DSA 50 Days Challenge: Clear Algorithms & Pseudocodes

Marie Ange MALLA NDASSI

Day 1: Two Sum

```
Input: array nums, integer target
For i from 0 to length(nums)-1:
For j from i+1 to length(nums)-1:
If nums[i] + nums[j] = target:
Return (i, j)
Return "No solution"
```

Day 2: Maximum Subarray Sum

```
Input: array nums
currentSum = nums[0], maxSum = nums[0]
For i from 1 to length(nums)-1:
currentSum = max(nums[i], currentSum + nums[i])
maxSum = max(maxSum, currentSum)
Return maxSum
```

Day 3: Sort Colors

```
Input: array nums with values {0,1,2}
low = 0, mid = 0, high = length(nums)-1
While mid <= high:
If nums[mid] = 0:
Swap(nums[low], nums[mid]); low++; mid++
Else if nums[mid] = 1:
mid++
Else:
Swap(nums[mid], nums[high]); high--
Return nums
```

Day 4: 4Sum

```
Input: array nums, integer target
Sort nums
For i from 0 to n-4:
For j from i+1 to n-3:
left = j+1, right = n-1
While left < right:
sum = nums[i]+nums[j]+nums[left]+nums[right]
If sum = target:
Save quadruplet
left++, right--
Else if sum < target: left++
Else: right--
Return all quadruplets
```

Day 5: Merge Intervals

```
Input: list of intervals [start,end]
Sort intervals by start
merged = []
For interval in intervals:
If merged empty OR merged.last.end < interval.start:
Append interval
Else:
merged.last.end = max(merged.last.end, interval.end)
Return merged
```

Day 6: Valid Parentheses String

```
Input: string s
stack = []
mark = array false
For i in 0..len(s)-1:
If s[i] = '(':
push i
Else if s[i] = ')':
If stack not empty: pop
Else: mark[i] = true
While stack not empty: mark[pop] = true
Build result skipping marked indices
Return result
```

Day 7: Sort Characters by Frequency

```
Input: string s
Count frequency of each char
Sort chars by frequency descending
result = ""
For each (char,freq): append char repeated freq times
Return result
```

Day 8: Permutation in String

```
Input: s1, s2
If len(s1) > len(s2): return false
need = freq(s1)
have = freq window in s2
Slide window of size len(s1):
If have = need: return true
Return false
```

Day 9: Palindrome Partitioning

```
Input: string s
Backtrack(start):
If start = len(s): save path
For end = start..len(s)-1:
If substring(start,end) is palindrome:
Add substring to path
Backtrack(end+1)
Remove substring
Return all partitions
```

Day 10: Minimum Window Substring

```
Input: s, t
need = freq(t), have = {}
formed = 0, required = unique chars in t
left=0, bestLen=inf, bestStart=0
For right in 0..len(s)-1:
Add s[right] to have
If have[c]=need[c]: formed++
While formed=required:
Update bestLen
Remove s[left] from have
If have[c]<need[c]: formed--
left++
Return best substring or ""
```

Day 11: Remove Linked List Elements

```
Input: head, val
dummy.next=head
prev=dummy, cur=head
While cur:
If cur.val=val: prev.next=cur.next
Else: prev=cur
cur=cur.next
Return dummy.next
```

Day 12: Reverse Linked List

```
prev=null, cur=head
While cur:
next=cur.next
cur.next=prev
prev=cur
cur=next
Return prev
```

Day 13: Subsets

```
result=[[]]
For x in nums:
newSets=[]
For set in result: newSets.append(set+[x])
result += newSets
Return result
```

Day 14: Generate Parentheses

```
Backtrack(path, open, close):
If len(path)=2n: save path
If open<n: backtrack(path+"(",open+1,close)
If close<open: backtrack(path+")",open,close+1)
Start with ("",0,0)
```

