# Lecture #19: Complexity and Search Trees

# Fast Growth

- Consider `Hackenmax` (a function from a test some semesters ago):

```
def Hakenmax(board, X, Y, N):
    if N <= 0:
        return 0
    else:
        return board(X, Y) \
               + max(Hakenmax(board, X+1, Y, N-1),
                     Hakenmax(board, X, Y+1, N-1))
```

- Time clearly depends on `N`. Counting calls to `board`, $C(N)$,the cost of calling `Hackenmax(board,X,Y,N)`, is

$$C(N) = \begin{cases} 0, & \text{for } N \leq 0 \\ 1 + 2C(N-1), & \text{otherwise.} \end{cases}$$

- Using simple-minded expansion,

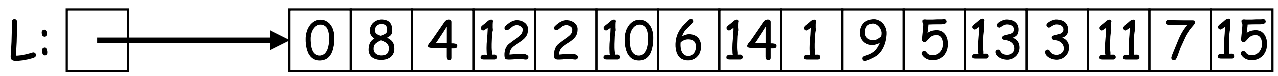
$$C(N) = 1{+}2C(N{-}1) = 1{+}2{+}4C(N{-}2) = \ldots = 1{+}2{+}4{+}8{+}\ldots{+}2^{N-1} \in \Theta(2^N).$$

# Some Useful Properties

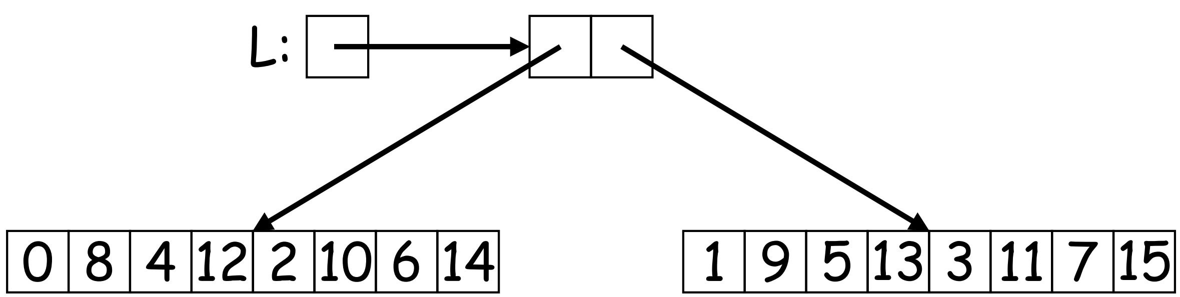
- We've already seen that $\Theta(K_0N + K_1) = \Theta(N)$ ($K$, $k$, $K_i$ here and elsewhere are constants).
- $\Theta(N^k + N^{k-1}) = \Theta(N^k)$. Why?
- $\Theta(|f(N)| + |g(N)|) = \Theta(\max(|f(N)|, |g(N)|))$. Why?
- $\Theta(\log_a N) = \Theta(\log_b N)$. Why? (As a result, we usually use $\log_2 N = \lg N$ for all logarithms.)
- Tricky: why *isn't* $\Theta(f(N) + g(N)) = \Theta(\max(f(N), g(N)))$?

# Searching, Again

- Consider the problem of searching a Python list `L` for some value:




- If we search linearly (left to right), it will take 16 comparisons in the worst case—the length of `L`.
- Suppose, however, we could divide our list in two, and somehow figure out quickly which of the two must contain our target, if it's to be found.:

