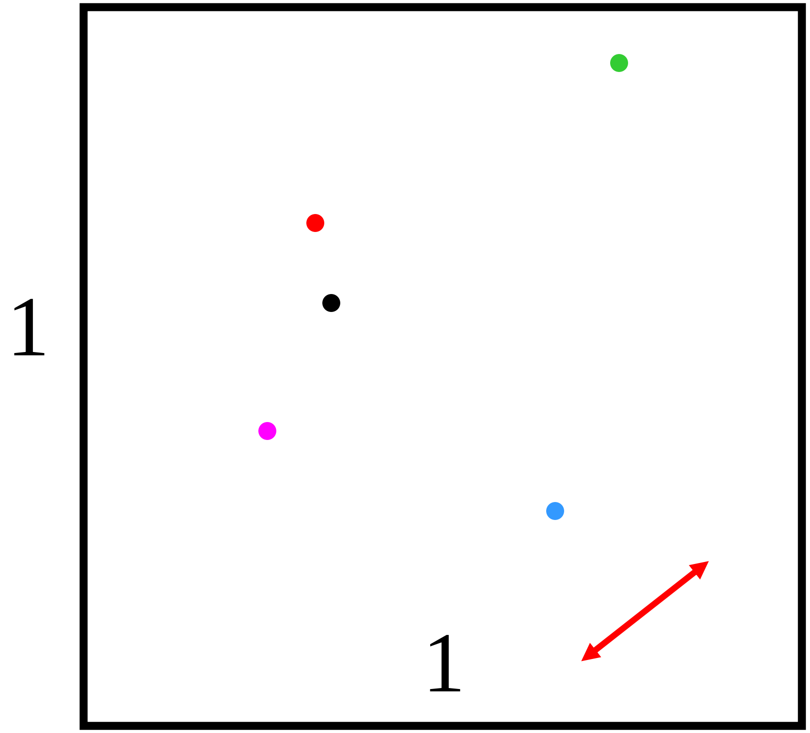
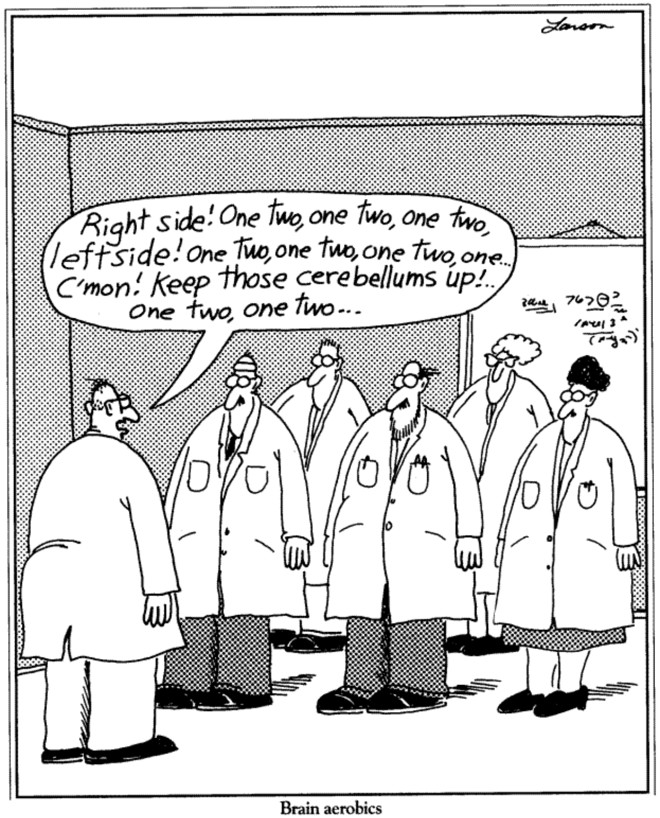
Problem: Given any five points in/on the unit square, is there always a pair with distance $\leq \frac{1}{\sqrt{2}}$?

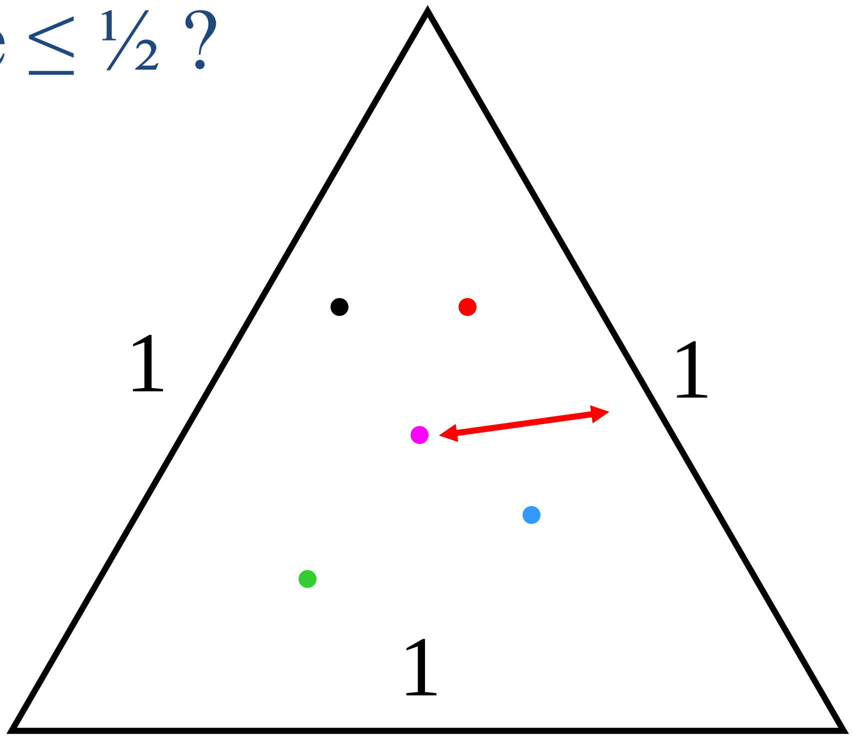


- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

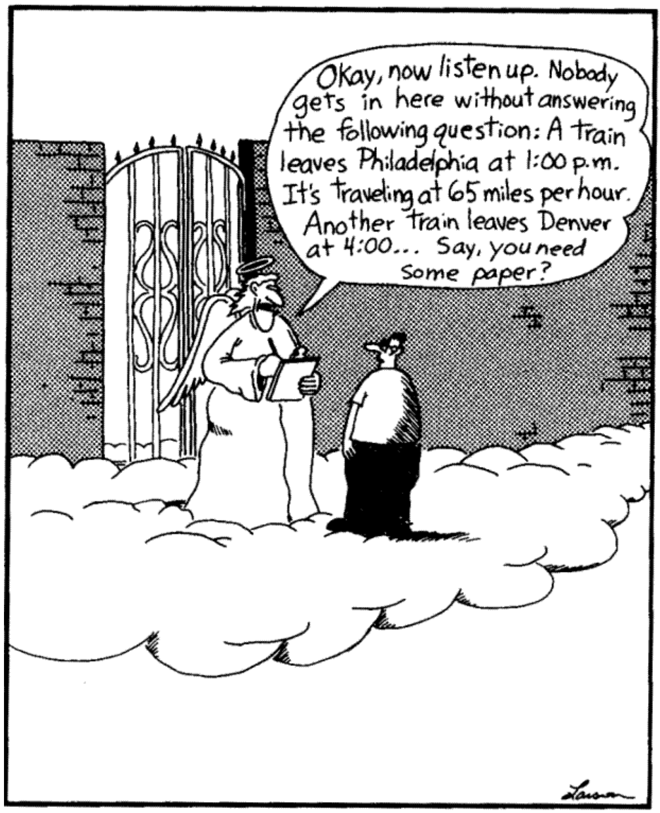


Brain aerobics

Problem: Given any five points in/on the unit equilateral triangle, is there always a pair with distance $\leq$ ½ ?

