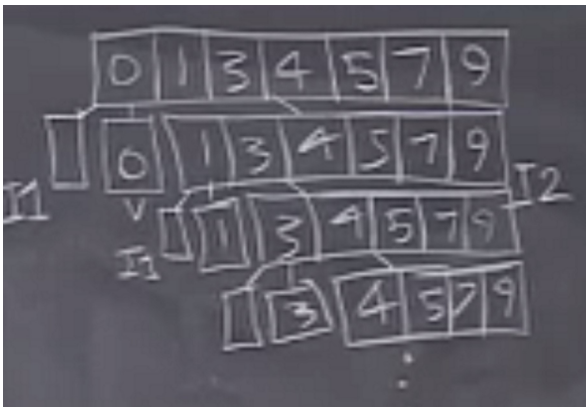
Quick Sort

Algorithm:

Start with list I of n items we want to sort
Choose pivot item v from I
Partition I into 2 unsorted lists I1 and I2
I1: Will Have All Values < v
I2: Will Have All Values > v
Any item = v can go into either list
Pivot v does not go in either list
Sort I1 recursively, yielding sorted list S1
Sort I2 recursively, yielding sorted list S2
Concatenate S1, V, S3, yielding sorted list S

Properties:

- Recursive divide and conquer algorithm
  - All work in dividing
  - Conquer is trivial
- Fastest known comparison based sorting algorithm for arrays
  - For LinkedList MergeSort faster
  - Constant factor hidden in Big O Notation is smaller than MergeSort
- In-Place is fast.
  - Very tricky to code
- $O(n^2)$ worst case time complexity
  - Hardly ever happens, mostly O(n log(n)) time.
- Likely to run in $O(n^2)$ if input is not very random at all or array is sorted.
- When input already sorted, choosing first or last item as pivot is bad policy.
  - Degrades partition step performance
  - Cannot fill I1 or I2
- Pivot should not be highest or lowest number (or close to highest or lowest number).
  - Degrades partition step performance (n^2)
  - Cannot fill I1 or I2
- All numbers same degrades to $O(n^2)$ performance.
  - Cannot fill I1 or I2
- Pivot Choosing Strategy:
  - Randomly Select Item as Pivot
    - Makes this now a "Randomized" algorithm
      - Algorithm where random numbers improve performance
    - Any pivot strategy you give me, except random, most likely find pivot that messes up.
    - On average, [1/4, 3/4] split.
      - Enough to prove randomized runs in expected O(n log(n)) time.
    - Big enough list, you use "Median-of-Three" strategy
      - Only pays off for big list
      - Choose three random items, take median, use as pivot
        - Do this at beginning of recursion tree and then as you get down, revert to single random algorithm.
    - Problem for LinkedList: Have to walk through List to find Pivot (Slow!)
      - Still worth extra effort
  - Selecting Middle Item as Pivot
    - Not bad, but pipe order input can give $O(n^2)$ complexity:
1	2	3	4	3	2	1
  - If you know distribution of data, you can choose good pivot value and increase speed.
    - Example, Symmetric Bell Curve Distribution
      - 1st pivot, choose mean
      - 2nd pivot, know more about distribution, then choose pivot again
      - … Key point is you know details about distribution to choose following pivots …
- QuickSort on Linked List
  - Equal to Pivot
    - Partition I into 3 lists instead of 2
      - I_1
      - I_2
      - I_V: Contains pivot and all items with same key as V
    - Sort I1 and I2, not Iv
    - Concatenate S1 + Iv + S2

