CSCI 262 Lecture 15 – Array List

Outline

- Array List – a list data structure built on dynamically allocated arrays
  - Very unlike linked lists, although same operations can be done
  - Essentially equivalent to a vector
- Critical operation: resize when array fills up
  - This is why we need dynamic allocation
  - For complexity reasons, it works well to double size each time (some wasted space, but complexity worth it)
- Good software design techniques:
  - Encapsulation into a class
    - Keep related info – size, capacity, array – all together with the operations that act on them
    - Protect consistency of critical variables
  - Refactoring – pull common code out into its own method

Readings

Read chapter 13.1 for Monday.

Wikipedia has a nice article on dynamic arrays (a more generic term for an array list) that also includes discussion of optimal growth strategies and differences in implementations among different languages and libraries: https://en.wikipedia.org/wiki/Dynamic_array

Self Check

1. In the _resize() method, the order of steps is fairly important. What happens if we reverse the last two steps (deallocating the old array and reassigning the pointer)? What happens if we simply don't do the deallocation step?

2. In the insert() method for our array_list class, our loop to move elements goes from _size down to the index we want to insert at – why do we do it this way instead of looping from the index up? Similarly, why does erase go up instead of down?

3. Insert and erase on the array list are both O(n) operations in the worst case. What is the best case? What is the average case, if we assume we will pick an element at random to remove?

For Further Practice

Many sorting algorithms rely on being able to swap elements in a sequence efficiently. Note that get/set at an index for array lists is O(1) – this is due to the fact that get/set basically just access the underlying array memory directly, which is an O(1) operation on modern computers. Linked lists are not well suited to these algorithms, since indexed get/set are O(n).

On the other hand, merge sort doesn't rely on swapping values; and with array lists (or vectors, or arrays) the usual algorithm requires additional space (to hold the new list/vector/array while merging). We say that merge sort has 2n space complexity (which is still O(n), so not really a problem). This leads to the following question – think about it, and consider coding a solution: can you implement merge sort efficiently (O(n log n)) on linked lists? What is the space complexity?