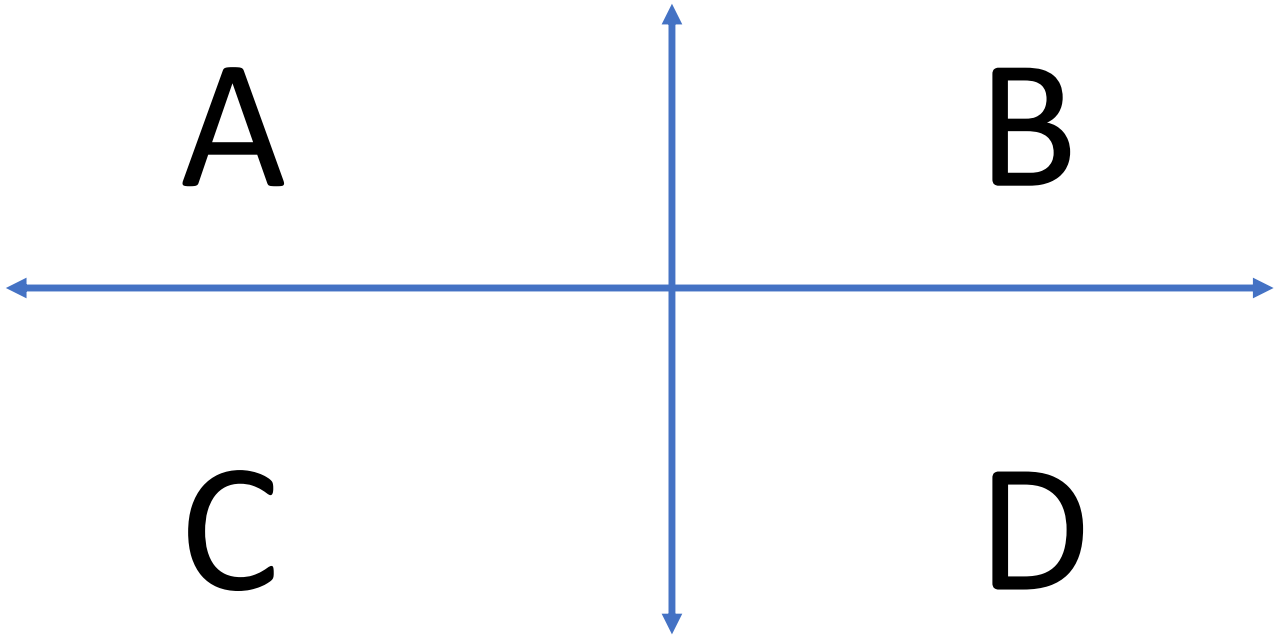
Looking forward

- The course picks up in pace and difficulty from here
- No big problems if you have learned the material to date
  - This week is easy because we know how to design recursive functions
  - Next week is a bit easier too
- Then things really pick up
  - m08 abstraction
  - m09 genrec and search
  - m10 accumulators
  - m11 graphs

Again, no big problems if you have learned the material to date

ANSWER NOW…

For problem set 4…
which quadrant most closely matches your experience?



In list of <number>:<password> lookup password

3:ilk 7:ruf 42:ily 1:abc 50:dug 10:why 27:wit 14:olp 4:dcj …

How long does it take if list has length n ?

In list of <number>:<password> lookup password

3:ilk 7:ruf 42:ily 1:abc 50:dug 10:why 27:wit 14:olp 4:dcj …

How long does it take if list has length n ?

What about in this list?

1:abc 3:ilk 4:dcj 7:ruf 10:why 14:olp 27:wit 42:ily 50:dug

Binary search tree

[from Wikipedia, edited]

From Wikipedia, the free encyclopedia

In computer science, **binary search trees** (**BST**), sometimes called **ordered** or **sorted binary trees**, are a particular type of container: a data structure that stores "items" (such as numbers, names etc.) in memory. They allow fast lookup, addition and removal of items, and can be used to implement either dynamic sets of items, or lookup tables that allow finding an item by its *key* (e.g., finding the phone number of a person by name).

Binary search trees keep their keys in sorted order, so that lookup and other operations can use the principle of binary search: when looking for a key in a tree (or a place to insert a new key), they traverse the tree from root to leaf, making comparisons to keys stored in the nodes of the tree and deciding, on the basis of the comparison, to continue searching in the left or right subtrees. On average, this means that each comparison allows the operations to skip about half of the tree, so that each lookup, insertion or deletion takes time proportional to the logarithm of the number of items stored in the tree. This is much better than the linear time required to find items by key in an

Binary search tree

Type	tree
Invented	1960
Invented by	P.F. Windley, A.D. Booth, A.J.T. Colin, and T.N. Hibbard

