In [8]:

```
import random

# 1) Bubble Sort O(n^2)
def bubble_sort(a):
    a = a[:]
    n = len(a)
    for i in range(n):
        swapped = False
        for j in range(0, n-1-i):
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
                swapped = True
        if not swapped:
            break
    return a
```

In [9]:

```
# Quick Sort helpers
def _partition(a, lo, hi, pivot_idx):
    a[pivot_idx], a[hi] = a[hi], a[pivot_idx]
    pivot = a[hi]
    i = lo
    for j in range(lo, hi):
        if a[j] <= pivot:
            a[i], a[j] = a[j], a[i]
            i += 1
    a[i], a[hi] = a[hi], a[i]
    return i

def _median_of_three_index(a, lo, hi):
    mid = (lo + hi) // 2
    x, y, z = a[lo], a[mid], a[hi]
    if (x <= y <= z) or (z <= y <= x): return mid
    if (y <= x <= z) or (z <= x <= y): return lo
    return hi
```

In [10]:

```
# 3a) Quick Sort: random pivot
def quick_sort_random_pivot(a):
    a = a[:]
    def qs(lo, hi):
        if lo >= hi: return
        pidx = random.randint(lo, hi)
        p = _partition(a, lo, hi, pidx)
        qs(lo, p-1); qs(p+1, hi)
    qs(0, len(a)-1)
    return a

# 3b) Quick Sort: "average pivot (down + middle + up)" -> median-of-three
def quick_sort_avg_pivot(a):
    a = a[:]
    def qs(lo, hi):
        if lo >= hi: return
        pidx = _median_of_three_index(a, lo, hi)
        p = _partition(a, lo, hi, pidx)
        qs(lo, p-1); qs(p+1, hi)
    qs(0, len(a)-1)
    return a
```

In [11]:

```
# 4) Merge Sort
def merge_sort(a):
    def merge(L, R):
        out = []
        i = j = 0
        while i < len(L) and j < len(R):
            if L[i] <= R[j]:
                out.append(L[i]); i += 1
            else:
                out.append(R[j]); j += 1
        out.extend(L[i:]); out.extend(R[j:])
        return out

    def ms(arr):
        if len(arr) <= 1: return arr
        m = len(arr)//2
        return merge(ms(arr[:m]), ms(arr[m:]))

    return ms(a[:])
```

In [12]:

```
# 5) Heap Sort
def heap_sort(a):
    a = a[:]
    n = len(a)

    def heapify(i, heap_size):
        while True:
            largest = i
            l, r = 2*i+1, 2*i+2
            if l < heap_size and a[l] > a[largest]: largest = l
            if r < heap_size and a[r] > a[largest]: largest = r
            if largest == i: break
            a[i], a[largest] = a[largest], a[i]
            i = largest

    for i in range(n//2 - 1, -1, -1):
        heapify(i, n)

    for end in range(n-1, 0, -1):
        a[0], a[end] = a[end], a[0]
        heapify(0, end)

    return a
```

Tests (various sizes)

- Edge cases: empty, single element, duplicates, already sorted, reverse sorted
- Random tests: sizes 0, 1, 2, 5, 10, 50, 200 (30 trials each)
- Correctness: compare each algorithm output with Python `sorted()`

In [13]:

```
#Test
def run_tests():
    algos = [
        bubble_sort,
        quick_sort_random_pivot,
        quick_sort_avg_pivot,
        merge_sort,
        heap_sort,
    ]

    test_arrays = [
        [], [1], [2,1],
        [1,2,3,4], [4,3,2,1],
        [3,1,2,3,3,0,-1]
    ]

    for n in [0,1,2,5,10,50,200]:
        for _ in range(30):
            test_arrays.append([random.randint(-1000,1000) for _ in range(n)])

    for f in algos:
        for arr in test_arrays:
            assert f(arr) == sorted(arr), f"Failed {f.__name__} on {arr}"

    return "All tests passed ✅"

run_tests()
```

```
Out[13]: 'All tests passed ✅'
```

Problem 2 — Complexity Analysis

- **Bubble Sort:** worst/avg O(n^2), best O(n) (early stop), space O(1)
- **Quick Sort (random pivot):** expected O(n log n), worst O(n^2), space expected O(log n)
- **Quick Sort (avg pivot: down+middle+up / median-of-three):** typically O(n log n), worst O(n^2), space same as quick sort
- **Merge Sort (Master Theorem):** T(n)=2T(n/2)+Θ(n) ⇒ Θ(n log n), space O(n)
- **Heap Sort:** O(n log n), space O(1)