Recipes

# Maximum Sub array

Type: Divide and Conquer

# Longest Common Subsequence

Type: Dynamic

Create the 2 2D arrays(Weight, Path). 1Dim for input 1, 2Dim for input 2.

We fill in the Weight array with the comparison value and the Path array with the direction.

Create recursive function that uses the direction array and the lengths to navigate the value array.

JAVA

Store is length+1 fill with 0  
Path is length

Comparison starts with 1 and paths is iterator-1

Equal values we +1 diagonal

[i-1,j] >= [i, j-1] copy [i-1,j] back

Else [i, j-1] up

# Longest Simple Path

Type: Dynamic

# Longest Palindrome

Type: Dynamic

# String Transform

Type: Dynamic

# Minimal

Type: Greedy

# Task Assignment

Type: Greedy

Sort task and assign extremes

# Linked List Cycle

Have 2 pointers slow and fast slow = head.next and fast = head.next.next 1 reaches the end first or both meet.

Data Structures

# Primitives

Int, char, double, boolean

# Array

Basic Data structure. Holds an order group of data.

Retrieve and update is an O(1).

Checking full array is O(n) where n is the size.

## Java

Arrays util

new int[]{1,2,3};

new int[][] //2D

# String

# Sets

Each element is represented by a key.

Search

Insert

Delete

Min

Max

Successor

Predecessor

# Stacks

Last In - First Out

Push(Insert)

Pop(Delete)

Peek(Search)

# Queue

First In – First out

Enqueue(Insert)

Dequeue(Delete)

Head and Tail (Properties)

# Linked List

Holds a Data unit and the link to the next node.

# Hash Tables

# Heap

# BST (Binary Search Trees)

Sorting

Quick

Insertion

Heap

Merge

Algorithms

# Divide and Conquer

Divide the problem into a number of subproblems that are smaller instances of the same problem.

Solve the subproblems recursively. If the subproblems are small enough solve them straight forward.

Combine the solutions.

# Greedy

Optimizations from a natural set of choices. Not always the optimum solution, but can provide insight.

# Dynamic

Counting and decisions. Make choices, optimization while building smaller problems.

1. Find optimal solution structure
2. Recursive define the value
3. Compute value
4. Construct

Usually recursive, built bottom-up

Resources

<https://yangshun.github.io/front-end-interview-handbook/>

<https://github.com/h5bp/Front-end-Developer-Interview-Questions>

<https://www.kitchensoap.com/2012/10/25/on-being-a-senior-engineer/>