Day 15: LRU Cache

```
Use hashmap + doubly linked list
get(key): if not in map return -1
move node to head, return value
```

```
put(key,value):
if key in map: update, move to head
else:
if size=capacity: remove tail
insert new node at head
```

Day 16: First Missing Positive

```
Place each number in correct index if possible
Scan array: first index i where nums[i]!=i+1
Return i+1
```

Day 17: Spiral Matrix

```
top=0,bottom=m-1,left=0,right=n-1
While top<=bottom and left<=right:
Traverse top row
Traverse right col
Traverse bottom row
Traverse left col
Update boundaries
```

Day 18: Valid Sudoku

```
Check each row, column, and 3x3 box
Ensure digits 1-9 appear at most once
Return true if all valid
```

Day 19: Word Search

```
DFS(r,c,idx):
If idx=len(word): return true
If out of bounds or mismatch: return false
Mark visited
Explore 4 neighbors
Unmark
Return true if any DFS succeeds
```

Day 20: Flatten Binary Tree

```
Flatten(node):
If null return
Flatten left, Flatten right
If left exists:
temp=node.right
node.right=node.left
node.left=null
Attach temp at end of new right
```

Day 21: Palindrome Linked List

```
Find middle
Reverse second half
Compare halves
Return true if equal
```

Day 22: Reverse Nodes in k-Group

```
While at least k nodes remain:
Reverse k nodes
Connect groups
Return new head
```

Day 23: Merge Two Sorted Lists

```
dummy, tail=dummy
While list1 and list2:
Attach smaller node
Attach remaining
Return dummy.next
```

Day 24: Add Two Numbers

```
carry = 0
dummy = new node(0)
tail = dummy
While l1 or l2 or carry:
x = (l1 ? l1.val : 0)
y = (l2 ? l2.val : 0)
sum = x + y + carry
carry = sum / 10
digit = sum % 10
```

```
tail.next = new node(digit)
tail = tail.next
If l1: l1 = l1.next
If l2: l2 = l2.next
Return dummy.next
```

Day 25: Swap Nodes in Pairs

```
dummy.next = head
prev = dummy
While prev.next and prev.next.next:
a = prev.next
b = a.next
prev.next = b
a.next = b.next
b.next = a
prev = a
Return dummy.next
```

Day 26: Add Two Numbers (Linked Lists)

```
Same as Day 24 algorithm
```

Day 27: Swap Nodes in Pairs

```
Same as Day 25 algorithm
```

Day 28: Largest Rectangle in Histogram

```
stack = []
maxArea = 0
For i = 0..n:
h = (i=n ? 0 : heights[i])
While stack not empty and h < heights[stack.top]:
top = stack.pop()
width = stack empty ? i : i - stack.top - 1
maxArea = max(maxArea, heights[top]*width)
push i
Return maxArea
```

Day 29: Min Stack

```
Use two stacks: main, min
push(x): main.push(x), min.push(min.empty?x:min(x,min.top))
pop(): main.pop(), min.pop()
top(): return main.top()
getMin(): return min.top()
```

Day 30: Stack Using Queues

```
Use two queues q1,q2
push(x): enqueue x to q2, move all from q1 to q2, swap q1,q2
pop(): dequeue q1
top(): front(q1)
empty(): q1 empty?
```

Day 31: BST Iterator

```
stack = []
pushAllLeft(root)
next():
node = stack.pop()
pushAllLeft(node.right)
return node.val
hasNext(): stack not empty
```

Day 32: Trapping Rain Water

```
left=0,right=n-1,leftMax=0,rightMax=0,water=0
While left<=right:
If height[left]<=height[right]:
leftMax=max(leftMax,height[left])
water+=leftMax-height[left]
left++
Else:
rightMax=max(rightMax,height[right])
water+=rightMax-height[right]
right--
Return water
```

Day 33: Maximum Depth of Binary Tree

```
If root=null: return 0
Return 1+max(depth(root.left),depth(root.right))
```

