**Recursions**

"Breaks down problem to be solved into smaller, discrete steps that are solved using **same** algorithm"

Text

Description automatically generated with medium confidence**Factorial**

**Linked List**

* Text

  Description automatically generatedIn this version, we are printing from end -> beginning
* If we switch orders of rprint and cout, we are printing in forward order

**Quick Sort**

One of the **most efficient** sorting algorithms.

qsort in stdlib.h

Idea

* Pick a pivot value from an array
* Move every value < pivot to the left of the pivot
* Move every value >= pivot to the right of the pivot
* Quicksort sub-array to the left of pivot
* Quicksort sub-array to the right of pivot

Text, application

Description automatically generated// pivotIndex - 1 <= begin or pivotIndex - 1 >= end happens when the pivot is at the positions that we DON'T NEED to do recursion anymore

(0 element or 1 element in sub-array)

// Select and Suffle will select the pivot, move stuffs to left & right of it (meaning, moving the pivot to the "middle", then **return new pivot index**

\*\* **selectAndSuffle**

Idea

[begin, begin + 1, begin + 2, … , end]

* Iterate through array, **swap** current-th element < pivot to a[i]
  + a[begin] is the pivot. Pivot stays at a[begin] til last step.
  + i starts from begin + 1 and loops til end.
  + Current is an index and starts at begin. It keeps track of the index where **smaller element should be placed to**.
  + Whenever we meet element < pivot, current moves up, and swaps a[i] to that index.
  + Current will **always** end at the index where pivot should be (if there are k elements smaller than pivot, the pivot will be placed to k-th index)
  + Therefore, after the loop, swap a[current] with a[begin] where the pivot is.

Text

Description automatically generated

Shape

Description automatically generated

**Complexity**

* Best case: Assume that we are always picking the pivot by median value, there would be log2(n) tree levels 🡪 Average performance, for each level of tree (scanning all n elements) O ( n\*log(n) )
* Worst case: Always choose smallest/largest pivot 🡪 Every time we only reduce the number of elements in the array by 1 🡪 n levels of tree 🡪 Complexity n2

Therefore, it's not a good practice to actually just pick the first value for pivot.

In reality, we usually compare first, last, and middle number to pick (good approximation)

median ( a[begin], a[middle], a[end] )