2-21-2019

typedef

qsort(....)

- function addresses
- sizeof(...)

binary search trees

typedef

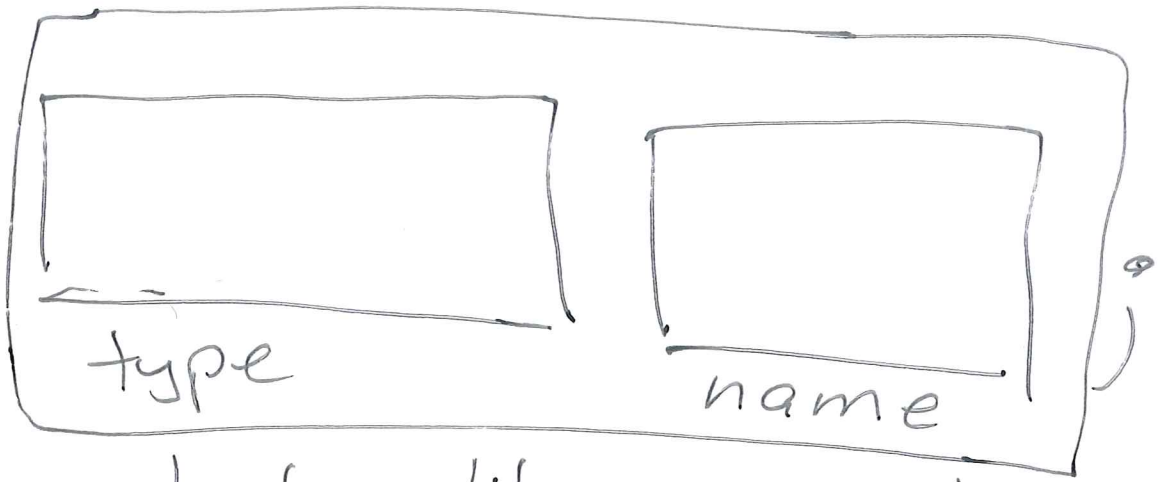
typedef `int Number;`

looks like a declaration (w/o initialize)

typedef

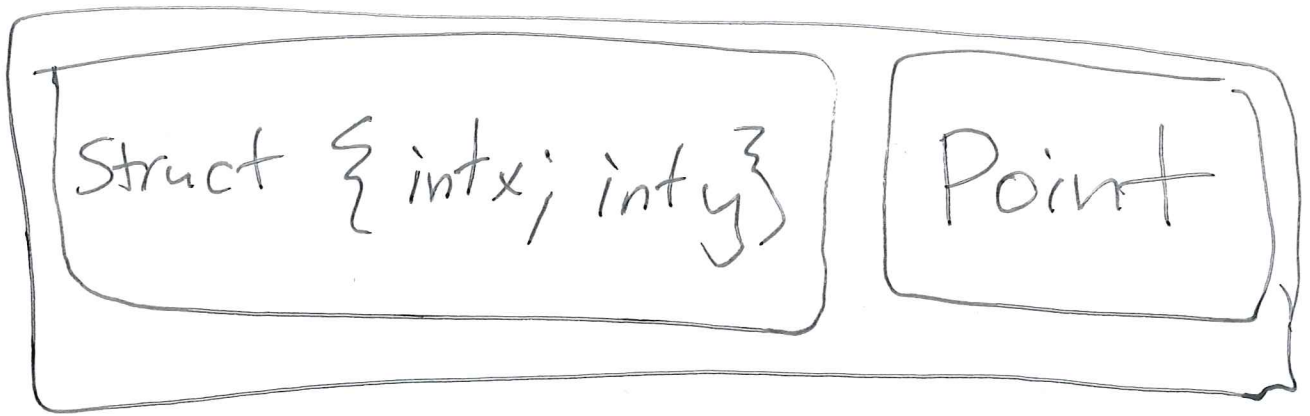
```
struct    {
    int x;
    int y;
} Point;
```

typedef



looks like an initializer

typedef



(This page is from Fall 2017)

typedef struct

```
typedef struct _Point {
    int x;
    int y;
    char* name;
} Point;
```

- `_Point` — optional
- `struct _Point` — canonical name (i.e. long name), e.g. `struct Point`
- `int x; int y; char* name;` — declarations of fields
- `Point` — new type name

Do not use sizeof(TYPE) in your code

sizeof(int*) ⇒ 8 on our platform

address

sizeof(int [4]) ⇒ size of the elements (all)

type count

array on stack

4 * 4 on our platform

```
int a[5] = { };  // 5*4
// sizeof(a) == 5*4 == 20
int* a = malloc(99 * sizeof(*a));  // 8
// sizeof(a) == 8
```

(on ecegrid)

Every subtree is also a tree.