Day 34: Lowest Common Ancestor

```
If root=null or root=p or root=q: return root
left=LCA(root.left,p,q)
right=LCA(root.right,p,q)
If left and right: return root
Else return left?left:right
```

Day 35: Kth Smallest in BST

```
stack=[]
node=root
While stack not empty or node:
While node: push node; node=node.left
node=stack.pop(); k--
If k=0: return node.val
node=node.right
```

Day 36: Level Order Traversal

```
queue=[root], result=[]
While queue:
level=[]
For size times:
node=dequeue
level.append(node.val)
enqueue children
result.append(level)
Return result
```

Day 37: Sum Root-to-Leaf Numbers

```
dfs(node,current):
If node=null: return 0
current=current*10+node.val
If leaf: return current
return dfs(node.left,current)+dfs(node.right,current)
Return dfs(root,0)
```

Day 38: Subtree of Another Tree

```
isSame(a,b):
If both null: return true
If one null or a.val!=b.val: return false
return isSame(a.left,b.left) and isSame(a.right,b.right)
isSubtree(r,s):
If r=null: return false
If isSame(r,s): return true
return isSubtree(r.left,s) or isSubtree(r.right,s)
```

Day 39: Implement Trie

```
Node: children[26], isEnd
insert(word): traverse chars, create nodes, mark end
search(word): traverse, return cur.isEnd
startsWith(prefix): traverse, return true if path exists
```

Day 40: Group Anagrams

```
map={}
For s in strs:
key=sorted(s)
map[key].append(s)
Return values(map)
```

Day 41: Subtree Check (reuse Day 38)

```
Same as Day 38
```

Day 42: Trie (reuse Day 39)

```
Same as Day 39
```

Day 43: Bipartite Graph

```
color=-1 for all
For each node:
If color[node]=-1:
BFS assign colors
If conflict: return false
Return true
```

Day 44: Flood Fill

```
old=image[sr][sc]
If old=newColor: return image
dfs(r,c):
If out of bounds or image[r][c]!=old: return
image[r][c]=newColor
dfs neighbors
Call dfs(sr,sc)
```

Day 45: Number of Islands

```
count=0
dfs(r,c):
If out of bounds or grid[r][c]='0': return
grid[r][c]='0'
dfs neighbors
For each cell:
If grid[r][c]='1': count++; dfs(r,c)
Return count
```

Day 46: Clone Graph

```
map={}
clone(u):
If u=null: return null
If u in map: return map[u]
copy=new Node(u.val)
map[u]=copy
For v in u.neighbors: copy.neighbors.append(clone(v))
return copy
Return clone(node)
```

Day 47: Longest Increasing Path in Matrix

```
memo[r][c]=0
dfs(r,c):
If memo[r][c]!=0: return memo[r][c]
best=1
For neighbor with larger value:
best=max(best,1+dfs(neighbor))
memo[r][c]=best
return best
ans=max(dfs(r,c) for all cells)
```

Day 48: Maximum Product Subarray

```
maxProd=minProd=ans=nums[0]
For i=1..n-1:
x=nums[i]
candidates={x,maxProd*x,minProd*x}
maxProd=max(candidates)
minProd=min(candidates)
ans=max(ans,maxProd)
Return ans
```

Day 49: Longest Common Subsequence

```
Input: text1 length m, text2 length n
dp[m+1][n+1] = 0
For i = 1..m:
For j = 1..n:
If text1[i-1] = text2[j-1]:
dp[i][j] = dp[i-1][j-1] + 1
Else:
dp[i][j] = max(dp[i-1][j], dp[i][j-1])
Return dp[m][n]
```

Day 50: Unique Paths

```
Input: m, n
dp[m][n]
dp[0][0] = 1
For i = 0..m-1:
For j = 0..n-1:
If (i,j) != (0,0):
dp[i][j] = (i>0 ? dp[i-1][j] : 0) + (j>0 ? dp[i][j-1] : 0)
Return dp[m-1][n-1]
```