Time complexity in big O notation

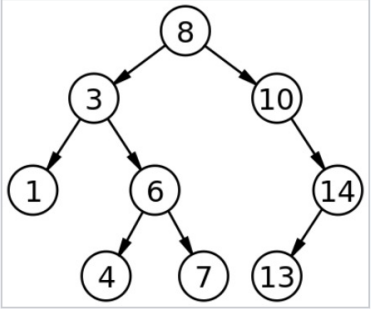
Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

Definition [edit]

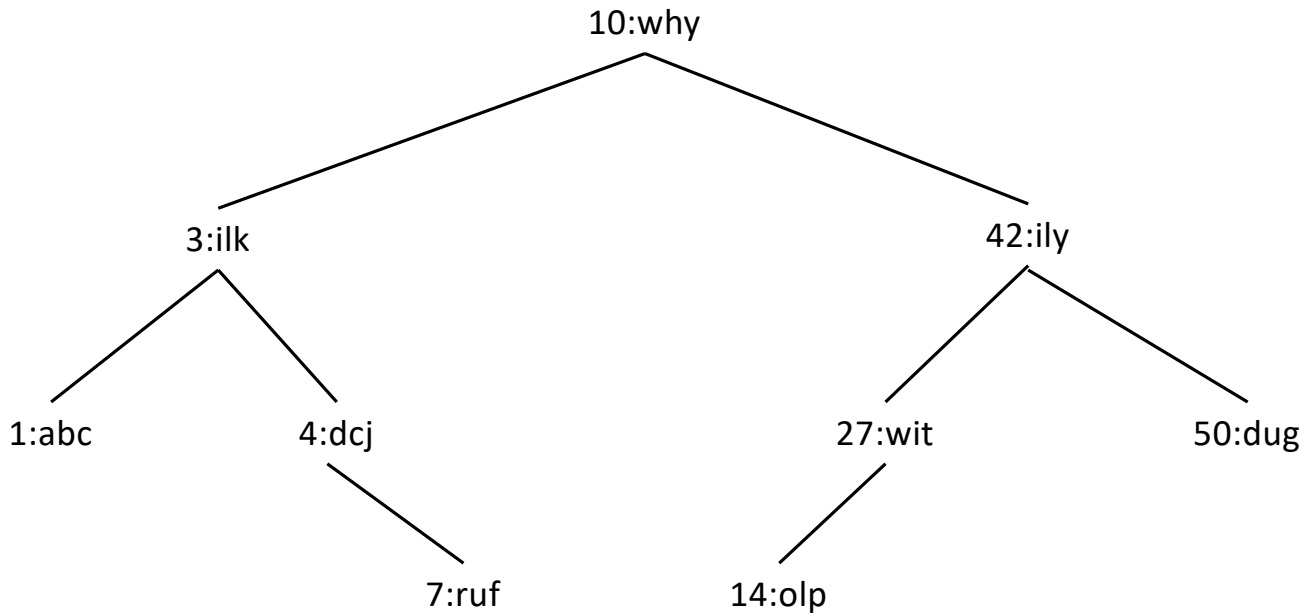
A binary search tree is a rooted binary tree, whose internal nodes each store a key (and optionally, an associated value) and each have two distinguished sub-trees, commonly denoted *left* and *right*. The tree additionally satisfies the binary search property, which states that the key in each node must be greater than or equal to any key stored in the left sub-tree, and less than or equal to any key stored in the right sub-tree.[1]:287 The leaves (final nodes) of the tree contain no key and have no structure to distinguish them from one another.

Frequently, the information represented by each node is a record rather than a single data element. However, for sequencing purposes, nodes are compared according to their keys rather than any part of their associated records. The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient; they are also easy to code.

Binary search trees are a fundamental data structure used to construct more abstract data structures such as sets, multisets, and associative arrays.

