```
Select(A,p,q,i)
if p=q then
   return A[p]
else
   x := Median(A,p,q)
   m := Partition(A,p,q,pivot x)
  // p ≤ m ≤ q
   if m-p = i-1 then
      return A[m]
   else if m-p > i-1 then
      return Select(A,p,m-1,i)
   else // m-p < i-1
      return Select(A,m+1,q,i-(m-p))
```

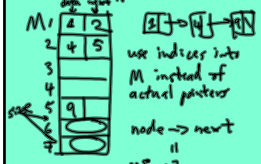
Linked structures, implemented using arrays

-Trees as linked structures

Red-Black trees

Suppose you only have simple types and arrays of simple types.

Imagine structs (records) are allowed as types.



node->next->data

M[M[node].next].data

etc.

struct {
 data;
 int next;
};

Without using struct, maintain a pair of arrays

Mdata & Mnext

Mdata: double? Mnext: int

n elements each

node -> next -> data

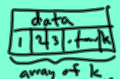
Mdata[Mnext[node]]

Trees as linked structures

Some common representations:

1. Fixed max number k of children per node. (usually k is small)
   Binary trees: k = 2.
2. Arbitrary number of children per node

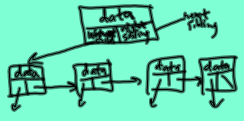
(1) Fix max k children:



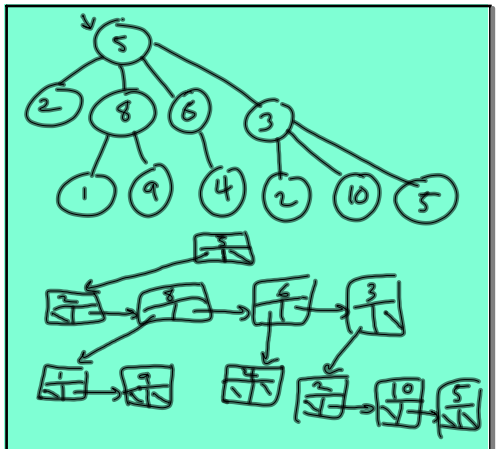
k=2:

(2) Unlimited # of children (subtrees) per node.

Typical: store children as a linked list:



Oct 4-12:51 PM



One variation: optional parent field (null iff root)

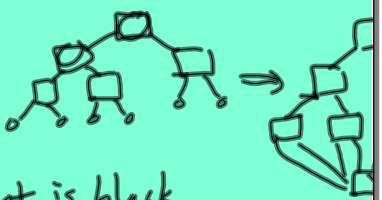
Red-Black Trees

R-B tree, AVL tree } implementations of a binary search tree that include adjustments to keep the tree balanced

[O(lg n) worst-case guarantee for all the BST operations: find, insert, remove; as well as the usual tree traversals in O(n) time.]

A red-black tree (RB tree) is a binary tree where

1. each node a color attrib. (either red or black) in addition to the other BS fields: data, left, right, par.
2. All leaves are dummy nodes (no data), and all are bla



3. Root is black.
   [empty tree: root is a leaf]

Oct 4-1:31 PM