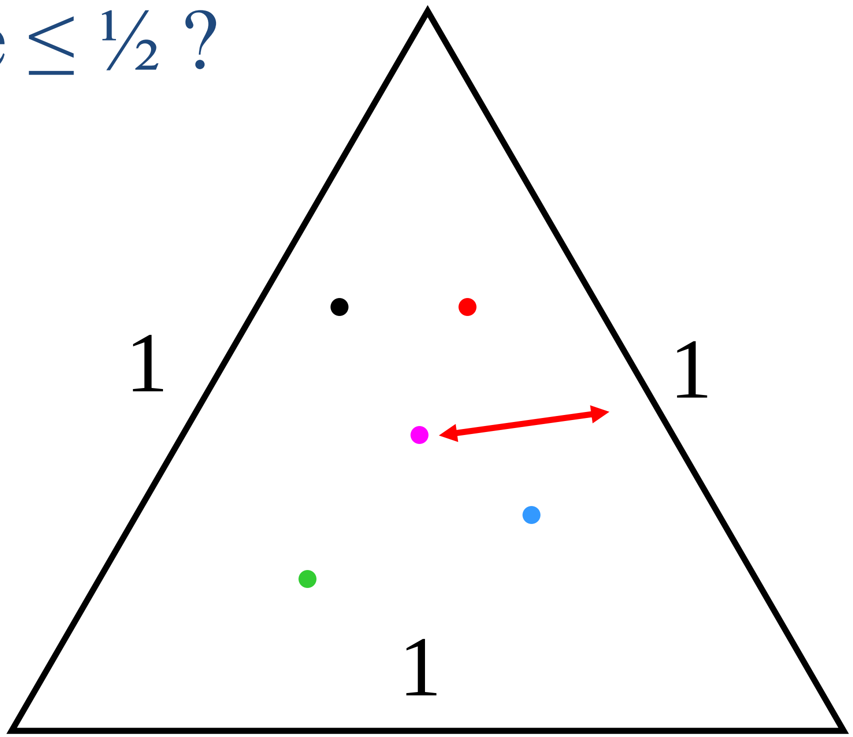


- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

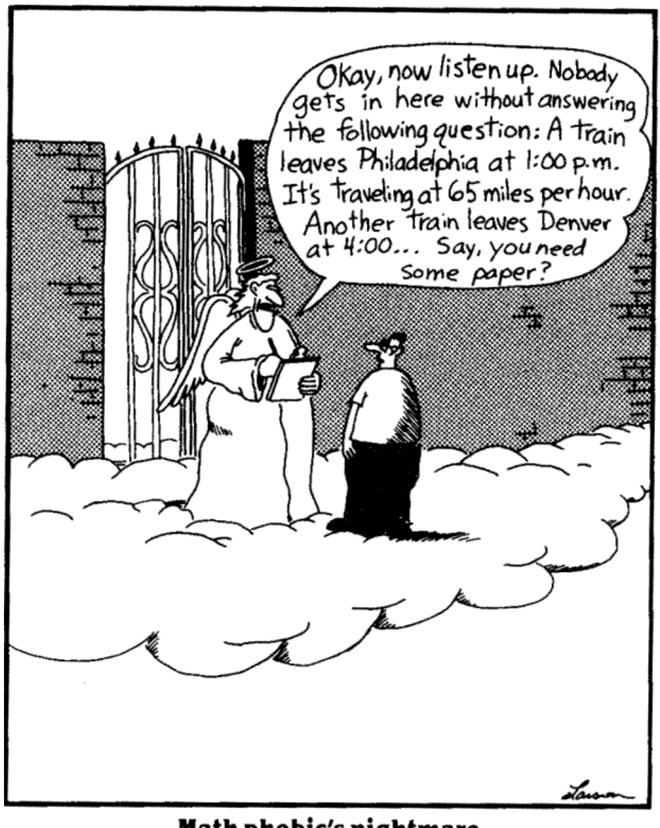


Math phobic's nightmare

Problem: Given any five points in/on the unit equilateral triangle, is there always a pair with distance $\leq \frac{1}{2}$?

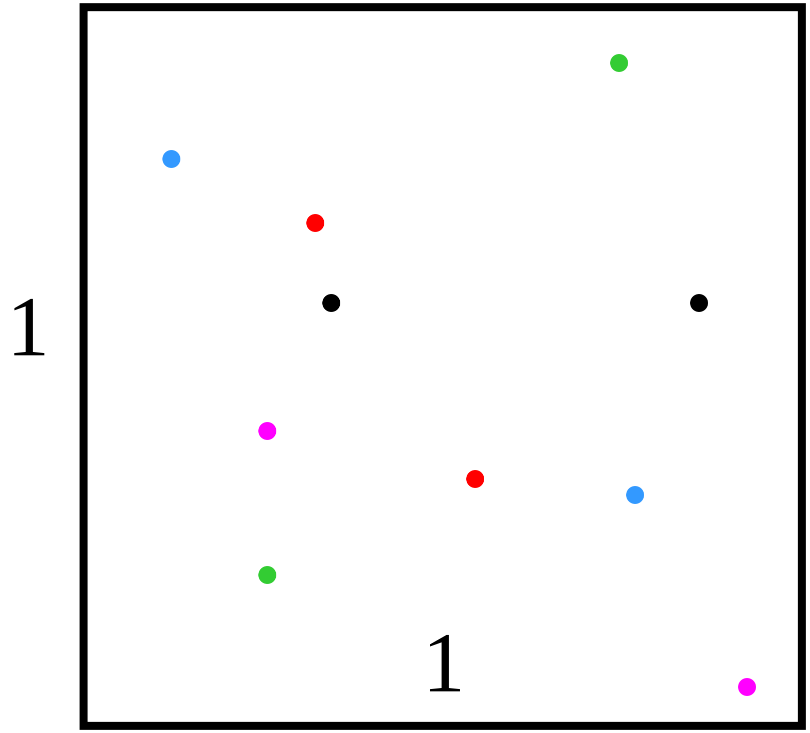


- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

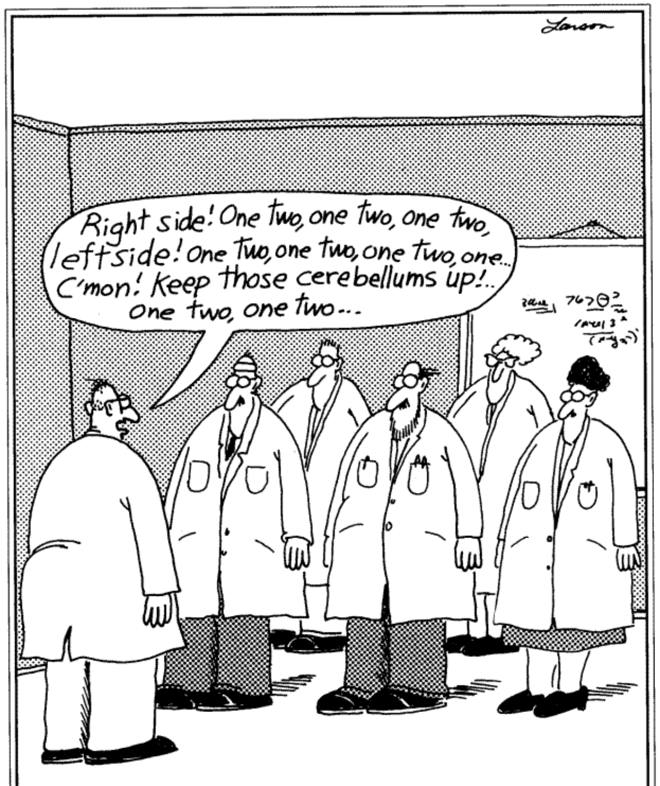


Math phobic's nightmare

Problem: Given any ten points in/on the unit square, what is the maximum pairwise distance?



- What approaches fail?
- What techniques work and why?
- Lessons and generalizations



Brain aerobics

Problem: Solve the following equation for X:

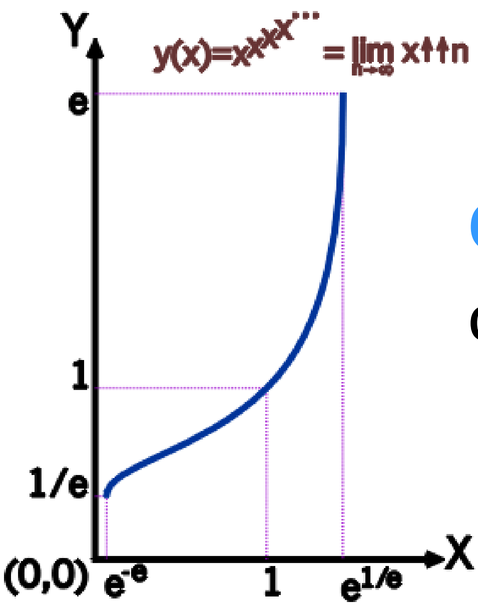
$$X^{X^{X^{X^{\cdot^{\cdot^{\cdot}}}}}} = 2 \qquad \Rightarrow X^2 = 2 \qquad \Rightarrow X = \sqrt{2}$$

