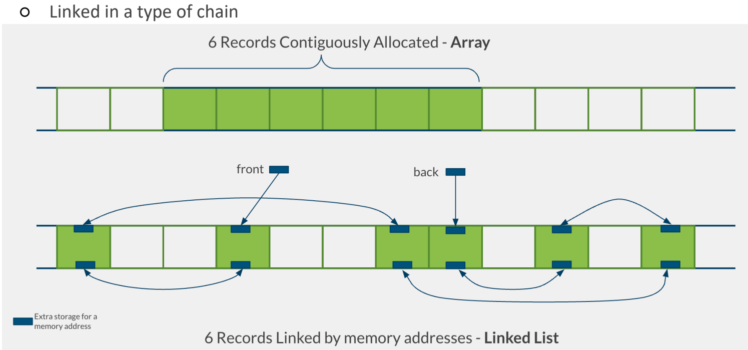
Basics & BST

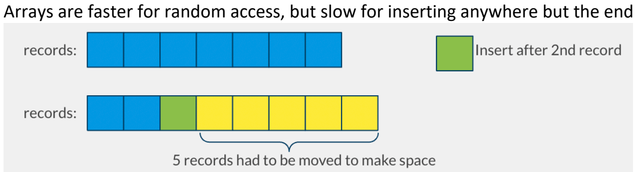
Wednesday, January 08, 2025 9:22 AM

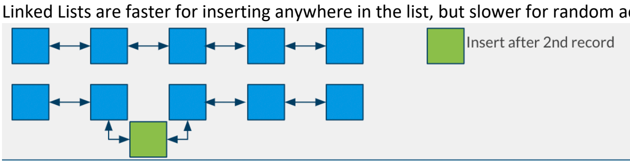
- **Record -** A collection of values for attributes of a single entity instance; a row of a table
- **Collection** - a set of records of the same entity type; a table
    - Trivially, stored in some sequential order like a list
- **Search Key** - A value for an attribute from the entity type
    - Could be >= 1 attribute

- If each record takes *x* bytes of memory, we need *n*x* bytes of memory
- **Continuously Allocated List**
    - All *n*x* bytes are allocated as a single "chuck" of memory
- **Linked List**
    - Each record needs *x* bytes + additional space for one or two memory addresses
    - Linked in a type of chain



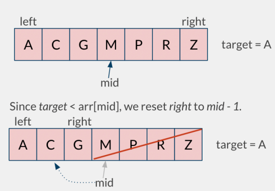
Pros and Cons

- Arrays are faster for random access, but slow for inserting anywhere but the end
- 

- Linked Lists are faster for inserting anywhere in the list, but slower for random access
- 

Binary Search

- Input: Sorted array, target value
- Output: Location (index) of where the target is located or a not found indicator
-
```
def binary_search(arr, target)
  left, right = 0, len(arr) - 1
  while left <= right:
    mid = (left + right) // 2
    if arr[mid] == target:
      return mid
    elif arr[mid] < target:
      left = mid + 1
    else:
      right = mid - 1
  return -1
```

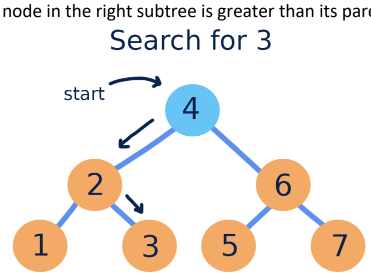


Time Complexity

- Linear Search
    - Best Case: 1
    - Worst Case: n (target not in array)
    - O(n) time complexity in worst case
- Binary Search
    - Best Case: 1 (target in middle)
    - Worst Case: log2(n) (target not in array)
    - O(log2(n)) time complexity

- An array of tuples (specialVal, rowNumber) sorted by specialVal
    a. We could use Binary Search to quickly locate a particular specialVal and find its corresponding row in the table

corresponding row in the table
   b. But, every insert into the table would be like inserting into a sorted array - slow…
- A linked list of tuples (specialVal, rowNumber) sorted by specialVal
   a. searching for a specialVal would be slow - linear scan required
   b. But inserting into the table would theoretically be quick to also add to the list.

Fast insert and search??
- **Binary Search Tree** - a binary tree where every node in the left subtree is less than its parent and every node in the right subtree is greater than its parent.
- 

-

Creating and inserting into a bin tree
23, 17, 20, 42, 31, 50

            23   --> root
      17   43
         20   31   50

Tree Traversals
- Pre Order
- Post Order
- In Order
- Level Order
   - 23, 17, 43, 20, 31, 50
   - Start at root, put left and right child in a queue, process next element of list (add it's left and right child to end and remove it)
      - Called a deque in Python (double ended queue)

Class BinaryTreeNode (self, value, left = None, right = None)
      value: int
      left: BinTreeNode
      right: BinTreeNode

Function
      root = BinaryTreeNode(23)
      root.left = BinaryTreeNode(17)
      root.right= BinaryTreeNode(43)
      root.left.right = BinaryTreeNode(20)
      # No need to implement insert function

*Use a dict to keep level index