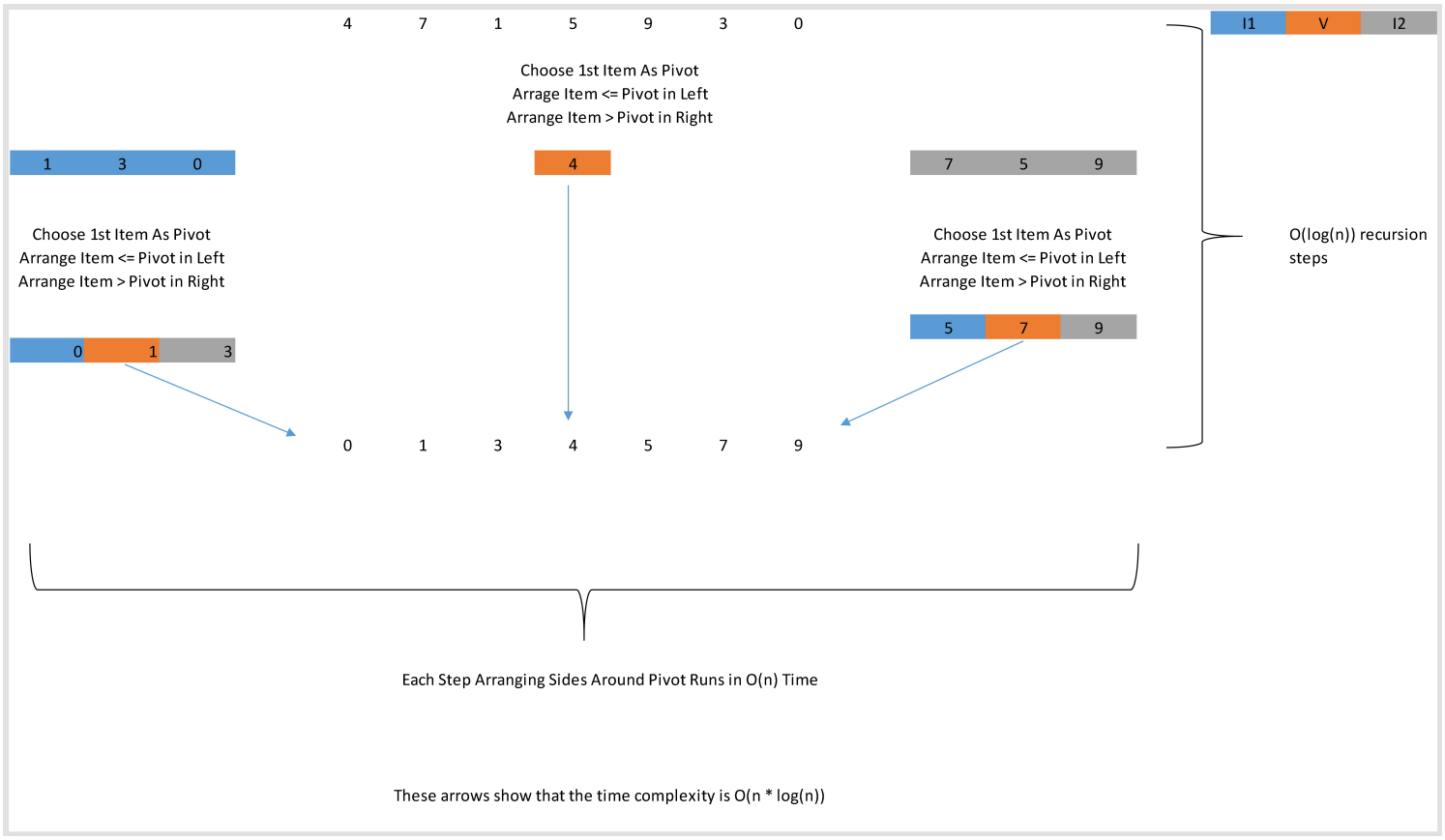


Picture illustrating why QuickSort doesn't work well if bad pivot is chosen with already sorted list.

This causes $O(n^2)$ behavior:

- The recursion divide step degrades
  - log(n) to n which implies $O(n * n) = O(n^2)$

Quick Sort - Spreadsheet


4 7 1 5 9 3 0
Choose 1st Item As Pivot
Arrage Item <= Pivot in Left
Arrange Item > Pivot in Right
1 3 0
4
7 5 9
Choose 1st Item As Pivot
Arrange Item <= Pivot in Left
Arrange Item > Pivot in Right
Choose 1st Item As Pivot
Arrange Item <= Pivot in Left
Arrange Item > Pivot in Right
0 1 3
5 7 9
0 1 3 4 5 7 9
I1 V I2
O(log(n)) recursion steps
Each Step Arranging Sides Around Pivot Runs in O(n) Time
These arrows show that the time complexity is O(n * log(n))

Quick Sort Algorithm Continued

Array "a" we want to sort.
    This will be recursive algorithm.
    Want to sort without creating whole new array.
Start by sorting region between a[low] and a[high]
Choose a pivot "V"
Swap with last item in array which is a[high]
Got pivot out of the way, all keys are continuous which allows sorting

Quick Sort Algorithm Continued - Spreadsheet

	3	8	0	9	5	7	4	
	Low				V		High	
								swap(v, high)
					Swap			set pointer I = low - 1
								set pointer j = high
	3	8	0	9	4	7	5	
i	Low						j	Stuff that needs to be sorted is sandwitched betwe
								Moving Forward, Enforce Invariants:
								All items left of index have key <= pivot
								All items to right of index have key >= piv
								Next,
								Advance I to key >= pivot
								Decrement j to key <= pivot
	3	8	0	9	4	7	5	
		i			j			Now,
								We have found two keys on
								wrong side of partition.
								Time to swap(I, j).
	3	4	0	9	8	7	5	
		j			i			Repeat until I >= j
	3	4	0	9	8	7	5	
			j	i				Now,
								Done partitioning. I and J have crossed.
								Swap pivot back in middle
								Whatever value at index i
	3	4	0	5	8	7	5	

Quick Sort Code

```
void quickSort(int[] arr, int left, int right) {
        int index = partition(arr, left, right);
        if(left < index - 1) {
                quickSort(arr, left, index - 1);
        }
        if(index < right) {
                quickSort(arr, index, right);
        }
}

int partition(int[] arr, int left, int right){
        int mid = (left + right) / 2;
        int pivot = arr[mid];
        while(left <= right) {
                while(arr[left] < pivot) {
                        left++;
                }
                while(arr[right] < pivot) {
                        right++;
                }
                if(left <= right) {
                        swap(arr, left, right);
                        left++;
                        right--;
                }
        }
        return left;
}
```