$$X^{X^{X^{\cdot^{\cdot^{\cdot}}}}} = 2$$

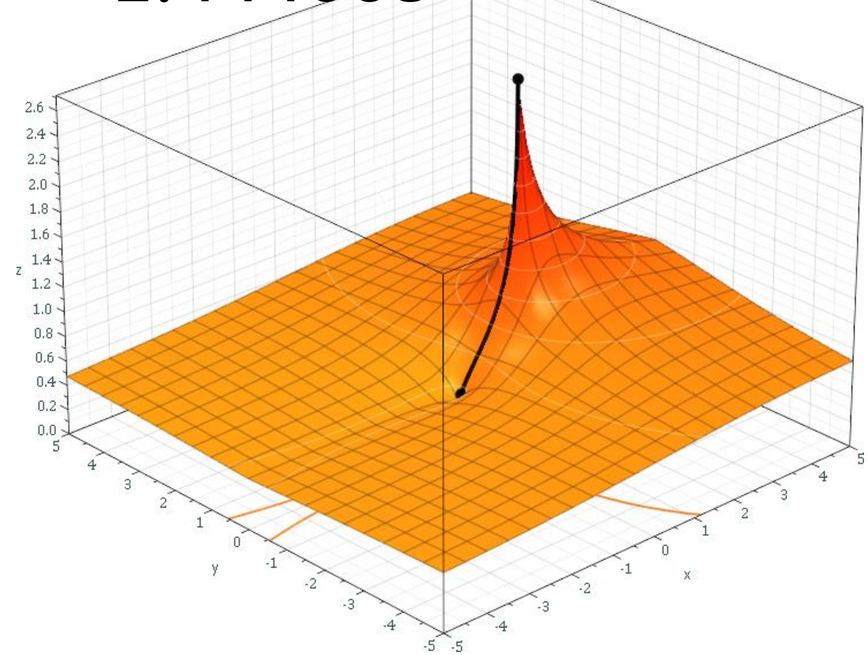
where the stack of exponentiated x's extends forever.

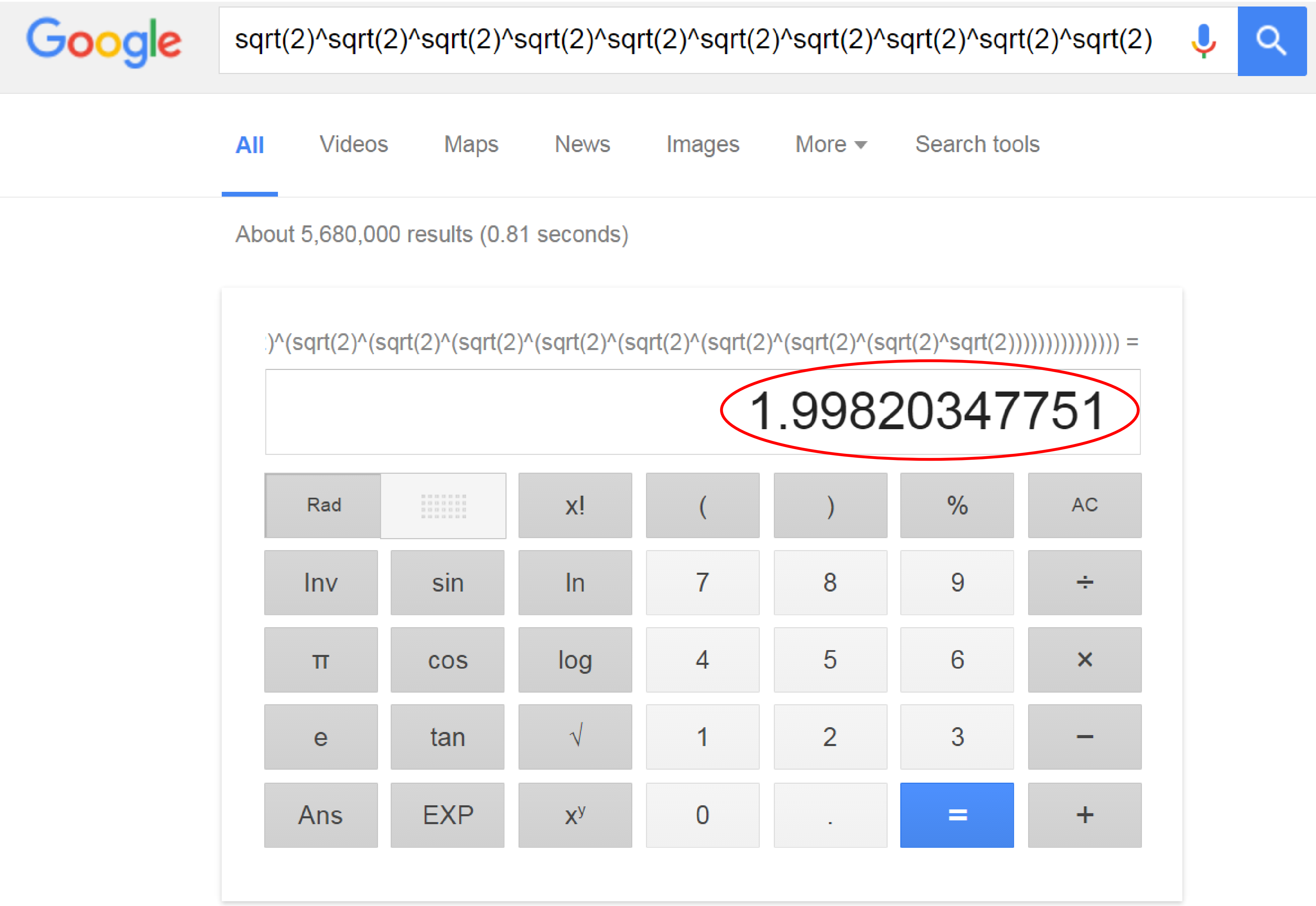
This "power tower" converges for:

$$0.065988 \approx e^{-e} < X < e^{1/e} \approx 1.444668$$



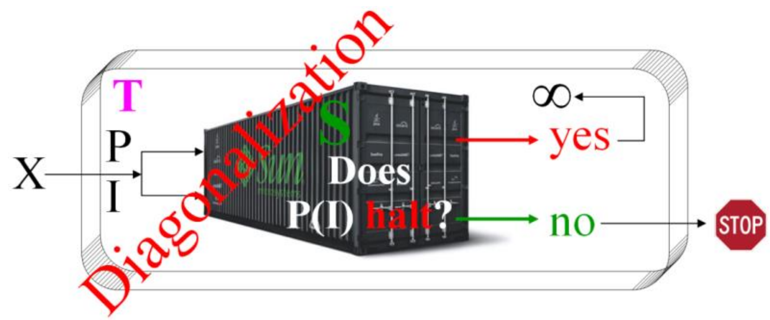
Generalization to complex numbers:



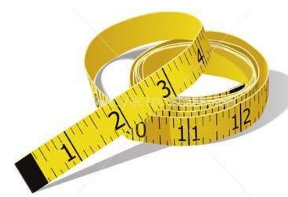

Google
sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)
All
Videos
Maps
News
Images
More
Search tools
About 5,680,000 results (0.81 seconds)
)^(sqrt(2)^(sqrt(2)^(sqrt(2)^(sqrt(2)^(sqrt(2)^(sqrt(2)^(sqrt(2)^(sqrt(2)^sqrt(2))))))))))))) =
1.99820347751
Rad
x!
(
)
%
AC
Inv
sin
ln
7
8
9
÷
π
cos
log
4
5
6
×
e
tan
√
1
2
3
−
Ans
EXP
xʸ
0
.
=
+

Algorithms

1. Existence



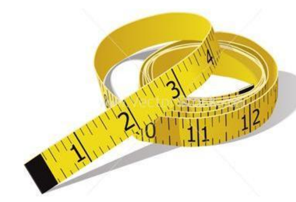
2. Efficiency
   - Time
   - Space

Worst case behavior analysis
as a function of input size

Asymptotic growth: $O \quad \Omega \quad \Theta \quad o \quad \omega$

Upper Bounds



Definition: $f(n) = O(g(n))$

$$\Leftrightarrow \exists\ c,k > 0 \ni 0 \le f(n) \le c \cdot g(n) \quad \forall\ n > k$$

$$\Leftrightarrow \lim_{n \to \infty} f(n) / g(n) \text{ exists}$$

$$O(g(n)) = \{f \mid \exists\ c,k>0 \ni 0 \le f(n) \le c \cdot g(n)\ \forall\ n>k\}$$

"$f(n)$ is big-O of $g(n)$"

