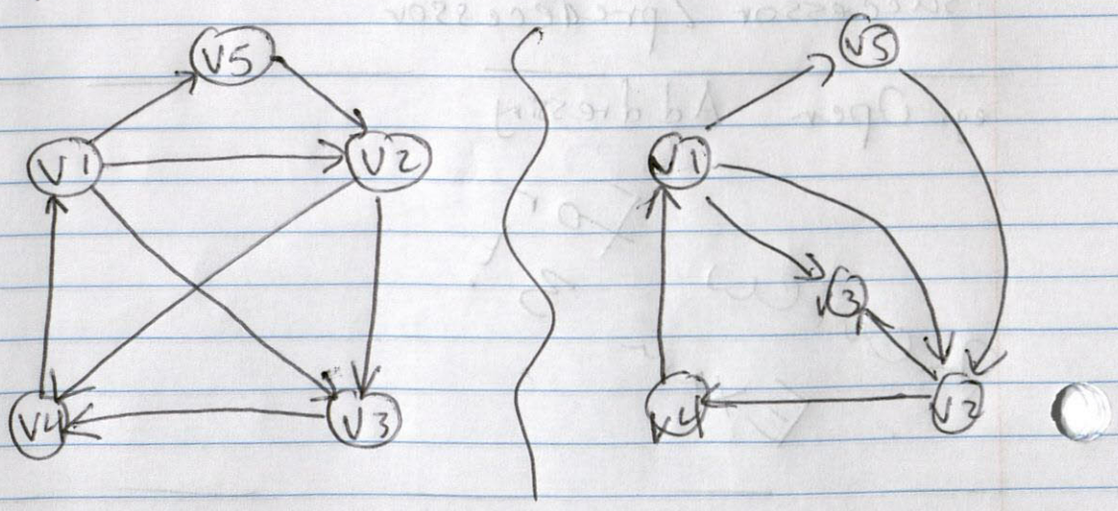
- A graph is an ordered triple (V, E, g) where:
  V - a non-empty set of vertices
  E - a set of edges
  g - a function associating with each edge an unordered pair vertices, called endpoints

eg.



- Directed Graph (digraph):
  A graph, G, is an ordered triple (V, A, g) where
  V = a non-empty set of vertices
  A = a set of Fares
  g = a function associating with each arc, a, an ordered pair of vertices (x, y) where x is the initial point and y is the terminal point.

eg.

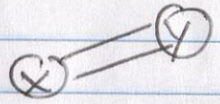


order of a graph: the number of vertices in the graph

size of a graph: the number of edges (arcs) in the graph.

adjacent vertices: vertices are said to be adjacent when an edge or arc exists between them.

parallel edges: when two or more edges (arcs) exist between the same pair of vertices.

eg: 

loop: an edge (or arc) which begins and ends at the same vertex.

eg: 

multigraph: a graph with parallel edges (or arcs) and/or loops

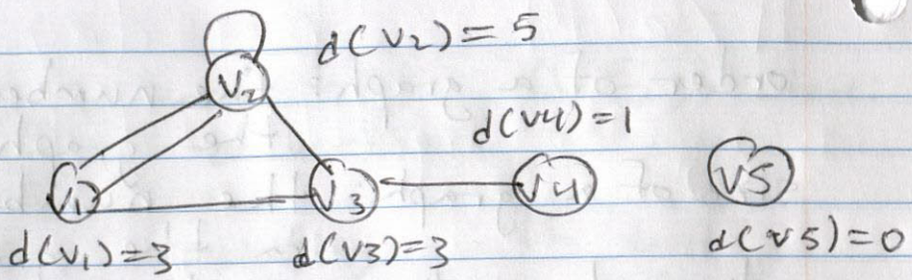
simple graph: no parallel edges and no loops.
↳ default graph

degree of vertex: the number of edges incident with that vertex

In-degree of a vertex: the number of edges incident to the vertex.

Out-degree of a vertex: the number of edges incident From the vertex.

eg. Degrees



eg.