Techniques

1. See the Data - *algo question = numbers and data*
2. Multiple Passes - *simplify & modify into more familiar problems*
3. Problem Toolbox
4. Data Type Toolbox

ADT Toolbox

hash-map *dictionary* - O(1) to look up data based on key, O(1) to set data based on key. **Your fave ADT for faster, simpler algos!**

list *array, sequence* - O(n) to linear search, O(1) to look up index - good for ordered data

stack *Last in First Out* - O(1) to add or remove - good for reversing things, recursive things

queue *First in First Out* - O(1) to add or remove - good for processing things in order

tree *root* node connected to children and descendants

graph *vertices* connected with *edges*

Sorting Algos

quicksort *time avg O(n log n) - space O(log n)* - pick a pivot, sort w.r.t. to pivot into two partitions, sort each partition using quicksort (another pivot)

mergesort *time avg O(n log n) - space O(n)* - split into arrays, sort each array using mergesort (smaller arrays), combine sorted arrays in order

insertion sort *time avg* $O(n^2)$ *- space O(1)* - "pick up" an item, and move left into sorted portion until you find a spot where it fits between, then place

selection sort *time avg* $O(n^2)$ *- space O(1)* - find smallest item, put on left in sorted portion, then repeat for next smallest in unsorted

bubblesort *time avg* $O(n^2)$ *- space O(1)* - keep on swapping adjacent items, until you go through the array with no more items needing to be swapped

Problem Toolbox

minimum *O(n)*

```
min_found = None
for n in list_of_numbers:
    if min_found == None:
        min_found = n
    if n < min_found:
        min_found = n
```

maximum *O(n)* - implementation is same as above, save for `n > max_found`

counting occurrences *O(n)*

```
c = {}
for item in items:
    if item not in c:
        c[item] = 0
    c[item] += 1
```

filtering collection *O(n)* - create a new, smaller list only containing data adhering to a condition

```
output = []
for item in items:
    if item > 5: # or something
        output.append(item)
```

transforming collection *O(n)* - create a list of same length, but with data changed

```
output = []
for item in items:
    # Any method or operation
    modified = item * 2
    output.append(modified)
```

intersection between collections *O(n)* - find items in one collection that are (or are not) in another collection. Use `set` instead of `list` since set operations are O(1)

```
data_2_set = set(data_2)
overlap = []
for item in data_1:
    if item in data_2_set:
        overlap.append(item)
```

cross operation $O(n^2)$ - perform operation on each element w.r.t. another element

```
out = []
for a in data_1:
    for b in data_2:
        out.append(a * b)
```

Data structures

Many approaches to implement ADTs:

Linked lists Implement with dicts

```
c = {"data": "C", "next": None}
b = {"data": "B", "next": c}
a = {"data": "A", "next": b}
```

Stacks, queues Just a list, but only using first and/or last items

Tree Impl. with dicts and lists

```
root = {
  "data": "Grandma",
  "children": [{
      "data": "Mom",
      "children": [{"data": "Me"}],
    }, {
      "data": "uncle",
      "children": []
  }]
}
```

Graph (directed) Store vertices in dict, outward edges in list

```
graph = {
  "NYC": ["SFO", "ORD", "DTW", "LAX"],
  "SFO": ["LAX", "OAK", "NYC", "DTW"],
  "SAC": ["OAK", "SFO", "LAX"],
}
```

Terminology

Problem Given an input of a certain type, make an output (solution)

Algorithm A precise series of steps that solves a problem

Implementation The code used to perform an algorithm

Time complexity *algorithmic speed* - "Given a problem input of size *n*, how many steps does algo take to solve?"

Big O Notation for time complexity

Abstract Data Types Ways to structure data, defined in English, useful to model data

Data structures Implementations with code of an ADT, used to implement algorithms