What’s elegance? What’s important?

- **Reverse the elements in an array**
  - 1, 2, 3, 4, 5 changes to 5, 4, 3, 2, 1
  - If we swap items in an n-element array, how many swaps?

```cpp
void reverse(tvector<string>& a)
// pre: a has a.size() elements
// post: elements are reversed
{
    for(int k=0; k < _______; k++)
    {
        Swap(     ,     );
    }
}
```

- **Suppose we use first, last, suppose just k, any difference?**
Change the problem, *interchange*

- Take first $m$ elements, make them last $m$ elements

  
  Original: 1 2 3 6 7 8 9

  Interchange: 6 7 8 9 1 2 3

- If you use an extra array, how do you interchange?
  - What is complexity, in terms of $N$, total # elements?

- If you don’t use an extra array, naïve solution is
  - Move first element to end, shift
  - Repeat $M$ times
  - What is complexity?

- What about a circularly linked list, how is problem solved?
A different approach to interchange

● Starting state

● Next step, how? Complexity?

● Next step, how? Complexity?

● What is the final step?

● Can we use the function Reverse we first wrote? Why?

● What is complexity of new method vs. complexity of old?
New problem: find the $k^{th}$ largest

- In an array, find the element that’s larger than $k$ elements
  - $0^{th}$ largest is the smallest element, larger than zero
  - $N^{th}$ largest is larger than $N$, it’s the largest
  - What about $7^{th}$ largest in a six element vector?

- Methods to solve the problem
  - We know how to find $0^{th}$ largest (min), repeat $k$ times
  - We know how to sort (call a function), where is $k^{th}$ largest?
    - How fast is quicksort?
  - Which of these methods is best? What about other methods?
    - We’ll use part of quicksort, called partition
Two steps to solve select \(k^{th}\) largest

- **Step one, partition vector**
  - Piv is index of \(x\)
  - Can do this in \(O(n)\) time

- Call `findKth(a, 0, a.size(), k)`; what’s missing?

```cpp
int findKth(tvector<int>& a, int left, int right, int k)
// pre: a has a.size() elements, left <= right
//      left <= k <= right, 0<=left, right<=a.size() 
// post: return kth largest element of a
{
    int piv = Partition(a, left, right);
    if (piv == k) return a[piv];
    if (k < piv) return findKth(a, 0, piv-1, k);
    else return findKth(a, , , );
}
```
Final Analysis

```cpp
int findKth(tvector<int>& a, int left, int right, int k)
    // pre: a has a.size() elements, left <= right
    //      left <= k <= right, 0<=left, right<=a.size()
    // post: return kth largest element of a
{
    int piv = Partition(a, left, right);
    if (piv == k) return a[piv];
    if (k < piv) return findKth(a, 0, piv-1, k);
    else return findKth(a,   ,   ,   );
}

● What’s the recurrence relation? Average case? Solution?
```