Quicksort 1: pdqsort (Pattern-Defeating Quicksort) Filename: qs_01_pdqsort.cpp

Source: https://github.com/orlp/pdqsort

Key OD: A highly complex hybrid algorithm combining Quicksort, Heapsort, and Insertion Sort with branchless partitioning logic. Serves as the modern, highly-optimized baseline.

Quicksort 2: Dual-Pivot Quicksort Filename: qs_02_dualpivot.cpp

Source: https://github.com/MichaelAxtmann/DualPivotQuicksort

Key OD: Uses two pivots to partition the array into three sections (< P1, P1 < x < P2, > P2). This is a significant structural change from standard single-pivot logic.

Quicksort 3: Classic 2-Way Quicksort Filename: qs_03_2way.cpp

Source: https://github.com/mmcetindag/SortingAlgorithms (Extracted partition2)

Key OD: The standard, baseline partitioning scheme where elements smaller than the pivot go left, and larger elements go right. Serves as the simple control group.

Quicksort 4: 3-Way Quicksort (Dutch National Flag) Filename: qs_04_3way.cpp

Source: https://github.com/mmcetindag/SortingAlgorithms (Extracted partition3)

Key OD: Partitions the array into three explicit sections (< pivot, == pivot, > pivot). Adds conditional logic to the inner loop specifically to handle duplicate elements efficiently.

Quicksort 5: Randomized Quicksort Filename: qs_05_randomizedqcksrt.cpp

Source: https://github.com/srushti-dhakate-2210/RANDOMIZED-QUICKSORT

Key OD: Calls rand() to select a random pivot index before partitioning. Tests the resilience of the random number generator call and non-deterministic logic flow.

Quicksort 6: Iterative Quicksort Filename: qs_06_iterative.cpp

Source: https://github.com/Iofsoft/IterativeQuickSort

Key OD: Replaces recursive function calls with an explicit, manually managed stack (std::vector) and a while loop. Tests stack memory resilience vs heap/recursion resilience.

Quicksort 7: Hoare Partition Quicksort Filename: qs_07_hoare.cpp

Source: https://github.com/A1A1G2/QuickSort/blob/main/quickSort.c (Logic adapted to C++)

Key OD: Uses two pointers starting at both ends and moving inward, swapping elements on the wrong side. Creates a different memory access pattern than Lomuto.

Quicksort 8: Lomuto Partition Quicksort Filename: qs_08_lomuto.cpp

Source: https://github.com/nmantzanas/Quicksort-Lomuto-

Key OD: Uses a single pointer scanning forward. Performs more swaps than Hoare but has simpler logic.

Quicksort 9: Parallel Quicksort Filename: qs_09_parallel.cpp

Source: Generated implementation (Standard std::async logic using Gemini 3.0 pro) to replace the test-suite-only code from https://github.com/GabTux/PPQSort

Key OD: Uses std::async to spawn threads for recursive calls. Tests the resilience of multi-threaded logic and thread management overhead.

Quicksort 10: Median-of-Medians Quicksort Filename: qs_10_median.cpp

Source: https://github.com/aviral92/Quicksort-Using-Median-of-Medians-as-Pivot

Key OD: Uses a complex recursive algorithm to find the exact median pivot, guaranteeing O(N log N) worst-case. Adds significant computational overhead to the pivot selection step.

Quicksort 11: Introsort (Self-Contained) Filename: qs_11_introsort.cpp

Source: Adapted from https://github.com/AlexandruValeanu/Introsort (Dependencies removed)

Key OD: Tracks recursion depth. Switches to Heapsort if depth exceeds 2 log N to prevent worst-case scenarios. Adds a conditional depth check to every recursive step.

Using this main function with the input (attach in file)

```
int main() {
    FILE *fp;
    int arr[50000];
    int n = 0;

    fp = fopen("./numbers.txt", "r");
    if (fp == NULL) {
        printf("Error: cannot open file!\n");
        return 1;
    }

    // read numbers from file
    while (fscanf(fp, "%d", &arr[n]) == 1) {
        n++;
    }
    fclose(fp);

    quicksort(arr, 0, n - 1);

    printf("After sorting:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    return 0;
}
```