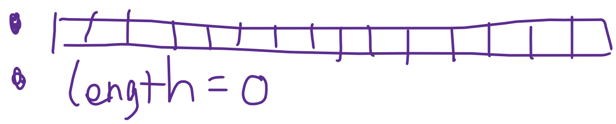
Time complexity of ArrayList methods

- ArrayList is a data structure that is a list and implemented with an array. The list class in python is a realization of ArrayList (but there are differences), so there is no additional ArrayList class in Python.

- Construction method
  - To construct a new ArrayList, we simply create an "empty" array. If the created array has length n, then the time complexity of construction *is* $\Theta(n)$. In addition, so that the user knows what the index of the last item is, we need an extra attribute $length$ (initially $= 0$) to store the number of items in the current ArrayList.

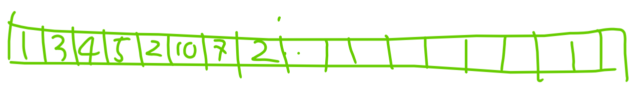


- Size()
  - Since there is attribute $length$, one can get size of an ArrayList in $\Theta(1)$.

- IsEmpty()
  - One can return $length == 0$, which takes $\Theta(1)$ time.

For the following methods, we assume that there are n items in the ArrayList.

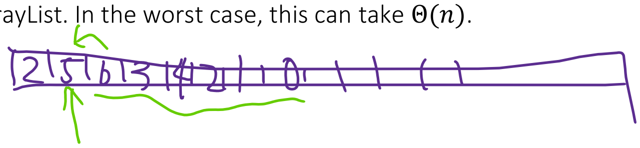
- Indexing(i)
  - To get the i^{th} item in an ArrayList, since items are stored contiguously in an array, so it only *takes* $\Theta(1)$ time.



- Search(item)
  - To search in an ArrayList is the same as searching in an array from index 0 to $length$. If the ArrayList is sorted, then we can use binary search, which has time complexity $O(\lg n)$; if the ArrayList is not sorted, we need to use linear search, and it has time complexity $O(n)$.

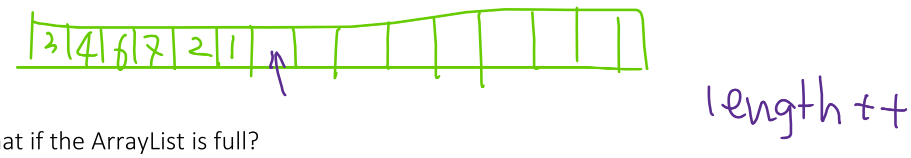
- Pop ()
  - Return the item at index $length - 1$, then decrease the $length$ of the ArrayList. These operations can be done in $\Theta(1)$ time.

- Pop (i)
  - Return the item at index i and move all items after it one spot to the left, then decrease the $length$ of the ArrayList. In the worst case, this can take $\Theta(n)$.

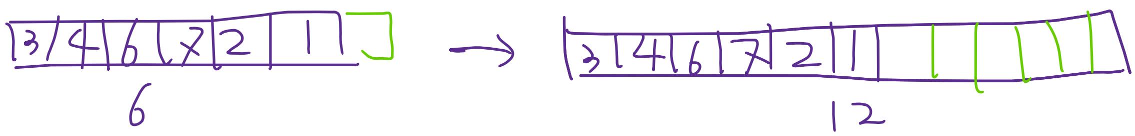


- Remove(item)
  - This can be done by search(item) + pop(i), so it has time complexity $O(n)$

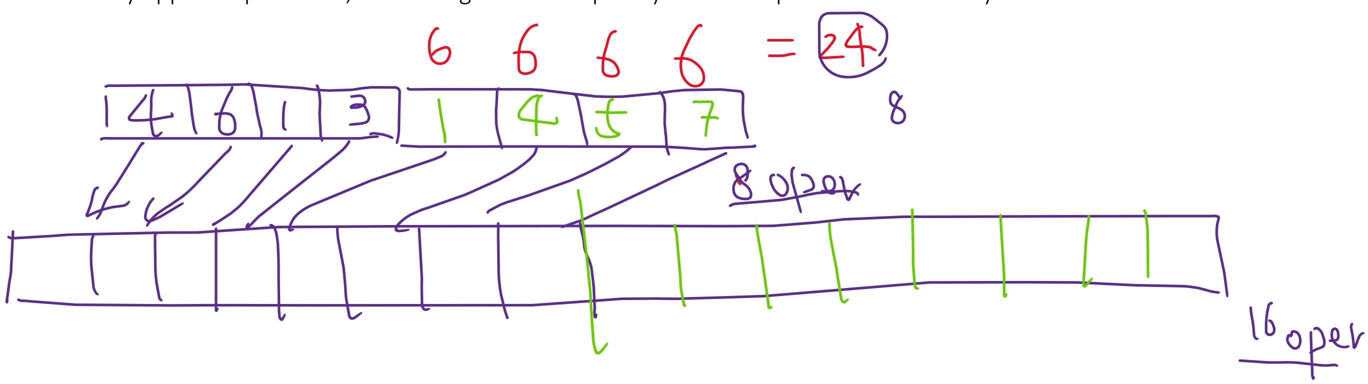
- Append(item)
  - When the array in the ArrayList is not full yet, one can simply add the item to at index $length$ then increase the $length$ of the ArrayList. These operations only need $\Theta(1)$ time.