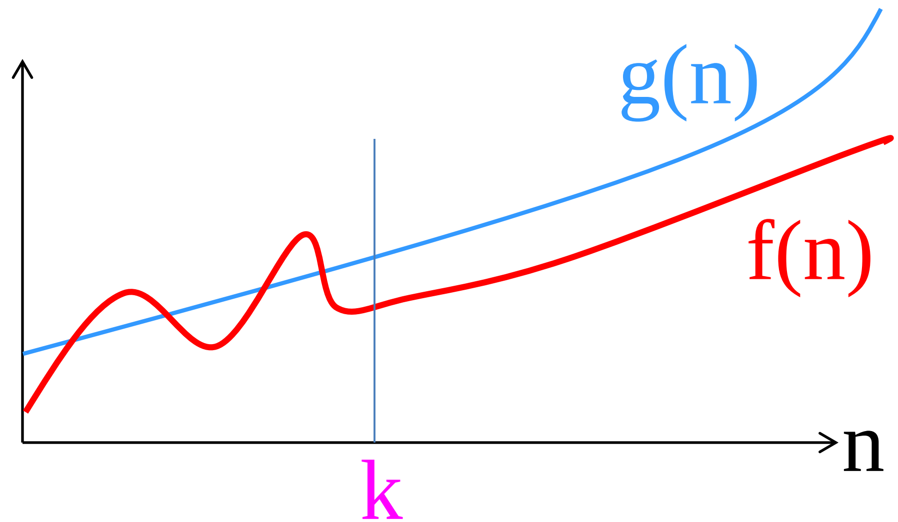
Ex: $n = O(n^2)$

$33n+17 = O(n)$

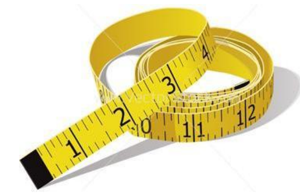
$n^8+n^7 = O(n^{12})$

$n^{100} = O(2^n)$

$213 = O(1)$



Lower Bounds



Definition: $f(n) = \Omega(g(n))$

$\Leftrightarrow \; g(n) = O(f(n))$

$\Leftrightarrow \; \lim_{n \to \infty} g(n) / f(n) \text{ exists}$

$\Omega(g(n)) = \{ f \mid \exists \; c,k>0 \ni 0 \le g(n) \le c \cdot f(n) \; \forall \; n>k \}$

"$f(n)$ is Omega of $g(n)$"

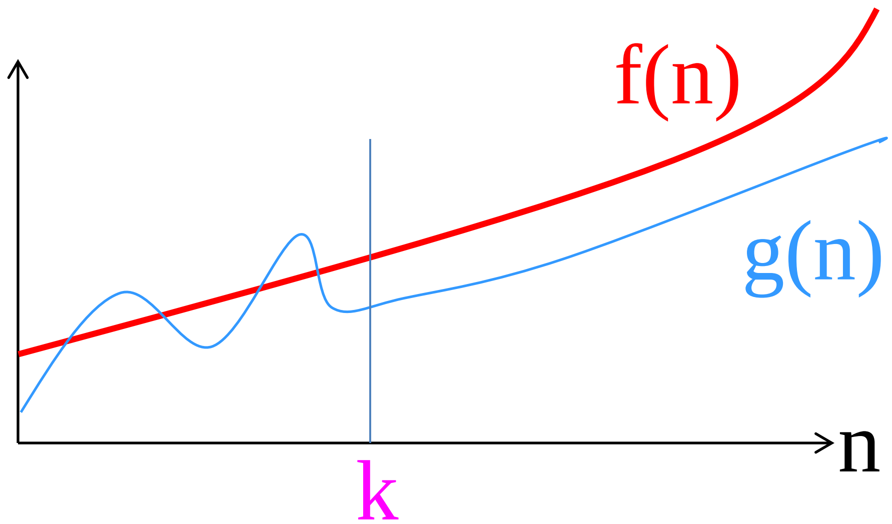
Ex: $100n = \Omega(n)$

$33n+17 = \Omega(\log n)$

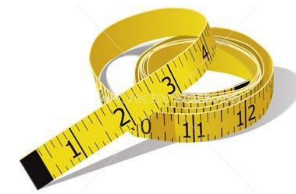
$n^8 - n^7 = \Omega(n^8)$

$213 = \Omega(1/n)$

$10^{100} = \Omega(1)$



Tight Bounds



Definition: $f(n) = \Theta(g(n))$

$\Leftrightarrow\ f(n)=O(g(n))$ and $g(n)=O(f(n))$

$\Leftrightarrow\ f(n)=O(g(n))$ and $f(n)=\Omega(g(n))$

$\Leftrightarrow\ \lim_{n\to\infty} g(n)/f(n)$ and $\lim_{n\to\infty} f(n)/g(n)$ exist

"$f(n)$ is Theta of $g(n)$"

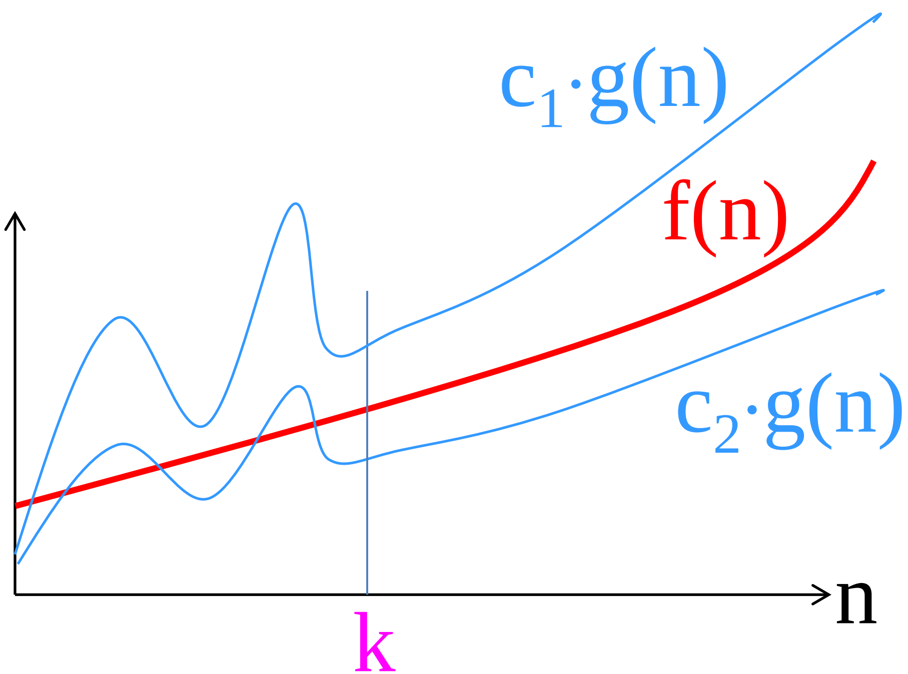
Ex: $99n = \Theta(n)$

$n + \log n = \Theta(n)$

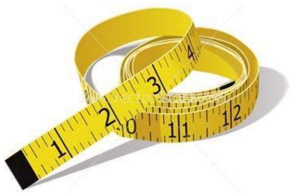
$n^8-n^7 = \Theta(n^8)$

$n^2 + \cos(n) = \Theta(n^2)$

$213 = \Theta(1)$



Loose Bounds



Definition: $f(n) = o(g_1(n))$

$\Leftrightarrow$ $f(n)=O(g_1(n))$ and $f(n)\neq\Omega(g_1(n))$

"$f(n)$ is little-o of $g_1(n)$"

Definition: $f(n) = \omega(g_2(n))$

$\Leftrightarrow$ $f(n)=\Omega(g_2(n))$ and $f(n)\neq O(g_2(n))$

"$f(n)$ is little-omega of $g_2(n)$"