5 3 2 4 7 6 8

empty tree

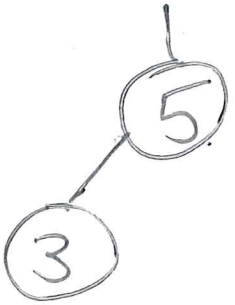
root

Insert 5

root

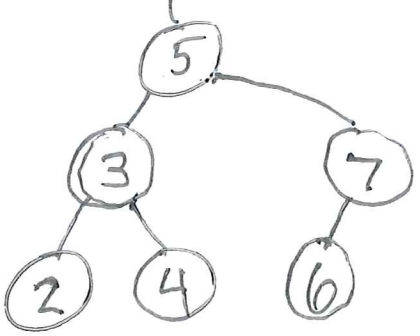
Insert 3

root

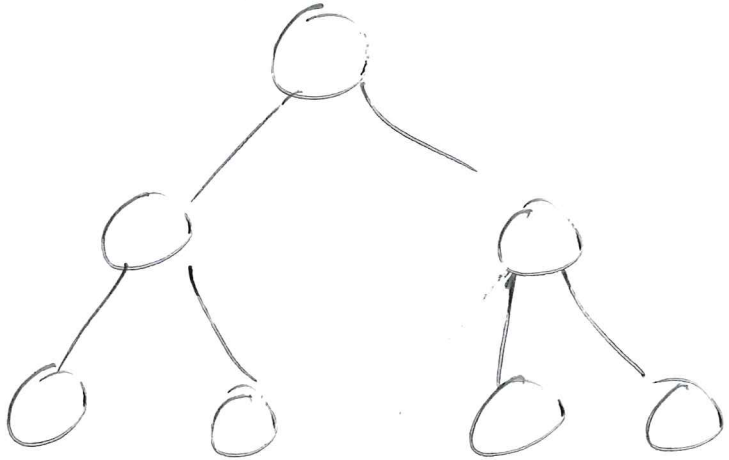