  - What if the ArrayList is full?
    We will copy everything in the current array into a new array of doubled length.

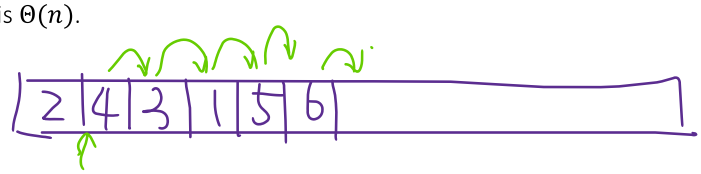


  - It is easy to see the operation will spend a lot of time, but why append(item) in the list class only takes $O(1)$ time?
    It is an **amortized cost**. An intuition is that this expansive operation doesn't appear very often, then if there are many append operations, the average time complexity of these operations is still very low.



  - One can also shrink the array in an ArrayList when it is too empty. If you half the size of the array whenever it is less than $1/4$ full, the amortized cost of pop and pop(i) will be the same.

- Insertion (i, item)
  - To insert an item to index i can also trigger the expansion of array, so the time complexity here is also an amortized cost. We need to move all items after index i one spot to the right, so in the worst case, the time complexity is $\Theta(n)$.



- Sort ()
  - Sorting an ArrayList is the same as sorting in an array from index 0 to $length$. So far, we have seen several $\Theta(n^2)$ sorting algorithms.
  - **Merge Sort**: this is the first recursive algorithm we see in this class. An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input.

MERGE-SORT $(A[p \dots r])$

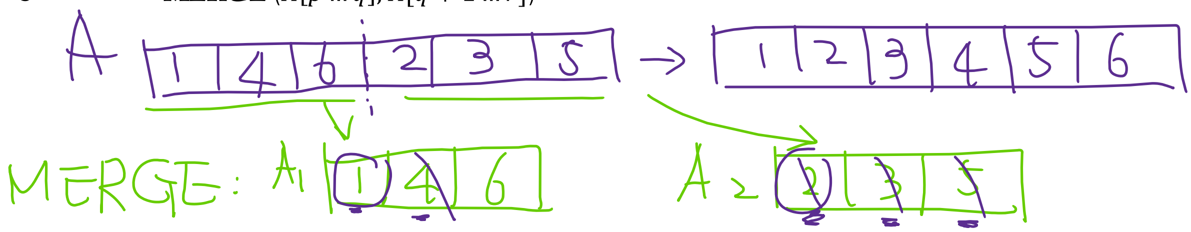
if $(p < r)$

  $q = \left\lfloor \frac{p+r}{2} \right\rfloor$

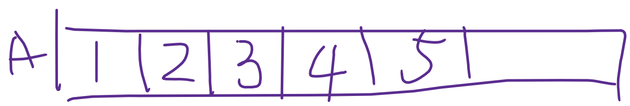
  MERGE-SORT $(A[p \dots q])$

  MERGE-SORT $(A[q+1 \dots r])$

  MERGE $(A[p \dots q], A[q+1 \dots r])$



MERGE $(A[p \dots q], A[q+1 \dots r])$



$A_1 = A[p \dots q]$

$A_2 = A[q+1 \dots r]$

while A_1 and A_2 are both nonempty

  Remove the smaller of first remaining elements of A_1 and A_2 from its array; and put it at the next available spot in A

  if this removal makes one list empty

    then remove all elements from the other list and append them to A

  - What is the time complexity to merge to arrays of size $\frac{n}{2}$?
    - After each comparison, at least one element will be moved from either A_1 or A_2 to A.
    - There are n elements in total in A_1 and A_2, all of them will be moved to A by the end of the algorithm.
    - Thus, there will be n moving operations at most $n-1$ comparisons. The time complexity to merge n elements is $\Theta(n)$.

  - Merge sort visualization
    https://opendsa-server.cs.vt.edu/embed/mergesortAV

  - What is the time complexity to merge sort n elements?
    Let $T(n)$ be the time complexity to merge sort n elements, then $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n)$.

- [Master Theorem] The recurrence $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, where $f(n)$ is a polynomial of n, can be solved as follows.
  - If $\frac{af\left(\frac{n}{b}\right)}{f(n)} < 1$, then $T(n) = \Theta(f(n))$.
  - If $\frac{af\left(\frac{n}{b}\right)}{f(n)} = 1$, then $T(n) = \Theta(f(n) \cdot \log_b n)$.

- If $\frac{af\left(\frac{n}{b}\right)}{f(n)} > 1,$ then $T(n) = \Theta(n^{\log_b a})$.

$\Theta(n) = c_1 \cdot n$

1. What is the time complexity to merge sort n elements?
   - In $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n)$, we have $a = 2, b = 2, f(n) = \Theta(n)$. Then $\frac{af\left(\frac{n}{b}\right)}{f(n)} = \frac{2 \cdot \Theta(\frac{n}{2})}{\Theta(n)} = \frac{\Theta(n)}{\Theta(n)} = 1$, and we are in the second case of Master Theorem, so $T(n) = \Theta(n \cdot \log_2 n)$.