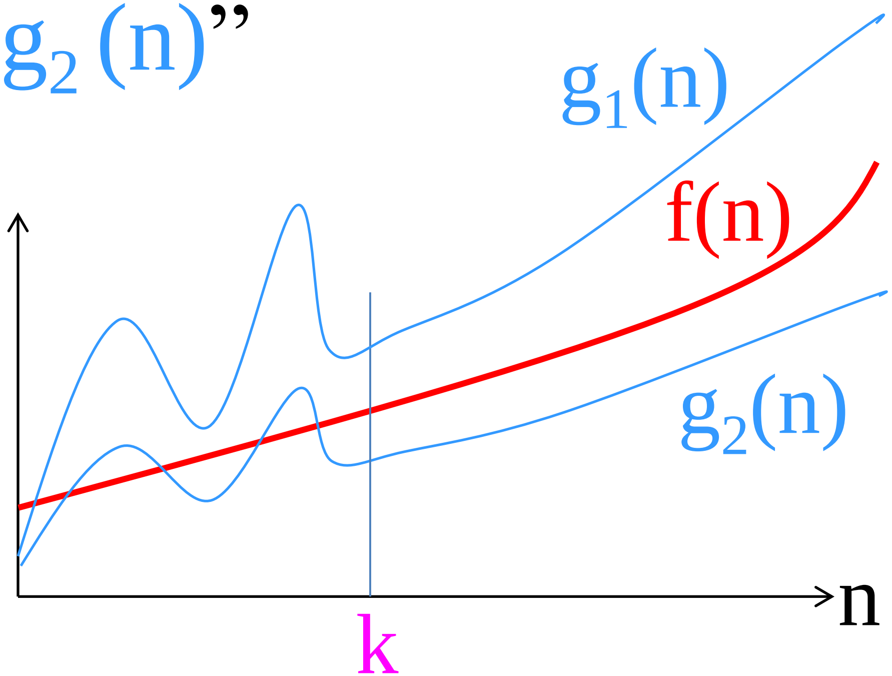
Ex: $8n = o(n \log \log n)$

$n \log n = \omega(n)$

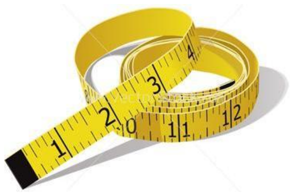
$n^6 = o(n^{6.01})$

$n^2 + \sin(n) = \omega(n)$

$213 = o(\log n)$



Growth Laws



Let $f_1(n)=O(g_1(n))$ and $f_2(n)=O(g_2(n))$

Thm: $f_1(n) + f_2(n) = O(\max(g_1(n) , g_2(n)))$

- Sequential code

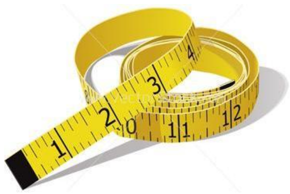
Thm: $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

- Nested loops & subroutine calls

Thm: $n^k = O(c^n) \ \ \forall\ c,k>0$

Ex: $n^{1000} = O(1.001^n)$

Solving Recurrences



$T(n) = a \cdot T(n/b) + f(n)$

$a \geq 1$, $b > 1$, and let $c = \log_b a$

Thm: $f(n) = O(n^{c-\varepsilon})$ for some $\varepsilon > 0 \Rightarrow T(n) = \Theta(n^c)$

$f(n) = \Theta(n^c) \Rightarrow T(n) = \Theta(n^c \log n)$

$f(n) = \Omega(n^{c+\varepsilon})$ some $\varepsilon > 0$ and $a \cdot f(n/b) \leq d \cdot f(n)$

for some $d < 1$ $\forall\ n > n_0 \Rightarrow T(n) = \Theta(f(n))$

Ex: $T(n) = 2T(n/2) + n \Rightarrow T(n) = \Theta(n \log n)$

$T(n) = 9T(n/3) + n \Rightarrow T(n) = \Theta(n^2)$

$T(n) = T(2n/3) + 1 \Rightarrow T(n) = \Theta(\log n)$

Stirling's Formula

Factorial: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-2) \cdot (n-1) \cdot n$

Theorem: $n! = \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \cdot \left(1 + \Theta\left(\frac{1}{n}\right)\right)$

where e is Euler's constant = 2.71828…

Theorem: $n! \approx \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n$

Corollary: $\log(n!) \approx \log\left(\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n\right) = n \cdot \log\left(\frac{n}{e}\right) + \frac{\log(2\pi n)}{2} = O(n \log n)$

$\log(n!) = O(n \log n)$

- Useful in analyses and bounds

Data Structures

Data Structures

- Techniques for organizing information effectively
- Allowed operations:
  - Initialize
  - Insert
  - Delete
  - Search
  - Min/max
  - Successor
  - Predecessor
  - Merge
  - Split
  - Revert

Data Structures

Primitive types:

1. Boolean
2. Character
3. Floating-point
4. Double
5. Integer
6. Enumerated type

Abstract data types:

7. Array
8. Container
9. Map
10. Associative array
11. Dictionary
12. Multimap
13. List
14. Set
15. Multiset / Bag
16. Priority queue
17. Queue
18. Double-ended queue
19. Stack
20. String
21. Tree
22. Graph

Composite types:

23. Array
24. Record
25. Union
26. Tagged union

Arrays:

27. Bit array
28. Bit field
29. Bitboard
30. Bitmap
31. Circular buffer
32. Control table
33. Image
34. Dynamic array
35. Gap buffer
36. Hashed array tree
37. Heightmap
38. Lookup table
39. Matrix
40. Parallel array
41. Sorted array
42. Sparse array
43. Sparse matrix
44. Iliffe vector
45. Variable-length array

Lists:

46. Doubly linked list
47. Array list
48. Linked list
49. Self-organizing list
50. Skip list
51. Unrolled linked list
52. VList
53. Xor linked list
54. Zipper
55. Doubly connected edge list
56. Difference list
57. Free list

Binary trees:

58. AA tree
59. AVL tree
60. Binary search tree
61. Binary tree
62. Cartesian tree
63. Order statistic tree
64. Pagoda
65. Randomized binary search tree
66. Red-black tree
67. Rope

Data Structures

Binary trees (continued):

68. Scapegoat tree
69. Self-balancing search tree
70. Splay tree
71. T-tree
72. Tango tree
73. Threaded binary tree
74. Top tree
75. Treap
76. Weight-balanced tree
77. Binary data structure

Trees:

78. Trie
79. Radix tree
80. Suffix tree
81. Suffix array
82. Compressed suffix array
83. FM-index
84. Generalised suffix tree
85. B-trie
86. Judy array
87. X-fast trie
88. Y-fast trie
89. Ctrie

B-trees:

90. B-tree
91. B+ tree
92. B*-tree
93. B sharp tree
94. Dancing tree
95. 2-3 tree
96. 2-3-4 tree
97. Queap
98. Fusion tree
99. Bx-tree
100. AList

Heaps:

101. Heap
102. Binary heap
103. Weak heap
104. Binomial heap
105. Fibonacci heap
106. AF-heap
107. Leonardo Heap
108. 2-3 heap
109. Soft heap
110. Pairing heap
111. Leftist heap
112. Treap
113. Beap
114. Skew heap
115. Ternary heap
116. D-ary heap
117. Brodal queue

Multiway trees:

118. Ternary tree
119. K-ary tree
120. And–or tree
121. (a,b)-tree
122. Link/cut tree
123. SPQR-tree
124. Spaghetti stack
125. Disjoint-set data structure
126. Fusion tree
127. Enfilade
128. Exponential tree
129. Fenwick tree
130. Van Emde Boas tree
131. Rose tree

Space-partitioning trees:

132. Segment tree
133. Interval tree
134. Range tree

Data Structures

Space-partitioning trees (cont):

135. Bin
136. Kd-tree
137. Implicit kd-tree
138. Min/max kd-tree
139. Adaptive k-d tree
140. Quadtree
141. Octree
142. Linear octree
143. Z-order
144. UB-tree
145. R-tree
146. R+ tree
147. R* tree
148. Hilbert R-tree
149. X-tree
150. Metric tree
151. Cover tree
152. M-tree
153. VP-tree
154. BK-tree
155. Bounding interval hierarchy
156. BSP tree
157. Rapidly exploring random tree

Application-specific trees:

158. Abstract syntax tree
159. Parse tree
160. Decision tree
161. Alternating decision tree
162. Minimax tree
163. Expectiminimax tree
164. Finger tree
165. Expression tree
166. Log-structured merge-tree

Hashes:

167. Bloom filter
168. Count-Min sketch
169. Distributed hash table
170. Double Hashing
171. Dynamic perfect hash table
172. Hash array mapped trie
173. Hash list
174. Hash table
175. Hash tree
176. Hash trie
177. Koorde
178. Prefix hash tree
179. Rolling hash
180. MinHash
181. Quotient filter
182. Ctrie

Graphs:

183. Graph
184. Adjacency list
185. Adjacency matrix
186. Graph-structured stack
187. Scene graph
188. Binary decision diagram
189. 0-suppressed decision diagram
190. And-inverter graph
191. Directed graph
192. Directed acyclic graph
193. Propositional dir. acyclic graph
194. Multigraph
195. Hypergraph

Other:

