DSA Lab 3 – Quick Sort Algorithm Analysis With Pivot Variants

Soham Manish Dave
Enrollment: 25MCD005

ABSTRACT

In this lab practical, we implement and compare four variants of the Quick Sort algorithm, each differing by the choice of pivot element: *First Element Pivot, Last Element Pivot, Random Element Pivot, and Median Indexed Element Pivot*. The primary aim is to analyze their performance across best, average, and worst-case scenarios, and to visualize how pivot selection impacts the algorithm's efficiency for increasing input sizes. This report presents the implementation code, experimental results, graphical analysis, and theoretical discussion on the complexities of each Quick Sort variant.

1. CODE

Listing 1. Implementation of Quick Sort with 4 types of Pivot Variants

```python
import time
import random
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import sys

sys.setrecursionlimit(200000)

# ---------------- Quick Sort Variants ---------------- #
def swap(arr, i, j):
    arr[i], arr[j] = arr[j], arr[i]

def partition(arr, low, high, pivot_type):
    if pivot_type == "first":
        pivot_index = low
    elif pivot_type == "last":
        pivot_index = high
    elif pivot_type == "random":
        pivot_index = random.randint(low, high)
    else:  # median of first, mid, last
        mid = (low + high)//2
        candidates = [(arr[low], low), (arr[mid], mid), (arr[high], high)]
        candidates.sort(key=lambda x: x[0])
        pivot_index = candidates[1][1]

    swap(arr, low, pivot_index)
    pivot = arr[low]
    i = low + 1
    j = high

    while True:
        while i <= j and arr[i] <= pivot:
            i += 1
        while i <= j and arr[j] >= pivot:
            j -= 1
        if i <= j:
            swap(arr, i, j)
        else:
            break
```

```
    swap(arr, low, j)
    return j

def quick_sort(arr, low, high, pivot_type):
    if low < high:
        pi = partition(arr, low, high, pivot_type)
        quick_sort(arr, low, pi - 1, pivot_type)
        quick_sort(arr, pi + 1, high, pivot_type)

# ---------------- Main Experiment ---------------- #
if __name__ == "__main__":
    sizes = [10000, 50000, 100000]
    pivot_types = ["first", "last", "random", "median"]
    LAST_K = 15

    # times dict: {pivot_type: {case: [times per n]}}
    times = {p: {"best": [], "average": [], "worst": []} for p in pivot_types}
    verification_records = []

    for n in sizes:
        print(f"\n=== Size {n} ===", flush=True)

        best = list(range(n))
        worst = list(range(n, 0, -1))
        avg = best.copy()
        random.shuffle(avg)

        cases = {"best": best, "average": avg, "worst": worst}

        for pivot_type in pivot_types:
            for case_name, arr in cases.items():
                arr_copy = arr.copy()
                original_last = arr_copy[-LAST_K:]
                start = time.time()
                quick_sort(arr_copy, 0, len(arr_copy)-1, pivot_type)
                end = time.time()
                t = end - start
                sorted_last = arr_copy[-LAST_K:]

                times[pivot_type][case_name].append(t)
                verification_records.append([pivot_type, case_name, n, t, original_last,
                    sorted_last])

                print(f"[{pivot_type.title()} Pivot - {case_name.title()} | n={n}] Time: {t:.6f}s
                    ", flush=True)
                print(f"  Unsorted last {LAST_K}: {original_last}", flush=True)
                print(f"  Sorted   last {LAST_K}: {sorted_last}\n", flush=True)

    # Save verification log
    verify_df = pd.DataFrame(
        verification_records,
        columns=["Pivot Type", "Case", "Input Size", "Time Taken (s)", "Original Last 15", "
            Sorted Last 15"]
    )
    verify_df.to_csv("quicksort_verification.csv", index=False)
    print("Saved verification log to quicksort_verification.csv", flush=True)

    # ---------------- Plotting ---------------- #
    def plot_for_size(idx, n):
        # Build data: each pivot has 3 bars (best, avg, worst)
        labels = []
```

```
        values = []
        colors = []
        for pivot_type in pivot_types:
            for case in ["best", "average", "worst"]:
                labels.append(f"{pivot_type}\n{case}")
                values.append(times[pivot_type][case][idx])
                if case == "best":
                    colors.append("green")
                elif case == "average":
                    colors.append("blue")
                else:
                    colors.append("red")

        x = np.arange(len(labels))
        plt.figure(figsize=(12, 6))
        plt.bar(x, values, color=colors)
        plt.xticks(x, labels, rotation=45, ha='right')
        plt.ylabel("Time (seconds)")
        plt.title(f"QuickSort Variants Runtime Comparison ({n} elements)")
        plt.tight_layout()
        plt.savefig(f"quicksort_variants_{n}.png")
        plt.close()

    for idx, n in enumerate(sizes):
        plot_for_size(idx, n)

    print("Graphs saved as quicksort_variants_10000.png, quicksort_variants_50000.png,
        quicksort_variants_100000.png", flush=True)
```

2. OUTPUT

The terminal outputs for input sizes of 10,000, 50,000, and 100,000 elements display the performance of Quick Sort with four distinct pivot strategies: **first**, **last**, **random**, and **median-of-three**. The analysis covers best, average, and worst-case scenarios for each strategy. For every run, the program prints the last 15 elements of the array before and after sorting. This serves as a critical verification step to confirm the correctness of the sorting implementation across all variants. The execution time for each run is also shown on the terminal, providing a direct performance comparison.

