# 2/42 List as Abstract Data Type

* List as an ADT:
  + Can store a given # of elements of a given data type
  + Write/modify elements at a position
  + Read element at a position
* Lets redefine our list as a dynamic list:
  + Empty list has size 0
  + Can insert element @ position
  + Can remove
  + Can count
  + Read/modify @ position
  + Specify data type
* To implement such a dynamic list using Arrays, add operations on top of arrays
  + Assume we want dynamic int list: **A [4, 5, 6, 7, 9, …]**
    - int A[MAXSIZE] // very large
    - int end = -1; // equal to position of final element. -1 when empty
    - insert(2) // always inserted at the tail of the list,
    - insert(4) … (6)…(7)…(9)
    - insert(5,2) // (value, position)
    - remove(0)
    - Implementation is complete, compare features to ADT. But what about MAXSIZE?
    - When array is full, create larger array, copy previous array into the new array. Free the memory for the previous array.
      * Creating/Copy is costly in time, we should avoid this. Instead each time array is full, simply create a new array double the size and copy into new array.
* The study of data structures also considers the cost of operations, let’s consider our Dynamic List:
  + Access – Read/Write element @ index
    - Constant time. **O(1)**
    - Big Oh notation: used to describe the time complexity of operations
  + Insert- T proportional to n (linear function)
    - n is the length of the list
    - **O(n)**
  + Remove **O(n)**
  + Add
    - Constant time if not full
    - **O(n)** if full, because we need to create larger array and copy current contents
* The use of an array as a dynamic list is not memory efficient because the list can grow and shrink a lot. Most of the time the array is empty and unused.
* Linked List, is a more memory efficient data structure for a dynamic list