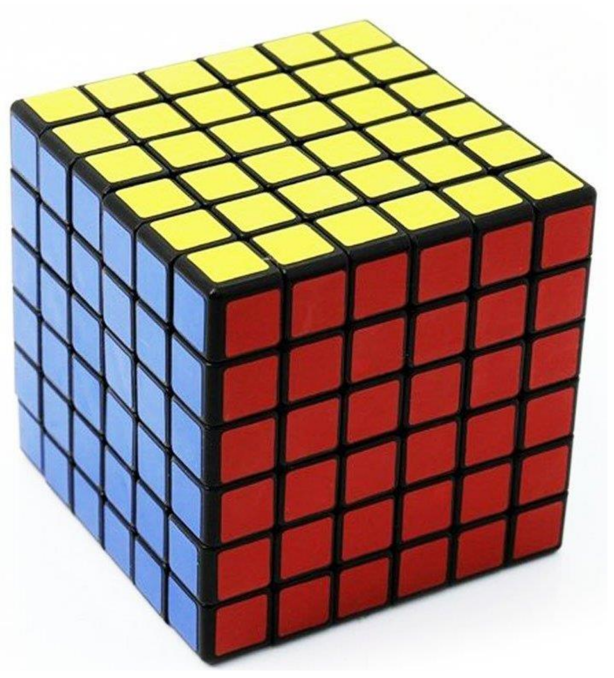
196. Lightmap
197. Winged edge
198. Doubly connected edge list
199. Quad-edge
200. Routing table
201. Symbol table

Arrays

- Sequence of "indexible" locations

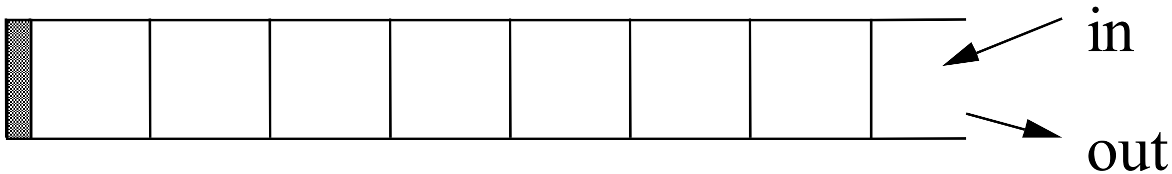
1	2	3	4	5	6	7	. . .

- Unordered:
  - O(1) to add
  - O(n) to search / delete
  - O(n) for min / max
- Ordered:
  - O(n) to add / delete
  - O(log n) to (binary) search
  - O(1) for min / max



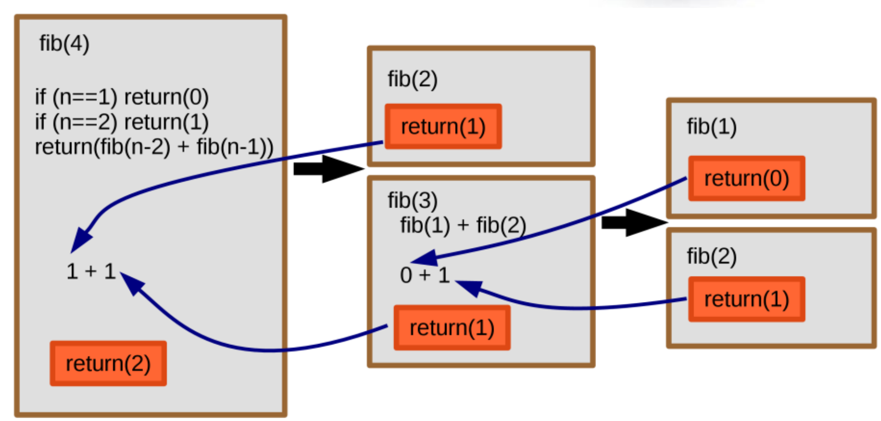
Stacks

- LIFO (Last-In First-Out)



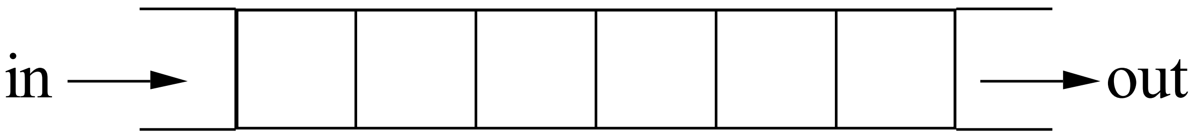
- Operations: push / pop O(1) each
- Can not access “middle”
- Analogy: trays/plates at cafeteria

- Applications:
  - Recursion
  - Compiling / parsing
  - Dynamic binding
  - Web surfing

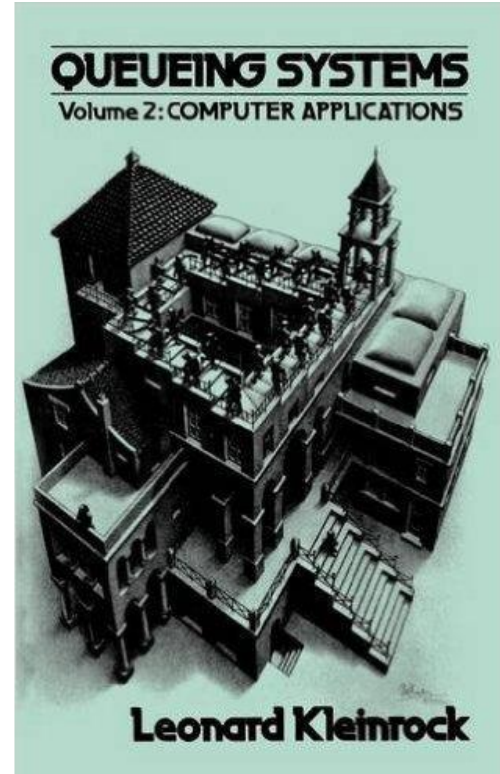


Queues

- FIFO (First-In First-Out)

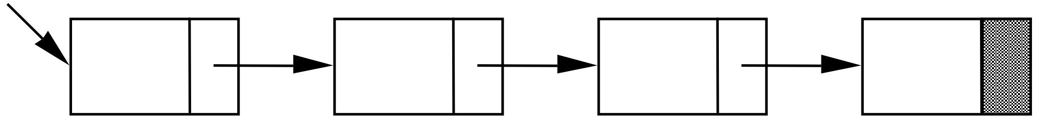


- Operations: push / pop O(1) each
- Can not access "middle"
- Analogy: line at the store
- Applications:
    - Simulations
    - Scheduling
    - Networks
    - Operating systems

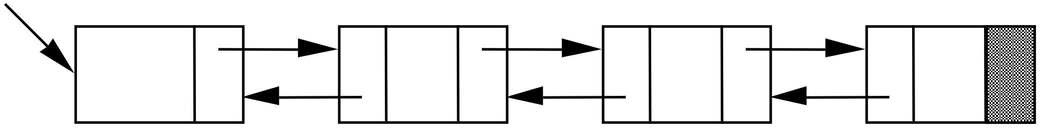


Linked Lists

- Successor / predecessor pointers
- Types:
  - Single linked



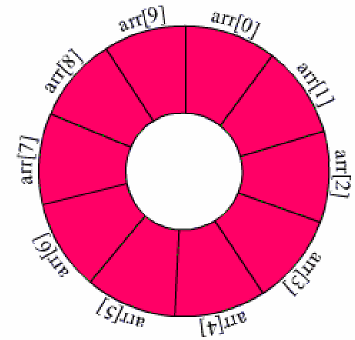
  - Double linked



  - Circular
- Operations:
  - Add: O(1) time
  - Search: O(n) time
  - Delete: O(1) time (given pointer)