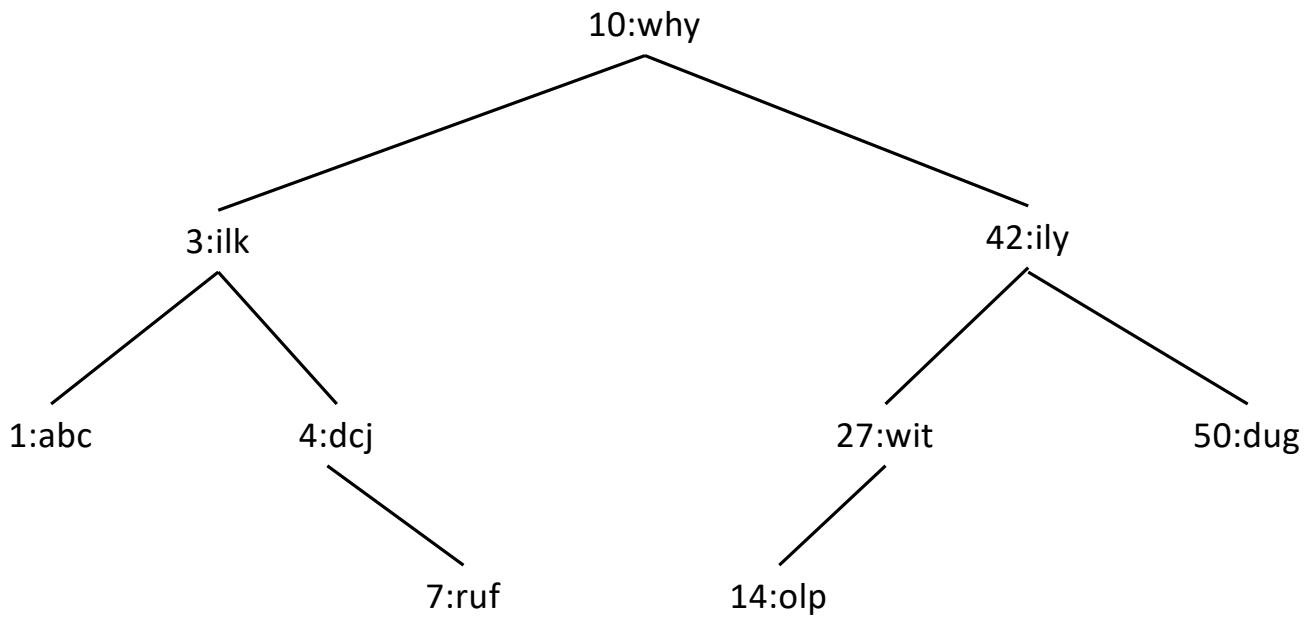


A binary search tree of size 9 and depth 3, with 8 at the root. The leaves are not drawn.


10:why
3:ilk
42:ily
1:abc
4:dcj
27:wit
50:dug
7:ruf
14:olp

compound key, value, left, right

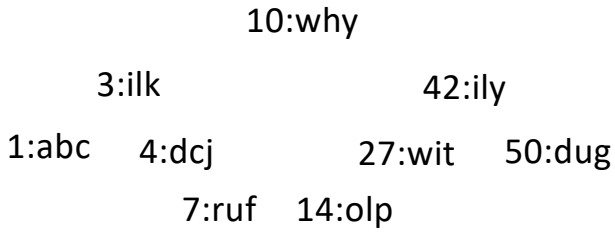
arb-sized

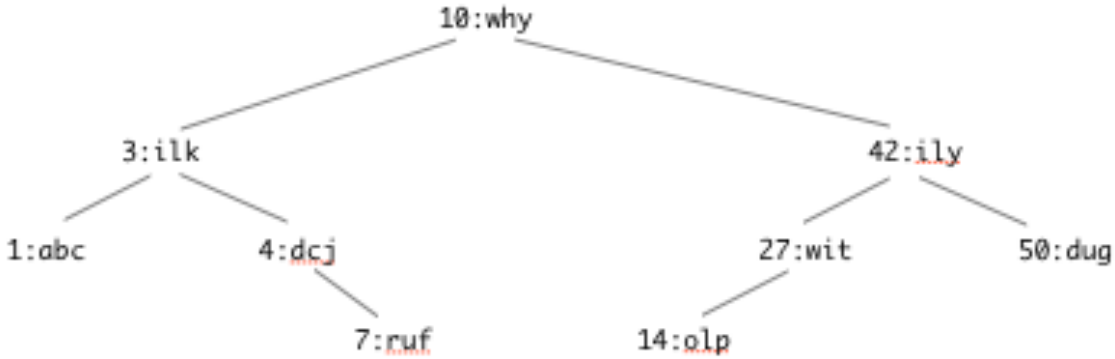


```
(@htdd BST)
(define-struct node (key val l r))
;; A BST (Binary Search Tree) is one of:
;;  - false
;;  - (make-node Integer String BST BST)
;; interp. false means empty BST
;;         key is the node key
;;         val is the node val
;;         l and r are left and right subtrees
;; CONSTRAINT: (INVARIANT) for a given node:
;;     key is > all keys in its l(eft)  child
;;     key is < all keys in its r(ight) child
;;     the same key never appears twice in the tree
```

```
(@htdd BST)
(define-struct node (key val l r))
;; A BST (Binary Search Tree) is one of:
;;  - false
;;  - (make-node Integer String BST BST)
;; interp. false means empty BST
;;         key is the node key
;;         val is the node val
;;         l and r are left and right subtrees
;; CONSTRAINT: (INVARIANT) for a given node:
;;     key is > all keys in its l(eft)  child
;;     key is < all keys in its r(ight) child
;;     the same key never appears twice in the tree


(define (fn-for-bst t)
  (cond [(false? t) (...)]
        [else
         (... (node-key t)
              (node-val t)
              (fn-for-bst (node-l t))
              (fn-for-bst (node-r t)))]))
```


10:why
3:ilk
42:ily
1:abc
4:dcj
27:wit
50:dug
7:ruf
14:olp


10:why
3:ilk
42:ily
1:abc
4:dcj
27:wit
50:dug
7:ruf
14:olp