Insert 2, 4, 7, 6

root

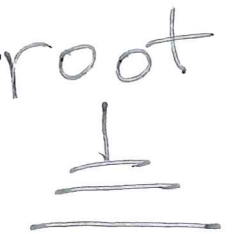


root



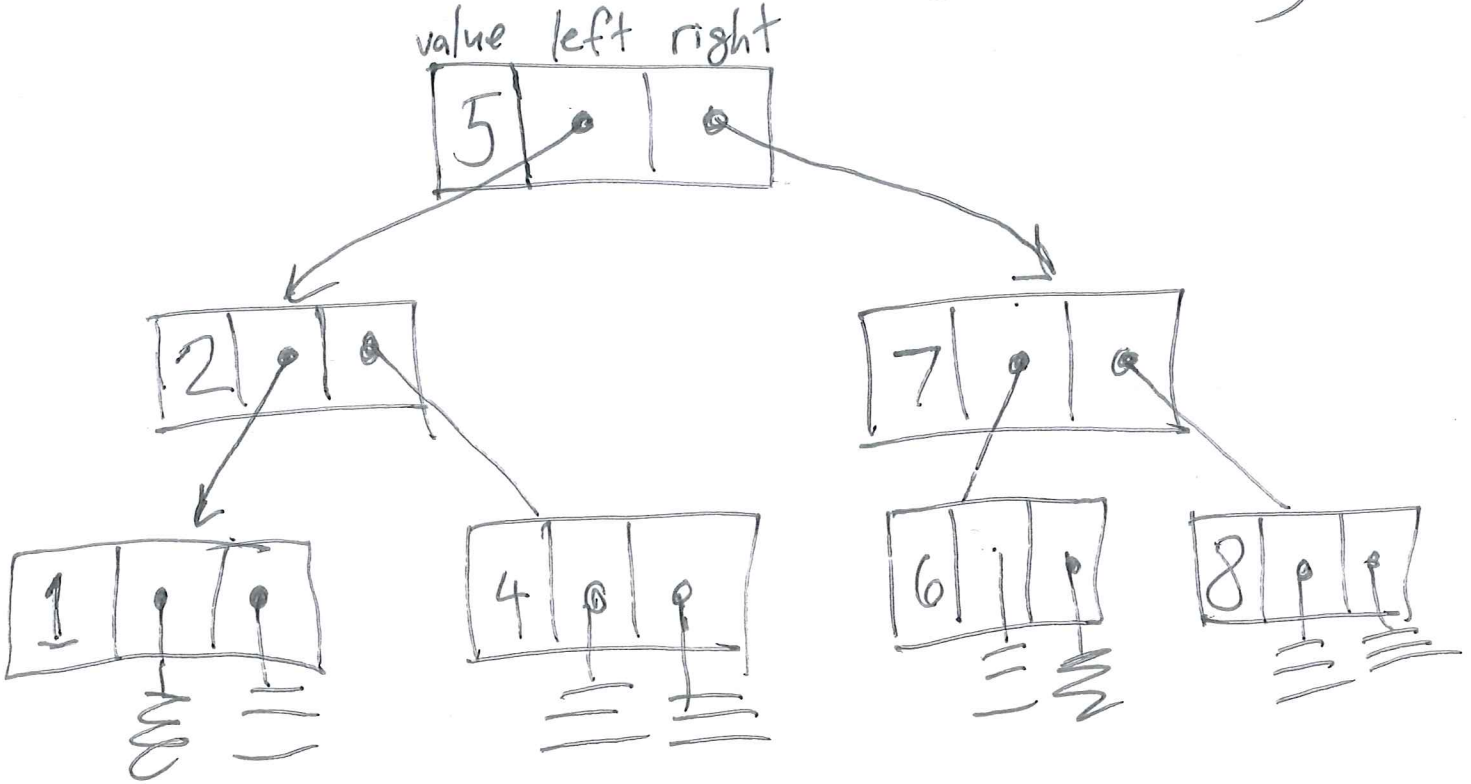
Just add in order and follow the BST constraint:
Everything in left subtree is < root.

