## **Sorting Algorithms**

### **1. Bubble Sort**

**Description:** Bubble Sort repeatedly steps through the list, compares adjacent pairs, and swaps them if they are in the wrong order. Each pass "bubbles" the largest unsorted element to its correct position at the end.

**Complexities:**

* Worst/Average: O(n²)
* Best (already sorted): O(n)
* Space: O(1)
* Stable: Yes

**Pseudocode:**

for i from 0 to n-2:

swapped = false

for j from 0 to n-2-i:

if A[j] > A[j+1]:

swap A[j], A[j+1]

swapped = true

if not swapped: break

**Diagram (array [5,3,8,4]):**

sequenceDiagram

participant A as A[0]=5

participant B as A[1]=3

participant C as A[2]=8

participant D as A[3]=4

Note over A,B: Compare 5>3? Yes, swap

B->>A: Swap 3,5

Note over A,B: Now A=[3,5,8,4]

Note over B,C: Compare 5>8? No

Note over C,D: Compare 8>4? Yes, swap

D->>C: Swap 4,8

Note over C,D: Now A=[3,5,4,8]

### **2. Insertion Sort**

**Description:** Insertion Sort builds the sorted section one element at a time by taking each new element and inserting it into its correct position among the already-sorted elements.

**Complexities:**

* Worst/Average: O(n²)
* Best (nearly sorted): O(n)
* Space: O(1)
* Stable: Yes

**Pseudocode:**

for i from 1 to n-1:

key = A[i]

j = i-1

while j>=0 and A[j]>key:

A[j+1] = A[j]

j--

A[j+1] = key

**Diagram (inserting 4 into [1,3,5,7]):**

graph LR

subgraph Sorted

A1[1] --> A3[3] --> A5[5] --> A7[7]

end

subgraph Key

K4[4]

end

A5 -->|Shift| A7'

A3 -->|Shift| A5'

K4 -->|Insert| A5

### **3. Selection Sort**

**Description:** Selection Sort repeatedly selects the minimum element from the unsorted portion and swaps it with the first unsorted element, growing the sorted section at the front.

**Complexities:**

* Worst/Average/Best: O(n²)
* Space: O(1)
* Stable: No (can be made stable with extra work)

**Pseudocode:**

for i from 0 to n-2:

minIdx = i

for j from i+1 to n-1:

if A[j]<A[minIdx]: minIdx=j

swap A[i], A[minIdx]

**Diagram (array [29,10,14,37,13]):**

graph LR

subgraph Unsorted

U29[29] --> U10[10] --> U14[14] --> U37[37] --> U13[13]

end

U10 -->|Select min| U10[10]

U10 -->|Swap with first| U29

Note over U10,U29: Now [10,29,14,37,13]

### **4. Merge Sort**

**Description:** Merge Sort uses a divide-and-conquer strategy: split the array in halves, recursively sort each half, then merge the sorted halves.

**Complexities:**

* Worst/Average/Best: O(n log n)
* Space: O(n)
* Stable: Yes

**Pseudocode:**

function mergeSort(A, l, r):

if l < r:

m = (l+r)//2

mergeSort(A, l, m)

mergeSort(A, m+1, r)

merge(A, l, m, r)

**Diagram (splitting [38,27,43,3,9,82,10]):**

tree

A[38,27,43,3,9,82,10]

A --> B[38,27,43]

A --> C[3,9,82,10]

B --> D[38]

B --> E[27,43]

E --> E1[27]

E --> E2[43]

C --> F[3,9]

C --> G[82,10]

G --> G1[82]

G --> G2[10]

%% Merges omitted for brevity

### **5. Quick Sort**

**Description:** Quick Sort picks a pivot, partitions the array so elements < pivot go left and > pivot go right, then recursively sorts subarrays.

**Complexities:**

* Average: O(n log n)
* Worst (bad pivot): O(n²)
* Space: O(log n) average
* Stable: No

**Pseudocode:**

function quickSort(A, low, high):

if low<high:

pi = partition(A, low, high)

quickSort(A, low, pi-1)

quickSort(A, pi+1, high)

**Partition Diagram (pivot=10 on [10,7,8,9,1,5]):**

sequenceDiagram

participant S as Start

participant L as LeftIndex

participant R as RightIndex

Note over S: pivot=5

L->>R: Compare A[j] and pivot

R->>S: Swap if < pivot

### **6. Heap Sort**

**Description:** Heap Sort builds a max-heap from the array, then repeatedly extracts the max (root) into the sorted portion by swapping with the last element and heapifying.

**Complexities:**

* Worst/Average/Best: O(n log n)
* Space: O(1)
* Stable: No

**Pseudocode:**

buildMaxHeap(A)

for i from n-1 downto 1:

swap A[0], A[i]

heapify(A, 0, i)

**Heap Diagram (array [4,10,3,5,1]):**

tree

A[10]

A --> B[5]

A --> C[3]

B --> D[4]

B --> E[1]

### **7. Counting Sort**

**Description:** Counting Sort counts occurrences of each distinct integer in a range, computes prefix sums to determine positions, and places elements in output array.

**Complexities:**

* O(n + k) where k = range of input
* Space: O(n + k)
* Stable: Yes

**Pseudocode:**

find min,max in A

d = max-min+1

create count[d], output[n]

for each x in A: count[x-min]++

for i in 1..d-1: count[i]+=count[i-1]

for i from n-1 downto 0:

output[count[A[i]-min]-1] = A[i]

count[A[i]-min]--

**Count Array Diagram ([4,2,2,8,3,3,1]):**

Value: 1 2 3 4 8

Count: 1 2 2 1 1

Prefix:1 3 5 6 7

## **Binary Search Tree Operations**

### **Insertion & Search**

**Description:** A BST maintains left child < parent < right child. Insertion/search follow root-to-leaf comparisons.

**Diagram (insert 65 into BST):**

tree

50 --> 30

50 --> 70

30 --> 20

30 --> 40

70 --> 60

70 --> 80

50 -->|65?| 70

70 -->|<65?| 60

60 -->| >65?| NULL (insert here)

**Inorder Traversal** yields sorted keys.

## **Huffman Coding**

**Description:** Builds an optimal prefix code by repeatedly extracting two minimum-frequency nodes from a min-heap, merging into a new node, and reinserting.

**Diagram:**

tree

"(root)" --> "f:100"

"(root)" --> "g:200"

root --> left[combined]

combined --> a:45

combined --> b:13

## **Graph Traversal**

### **DFS & BFS on Adjacency List**

**Description:**

* **DFS** uses recursion or a stack to explore as deep as possible before backtracking.
* **BFS** uses a queue to explore neighbors level by level.

**Graph Example:**

graph TD

0 -- 1

0 -- 2

1 -- 3

1 -- 4

**DFS Order:** 0 → 1 → 3 → 4 → 2  
 **BFS Order:** 0 → 1 → 2 → 3 → 4

## **Minimum Spanning Tree (MST)**

### **Prim’s Algorithm**

**Description:** Starts from a root, repeatedly adds the minimum-weight edge connecting the growing MST to a new vertex.

**Diagram:**

graph TD

A -- 2 --> B

A -- 3 --> C

B -- 1 --> C

C -- 4 --> D

C -- 5 --> E

%% Highlight edges A--2--B, B--1--C, C--4--D as MST

### **Kruskal’s Algorithm**

**Description:** Sorts all edges by weight and adds them in increasing order if they don’t form a cycle (using Union-Find).

**Diagram:**

graph TD

A -- 1 --> B

B -- 2 --> C

A -- 3 --> C

C -- 4 --> D

%% Pick A-B, B-C, C-D