TUTORIAL 4: ADTs, BSTs

ABSTRACT DATA TYPES (ADTs)

Fundamentally, a data type in C will consist of 2 things:

Type

Interface:

```
typedef struct node * List;
```

Implementation:

```
struct node { /* fields */ };
```

Operations

Interface:

```
List new_list(void);
```

Implementation:

```
List new_list(void) {
    /* impl. */
}
```

But: to use a data type, we really just need to know its interface.
Also: an interface can have many possible implementations.

⇒ ADT: a data type which separates the interface (constant) from the implementation (maybe changing).

How? Put the interface in .h files, implementation in .c files.

Benefits:
- easy to swap out implementations without users needing to make changes to their code (the interface is still the same!)
- safe, because users can't accidentally alter internal ADT data and potentially cause problems inside it
- generalised nature of ADTs makes them reusable for other users

BINARY SEARCH TREES (BSTs)

Motivation: searching for things in a sorted array is fast, because we can use binary search:

7

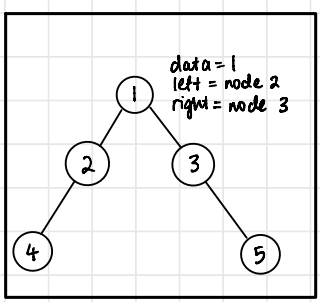
[1 2 4 6 8 9 12]

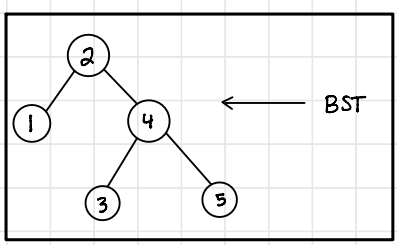
Problem: we now know sorting arrays is expensive — $\Omega(n \log n)$ at best. Inserting and removing values from a sorted array are $\Omega(n)$ as well.

Maintenance sounds like a pain. Can we do better?

Yes: think hierarchically instead of laterally!

Binary tree: structure of nodes, where each has some associated value and up to 2 children:

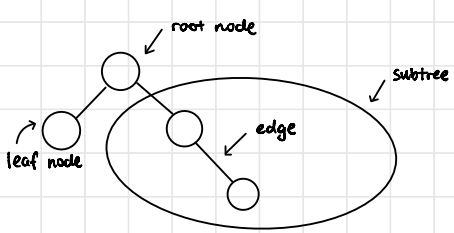




Binary search tree: all nodes have the property left data < data < right data

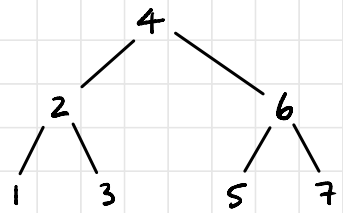
Selling point: If done properly, search and insert are both $O(\log n)$!

Terminology:



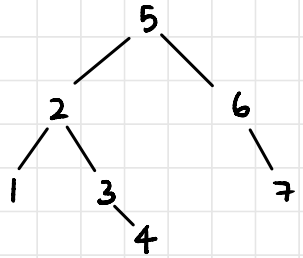
1. Insert values into the BST

(a) 4 6 5 2 1 3 7



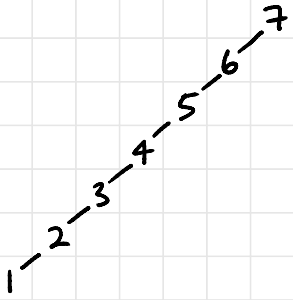
Note: This tree is balanced, i.e. has minimal height for the number of nodes
⇒ we get O(log n) search, inserts

(b) 5 2 3 4 6 1 7



This tree is less balanced, but still not that bad
⇒ search, insert are somewhere between O(log n) & O(n)

(c) 7 6 5 4 3 2 1



This is a really unbalanced tree, since it is basically a linked list (degenerate tree)
⇒ insert, search are O(n) :(