Empty tree



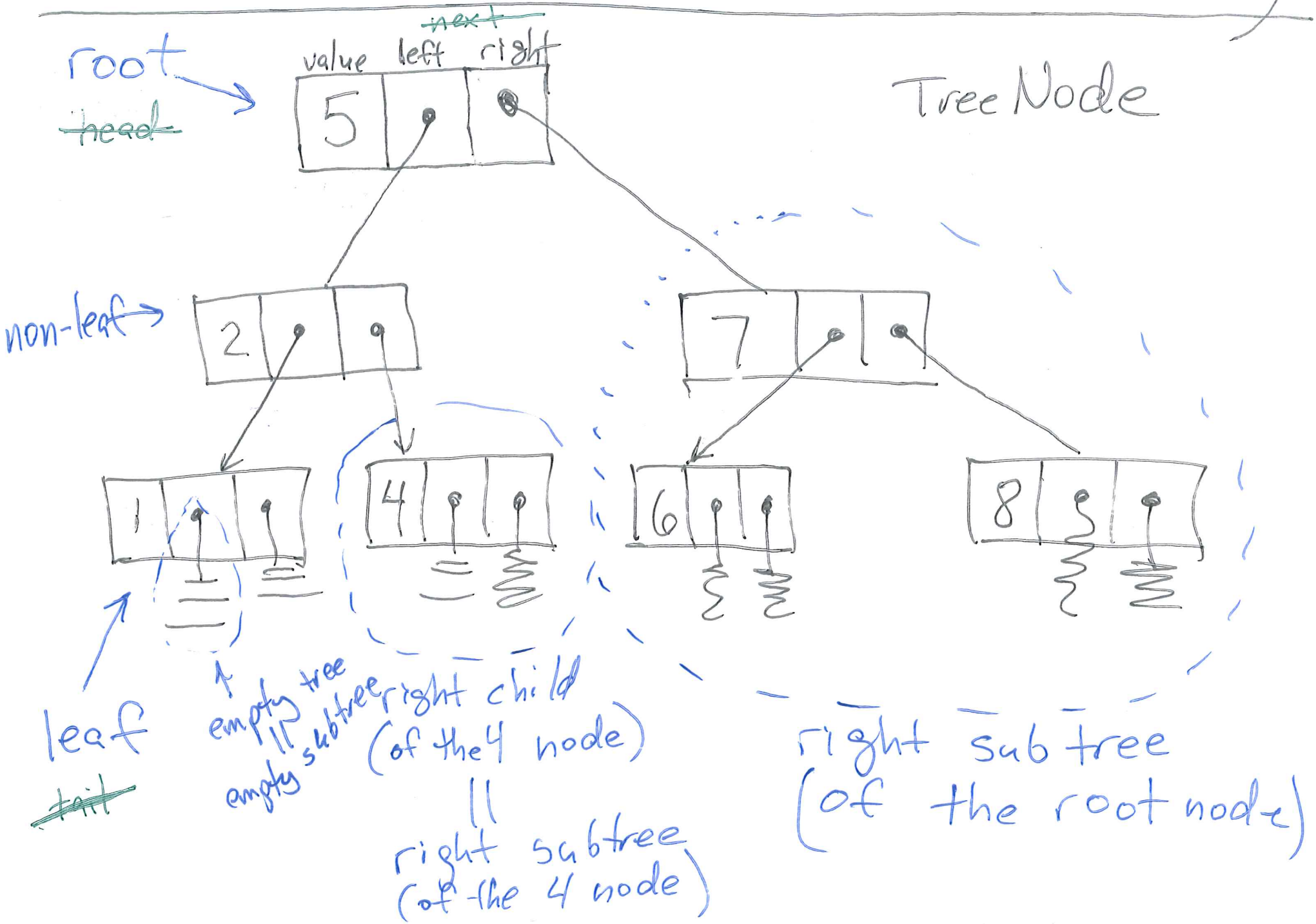
Empty list

Binary Search Trees (BSTs)



5 ✓
2 ✓
7 ✓
1 ✓
4 ✓
6 ✓
8 ✓

Binary Search Trees (BSTs)



Alternate way to think about traversals

1 2 3 4 5 6 7

traverse left subtree of 4

visit root of 4

traverse right subtree of 4

traverse left subtree of 2

visit root of 2

traverse right subtree of 2

traverse left subtree of 6

visit root of 6

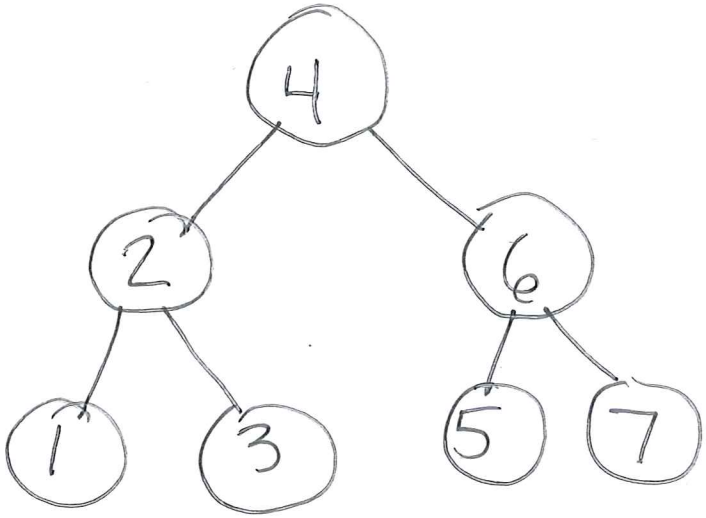
traverse right subtree of 6

visit root of 1

visit root of 3

visit root of 5

visit root of 7



In-order traversal (pseudo-code)

```
traverse(root):
    traverse(root->left)
    print root->value
    traverse root->right
```