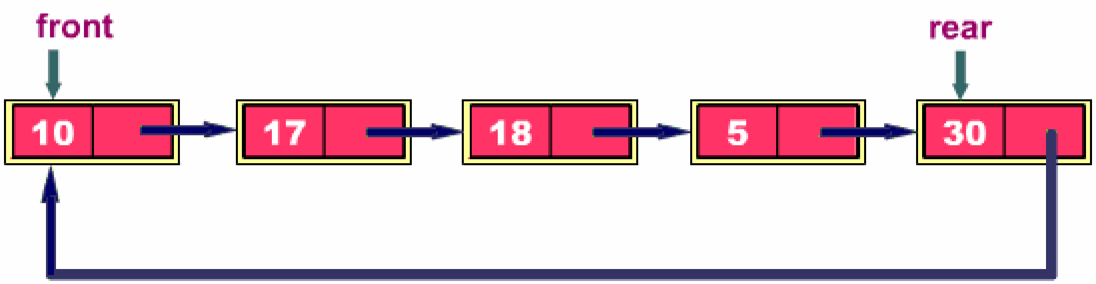
Building a linked list

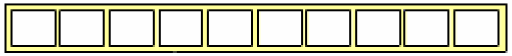
Building of Circular Queue



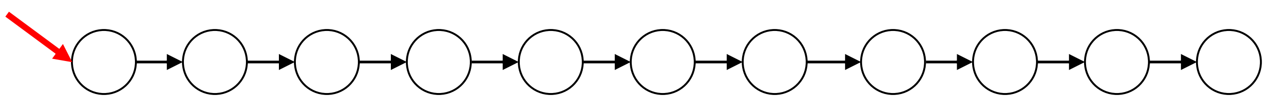
Circular queue as a linked list

Deletion of a node

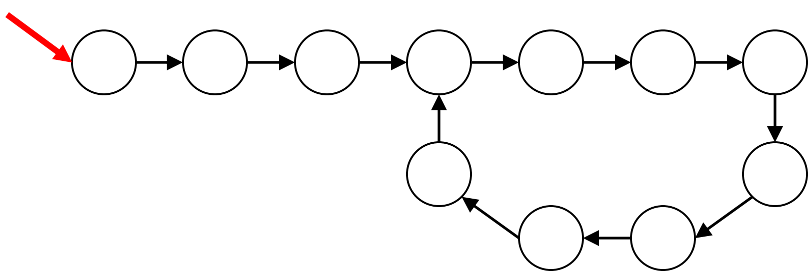




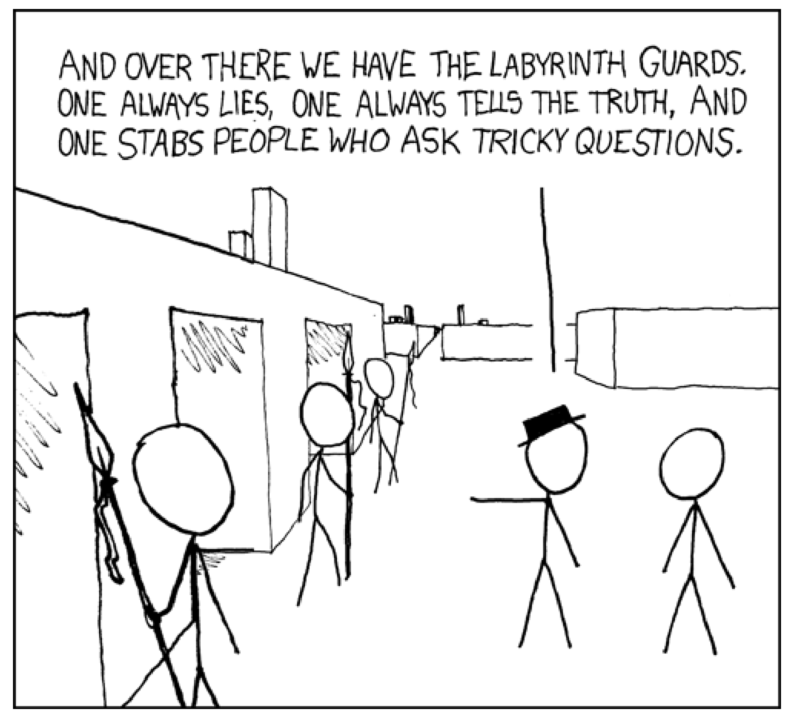
Extra Credit Problem: Given a pointer to a read-only (unmodefiable) linked list containing an unknown number of nodes n, devise an O(n)-time and O(1) space algorithm that determines whether that list contains a cycle.



or

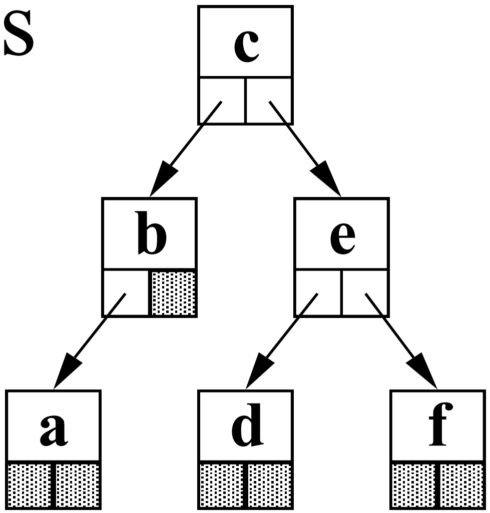


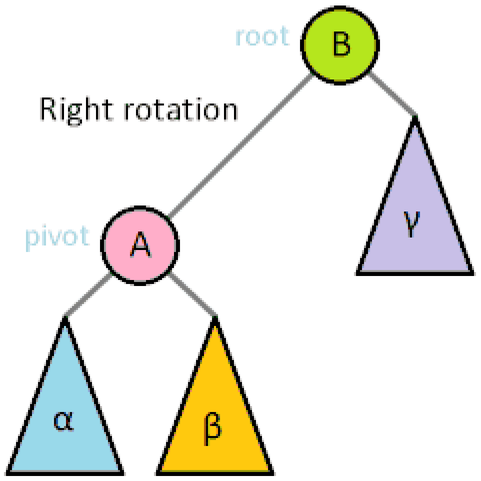
- What approaches fail?
- What techniques work and why?
- Lessons and generalizations



Trees

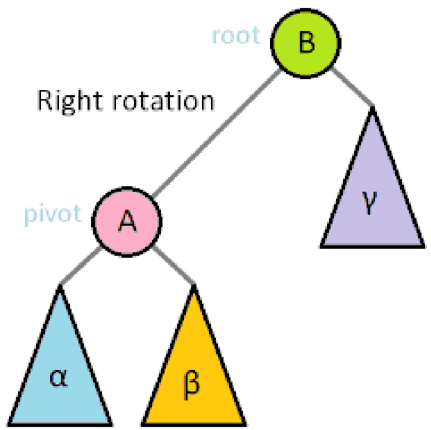
- Parent/children pointers
- Binary / N-ary
- Ordered / unordered
- Height-balanced:
  - B-trees
  - AVL trees
  - Red-black trees
  - 2-3 trees
  - add / delete / search in O(log n) time





B-Trees

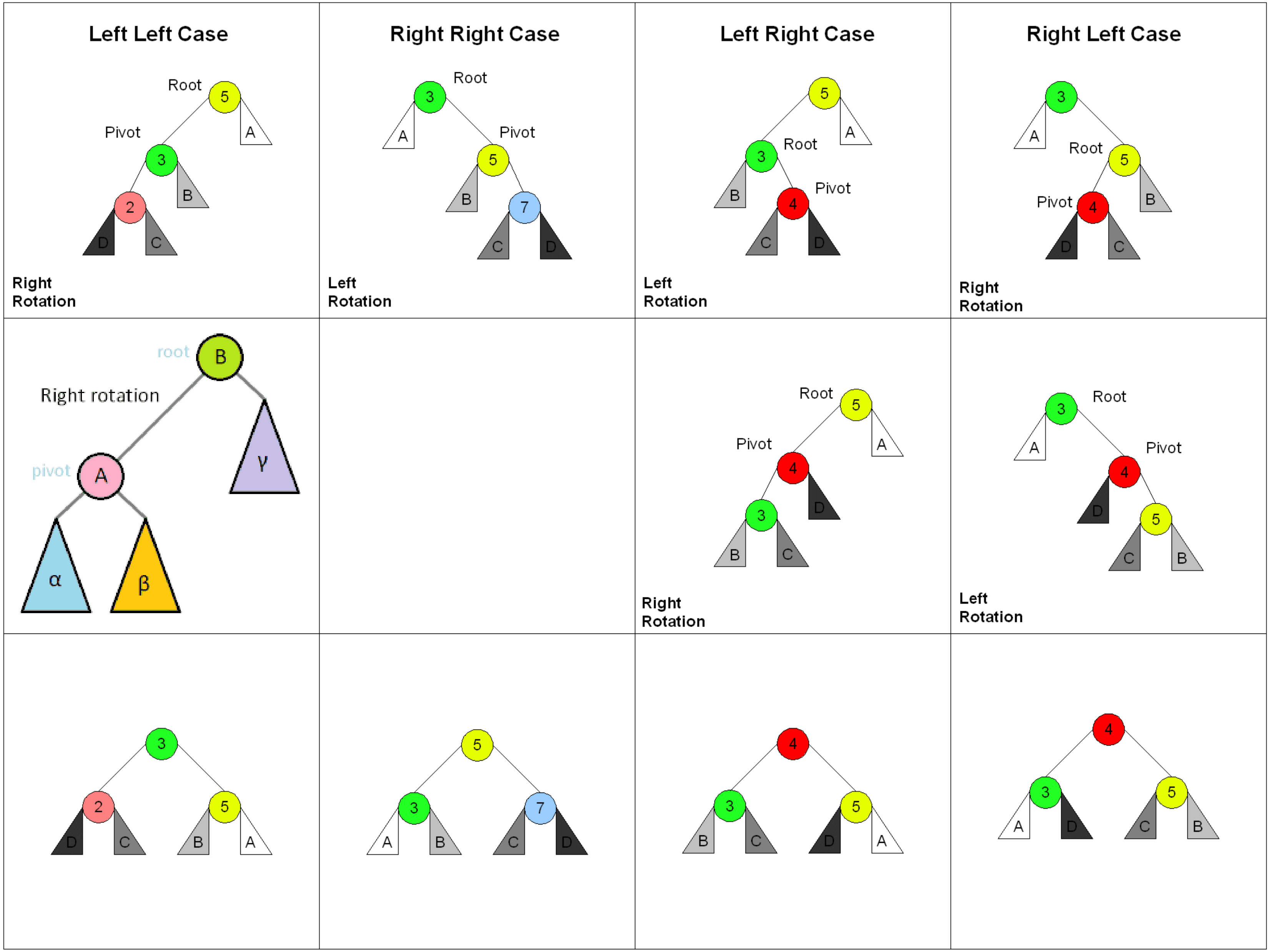
- Multi-rotations occur infrequently
- Rotations don't propagate far
- Larger tree $\Rightarrow$ fewer rotations
- Same for other height-balanced trees
- Non-balanced search trees average O(log n) height



