Searching in B-Tree

- Check if "current node" is =x or =NIL.
- Equal to x    -- return,
- Equal to NIL  -- return "not found"
else:
  if x ≤ current key  ---- search in the left tree
  otherwise search in the right tree.

"Go down the tree, turning right/left as appropriate..."
- Running time: O(h),  h=height of the tree.
- Note that this was impossible to do with a heap!

Inserting into B-Tree

- Insertion: search for key, and put it in the first empty space.
- Insertion takes O(h).
- Sort:
  » Insert item-by-item,
  » h-order walk,
  » O(n^2)...
  » Min/max - go all the way left or all the way right.

Relation to Quicksort

- Randomly permute input.
- Consider example: 3 1 8 2 6 7 5
  » Quicksort chooses 3, then compares 1, 8, 2, 6, 7, 5 to 3.
  » Then chooses 2, compares 1 to 2
  » chooses 6, compares 6, 7, 5 to 8.
  » B-Tree: chooses 3, places as root
  » Then chooses 1, compares with 3, put in place.
  » chooses 2, compares with 1, 3, put in place etc...
  » Overall, same comparisons, only different order!!

Successor/Predecessor

- Successor:
  » If right(x) != NIL, return TreeMin(right(x))
  » y=parent(x)
  » while y != NIL & x=right(y)
  » x=y
  » y=parent(x)
  » return y
- Successor of 6 is 8.

Deletion

- 3 Cases:
  » No children
  » 1 child
  » 2 children
- 3rd case: put successor(s) instead of x.
  » B-Tree property satisfied.
  » Delete "hole" using case 1 or 2. (successor does not have left child!)