CSC148-Section:L0301
Week#10-Friday

Instructed by

AbdulAziz Al-Helali

a.alhelali@mail.utoronto.ca

Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap course material winter17

Computer Science
UNIVERSITY OF TORONTO

Outline

- Efficiency
  - Another example on Momization
  - Searching
  - Height analysis
  - Sorting

Another example on Momization

```
def count_states(s1: SubtractSquareState) -> int:
    """ Return the number of game states reachable from here.
    """
    moves = s1.get_possible_moves()
    states = [s1.make_move(m) for m in moves]
    return 1 + sum([count_states(x) for x in states])
```

Another example on Momization

```
def count_states_mem(s1: SubtractSquareState, seen: dict) -> int:
    """
    Return the number of game states reachable from here *quickly*.
    """
    moves = s1.get_possible_moves()
    states = [s1.make_move(m) for m in moves]
    if s1.__repr__() not in seen:
        seen[s1.__repr__()] = 1
                         + sum([count_states_mem(x, seen)
                                         for x in states])
    return seen[s1.__repr__()]
```

Computer Science
UNIVERSITY OF TORONTO

Search speed of _contains_

- In the following:
  - List
  - Tree
  - Binary Tree
  - BST

contains

- Suppose *v* refers to a number. How efficient is the following statement in its use of time?
  - *v* in [10, 100, 20, 44, 50, 78, 96, 52]

- Roughly how much longer would the statement take if the list were 2, 4, 8, 16,... times longer?

- Does it matter whether we used a built-in Python list or our implementation of LinkedList?

contains

- Suppose *v* refers to a number. How efficient is the following statement in its use of time?
  - *v* in [10, 100, 20, 44, 50, 78, 96, 52]
  - Best case if data is first
  - Worst if data is not in the list

- Roughly how much longer would the statement take if the list were 2, 4, 8, 16,... times longer?
  - We have to look at every element, so it is proportional to the length of the list
- Does it matter whether we used a built-in Python list or our implementation of LinkedList?

What if we order the list?

- Suppose we know the list is sorted in ascending order?
  - [10, 20, 44, 50, 52, 78,96, 100]
  - If data is out of range?
  - If data within range?

- How does the running time scale up as we make the list 2, 4, 8, 16,... times longer?

What if we order the list?

- Suppose we know the list is sorted in ascending order?
  - [10, 20, 44, 50, 52, 78,96, 100]
  - If data is out of range? Very fast
  - If data within range? Cutting in half, 3 steps

- How does the running time scale up as we make the list 2, 4, 8, 16,... times longer?
  - One step for each doubling

lg(n)

- Key insight: the number of times I repeatedly divide n in half before I reach 1 is the same as the number of times I double 1 before I reach (or exceed) n: $\log_2 (n)$
  - often known in CS as lg(n), since base 2 is our favorite base.
- For an n-element list, it takes time proportional to n steps to decide whether the list contains a value, but only time proportional to lg(n) to do the same thing on an ordered list. What does that mean if n is 1,000,000? What about 1,000,000,000?

n	$\log_2 n$
10	3.3
10^2	6.6
10^3	10
10^4	13
10^5	17
10^6	20

trees

- How efficient is contains on each of the following:
  - our general Tree class?
  - our general BTNode class?
  - our BST class?

trees

- How efficient is contains on each of the following:
  - our general Tree class? Visit every node– linear with the number of nodes
  - our general BTNode class? Visit every node– linear with the number of nodes
  - our BST class? If the tree is “balanced” visit in about lg(n)

node packing...

- maximum number of nodes in a binary tree of height:
- 0
- 1?
- 2?
- 3?
- 4?
- n?

node packing...

- maximum number of nodes in a binary tree of height:
- 0 0
- 1? 1
- 2? 3
- 3? 7
- 4? 15
- n? 2^h-1

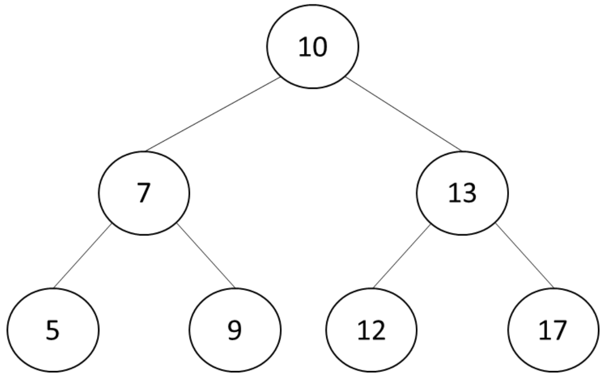
For a given number of nodes n, what is the tree height h?

node packing...

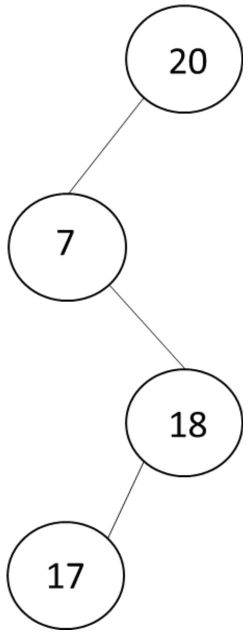
- for a given number of nodes n, what is the tree height h?
- maximum number of nodes in a binary tree = 2^h-1
  - So, n <= 2^h-1
  - n+1 <= 2^h
  - To find h take $\log_2$ both sides:
    - $\log_2(n+1) <= \log_2(2^h)$
    - $\log_2(n+1) <= h$
    - $h >= \log_2(n+1) = h$
- Will search time be proportional to lg(n)?

node packing...

- Will search time be proportional to lg(n)?
  - Only if the tree is **balanced**.
  - Searching in an **un**balanced tree is proportional to n
  - Balanced tree (AVL trees) will be covered in other courses.



Balanced BST



Unbalanced **BST**

node packing...

- Is BST the best data structure to search for ordered data?
- How would you store strings for fast retrieval?
  - What will be the arity if a tree is used to store the words

Sorting

- how does the time to sort a list with n elements vary with n?
- it depends on the search algorithm:
  - bubble sort -> n^2
  - selection sort -> n^2
  - insertion sort -> n^2
  - Quick sort -> n*lg(n) what if the list is already sorted?

Quick Sort

- idea: break a list up (partition) into the part smaller than some value (pivot) and not smaller than that value, sort those parts, then recombine the list

Quick Sort

```
def qs(list_):
    """
    Return a new list consisting of the elements of list_ in
    ascending order.

    @param list list_: list of comparables
    @rtype: list

    >>> qs([1, 5, 3, 2])
    [1, 2, 3, 5]
    """
    if len(list_) < 2:
        return list_[:]
    else:
        return (qs([i for i in list_ if i < list_[0]]) +
                [list_[0]] +
                qs([i for i in list_[1:] if i >= list_[0]]))
```

Quick Sort

$$qs([4, 2, 6, 1, 3, 5, 7])$$

$$qs([2, 1, 3]) + [4] + qs([6, 5, 7])$$

$$qs([1])+[2]+qs([3]) \quad + \quad [4] \quad + \quad qs([5])+[6]+qs([7])$$

$$[1] + [2] + [3] \quad + \quad [4] \quad + \quad [5] + [6] + [7]$$

$$[1, 2, 3] \quad + \quad [4] \quad + \quad [5, 6, 7]$$

$$[1, 2, 3, 4, 5, 6, 7]$$