Height-Balanced AVL Trees

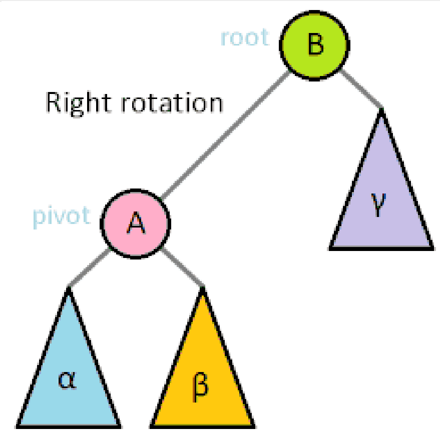
There are 4 cases in all, choosing which one is made by seeing the direction of the first 2 nodes from the unbalanced node to the newly inserted node and matching them to the top most row.

Root is the initial parent before a rotation and **Pivot** is the child to take the root's place.

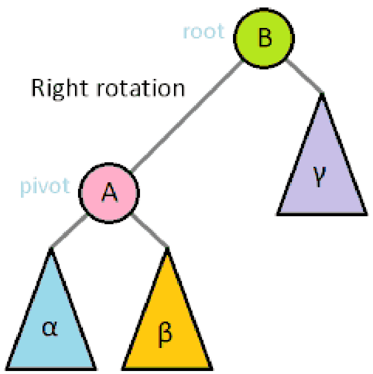


AVL Trees

- Multi-rotations occur infrequently
- Rotations don't propagate far
- Larger tree $\Rightarrow$ fewer rotations
- Same for other height-balanced trees
- Non-balanced trees average O(log n) height



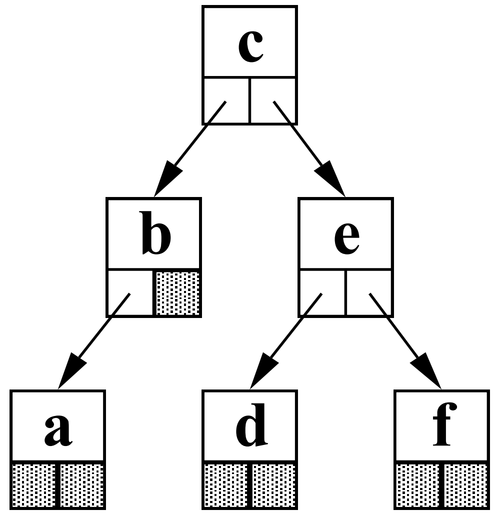
Self-Adjusting Trees

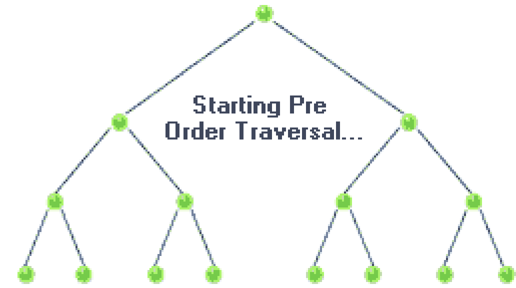


Tree Traversal

- Pre-order:
  1. process node
  2. visit children

  $\Rightarrow$ c b a e d f



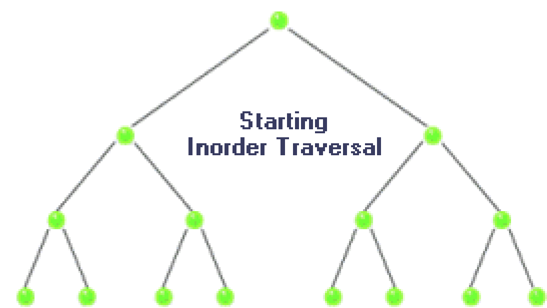


- Post-order:
  1. visit children
  2. process node

  $\Rightarrow$ a b d f e c

- In-order:
  1. visit left-child
  2. process node
  3. visit right-child

  $\Rightarrow$ a b c d e f

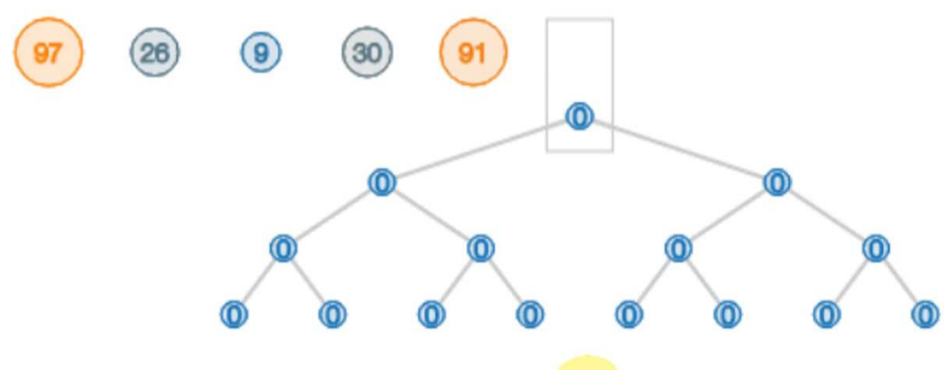


Heaps

- A tree where all of each node’s children
  have larger / smaller “keys”
- Can be implemented
  using binary tree or array

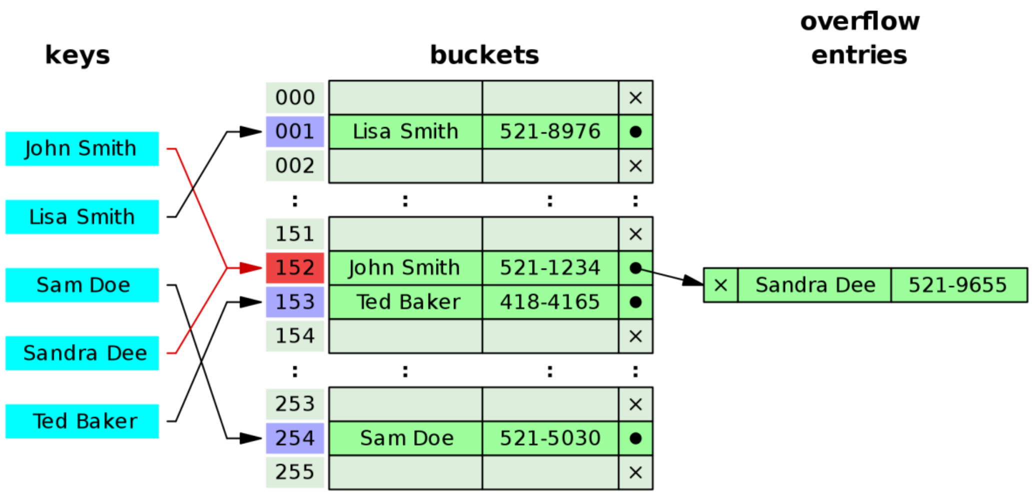
6 5 3 1 8 7 2 4

- Operations:
  - Find min/max: O(1) time
  - Add: O(log n) time
  - Delete: O(log n) time
  - Search: O(n) time



Hash Tables

- Direct access
- Hash function
- Collision resolution:
  - Chaining
  - Linear probing
  - Double hashing
- Universal hashing
- O(1) average access
- O(n) worst-case access



Q: Improve worst-case access to O(log n)?

3.4 Linear Probing Hash Table Demo

click to begin demo