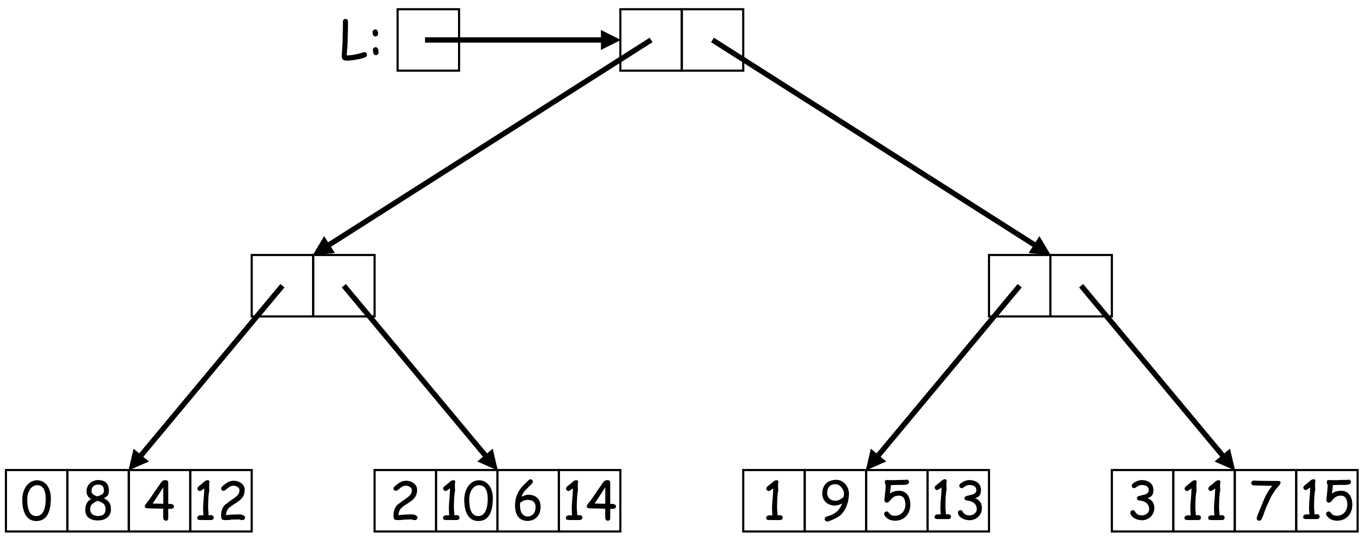



- Now the cost of finding our target is at worst

$8 +$ Cost of deciding which list it must be in

# More Slicing and Dicing

- Continuing, we'd get




- With a cost of

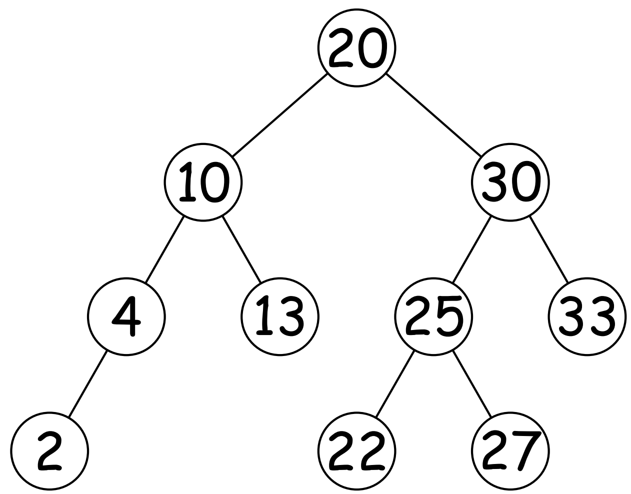
$$4 + 2 \times \text{Cost of deciding which list it must be in}$$

- As you can see, we are forming a tree.
- If we go all the way to the end (single values), we'll have a cost of

$$1 + 4 \times \text{Cost of deciding which list it must be in}$$

# Search Trees

- The preceding slides show the idea behind the *search tree*.
- The most common example is the *binary search tree*, where each decision is between two lists, and the decision criterion is whether the target is less than, greater than, or equal to a given value:




- (These trees are a bit different from what we've been using, since they have the possibility of *empty trees*, such as the missing right link at the node containing 4.)
- In more general search trees, as in this example, we don't have to divide sets of data exactly each time. Also, could have more than two branches.

# Slow Growth

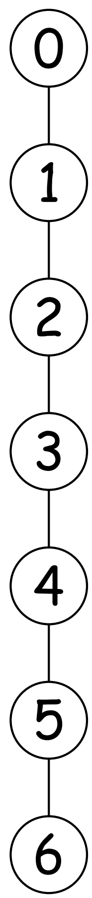
Consider a problem with this structure:

```
def tree_find(T, disc):
    p = disc(T.label)
    if p == -1:
       return T.label
    elif T.is_leaf():
       return None
    else:
       return tree_find(T.children[p], disc)
```

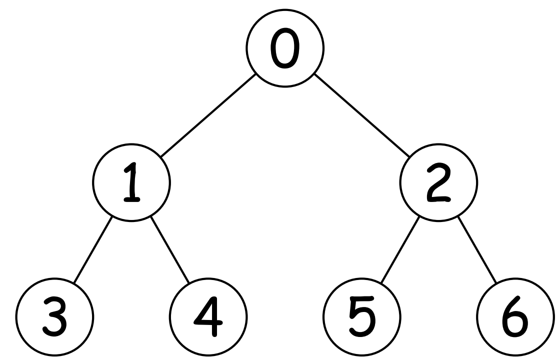
Assume that function `disc` takes (no more than) a constant amount of time.

# Kinds of Tree

- Assume we are dealing with binary trees (number of children $\leq 2$).
- Trees could have various shapes, which we can classify as "shallow" (or "bushy") and "stringy."




Maximally Deep ("Stringy") Tree




Maximally Shallow ("Bushy") Tree

# Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children $\leq 2$)?
  - 1. As a function of $D$, the depth of the tree?
  - 2. As a function of $N$, the number of keys in the tree?
  - 3. As a function of $D$ if the tree is as shallow as possible for the amount of data?
  - 3. As a function of $N$ if the tree is as shallow as possible for the amount of data?

# Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children $\leq 2$)?
  - 1. As a function of $D$, the depth of the tree? $\Theta(D)$
  - 2. As a function of $N$, the number of keys in the tree?
  - 3. As a function of $D$ if the tree is as shallow as possible for the amount of data?
  - 3. As a function of $N$ if the tree is as shallow as possible for the amount of data?

# Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children $\leq 2$)?
  - 1. As a function of $D$, the depth of the tree? $\Theta(D)$
  - 2. As a function of $N$, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of $D$ if the tree is as shallow as possible for the amount of data?
  - 3. As a function of $N$ if the tree is as shallow as possible for the amount of data?

# Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children $\leq 2$)?
  - 1. As a function of $D$, the depth of the tree? $\Theta(D)$
  - 2. As a function of $N$, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of $D$ if the tree is as shallow as possible for the amount of data? $\Theta(D)$
  - 3. As a function of $N$ if the tree is as shallow as possible for the amount of data?

# Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children $\leq 2$)?
  - 1. As a function of $D$, the depth of the tree? $\Theta(D)$
  - 2. As a function of $N$, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of $D$ if the tree is as shallow as possible for the amount of data? $\Theta(D)$
  - 3. As a function of $N$ if the tree is as shallow as possible for the amount of data? $\Theta(\lg N)$