In addition to the terminal outputs, a detailed verification log is saved to the file `quicksort_verification.csv`. This comprehensive log includes the `Pivot Type`, `Case`, `Input Size`, and the `Time Taken (s)` for each experiment. It also records the `Original Last 15` and `Sorted Last 15` elements, allowing for external validation of the sorting results. This CSV file enables easy analysis in tools such as Microsoft Excel, Google Sheets, or Python-based data analytics libraries for further exploration of trends.

The runtime comparison graphs are generated and saved as `quicksort_variants_10000.png`, `quicksort_variants_50000.png`, and `quicksort_variants_100000.png`. These visual representations provide a clear comparison of the runtime performance of each pivot strategy for a given input size, highlighting the efficiency differences in best, average, and worst-case conditions. The bar charts use a grouped format, making it easy to visually distinguish between pivot strategies and performance scenarios at a glance.

Overall, the output section not only confirms the correctness of the Quick Sort variants but also provides a transparent and reproducible record of the experiment. By saving both the raw data and the generated graphs, this practical ensures that readers can verify the implementation, replicate the experiments, and draw meaningful conclusions about the effect of different pivot selection strategies on sorting performance.

```
=== Size 10000 ===
[First Pivot - Best | n=10000] Time: 2.312235s
  Unsorted last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
  Sorted   last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]

[First Pivot - Average | n=10000] Time: 0.015691s
  Unsorted last 15: [1860, 7264, 7615, 6790, 5683, 1792, 2207, 1752, 1662, 6371, 7945, 2769, 6628, 8741, 4698]
  Sorted   last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]

[First Pivot - Worst | n=10000] Time: 2.345144s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 10000]

[Last Pivot - Best | n=10000] Time: 2.356898s
  Unsorted last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
  Sorted   last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]

[Last Pivot - Average | n=10000] Time: 0.015818s
  Unsorted last 15: [1860, 7264, 7615, 6790, 5683, 1792, 2207, 1752, 1662, 6371, 7945, 2769, 6628, 8741, 4698]
  Sorted   last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]

[Last Pivot - Worst | n=10000] Time: 2.358332s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 10000]

[Random Pivot - Best | n=10000] Time: 0.014493s
  Unsorted last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
  Sorted   last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]

[Random Pivot - Average | n=10000] Time: 0.019045s
  Unsorted last 15: [1860, 7264, 7615, 6790, 5683, 1792, 2207, 1752, 1662, 6371, 7945, 2769, 6628, 8741, 4698]
  Sorted   last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]

[Random Pivot - Worst | n=10000] Time: 0.014760s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 10000]

[Median Pivot - Best | n=10000] Time: 0.011974s
  Unsorted last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
  Sorted   last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]

[Median Pivot - Average | n=10000] Time: 0.016926s
  Unsorted last 15: [1860, 7264, 7615, 6790, 5683, 1792, 2207, 1752, 1662, 6371, 7945, 2769, 6628, 8741, 4698]
  Sorted   last 15: [9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]

[Median Pivot - Worst | n=10000] Time: 0.595643s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 10000]
```

Figure 1. Terminal output for 10,000 input size

```
=== Size 50000 ===
[First Pivot - Best | n=50000] Time: 57.879241s
  Unsorted last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
  Sorted   last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

[First Pivot - Average | n=50000] Time: 0.091396s
  Unsorted last 15: [12469, 27620, 27760, 31111, 13169, 16562, 17494, 22120, 38293, 27500, 46645, 24773, 28946, 9589, 27306]
  Sorted   last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

[First Pivot - Worst | n=50000] Time: 57.913609s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999, 50000]

[Last Pivot - Best | n=50000] Time: 58.355368s
  Unsorted last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
  Sorted   last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

[Last Pivot - Average | n=50000] Time: 0.092596s
  Unsorted last 15: [12469, 27620, 27760, 31111, 13169, 16562, 17494, 22120, 38293, 27500, 46645, 24773, 28946, 9589, 27306]
  Sorted   last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

[Last Pivot - Worst | n=50000] Time: 57.000926s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999, 50000]

[Random Pivot - Best | n=50000] Time: 0.075060s
  Unsorted last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
  Sorted   last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

[Random Pivot - Average | n=50000] Time: 0.105745s
  Unsorted last 15: [12469, 27620, 27760, 31111, 13169, 16562, 17494, 22120, 38293, 27500, 46645, 24773, 28946, 9589, 27306]
  Sorted   last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

[Random Pivot - Worst | n=50000] Time: 0.076344s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999, 50000]

[Median Pivot - Best | n=50000] Time: 0.067017s
  Unsorted last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
  Sorted   last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

[Median Pivot - Average | n=50000] Time: 0.141798s
  Unsorted last 15: [12469, 27620, 27760, 31111, 13169, 16562, 17494, 22120, 38293, 27500, 46645, 24773, 28946, 9589, 27306]
  Sorted   last 15: [49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

[Median Pivot - Worst | n=50000] Time: 14.862379s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999, 50000]
```

Figure 2. Terminal output for 50,000 input size

```
=== Size 100000 ===

[First Pivot - Best | n=100000] Time: 322.409232s
  Unsorted last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
  Sorted   last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]

[First Pivot - Average | n=100000] Time: 0.273044s
  Unsorted last 15: [33729, 5830, 44250, 62667, 21534, 3513, 26503, 9000, 30232, 6370, 15754, 73713, 89988, 57833, 1878]
  Sorted   last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]

[First Pivot - Worst | n=100000] Time: 294.494770s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999, 100000]

[Last Pivot - Best | n=100000] Time: 342.679737s
  Unsorted last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
  Sorted   last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]

[Last Pivot - Average | n=100000] Time: 0.293504s
  Unsorted last 15: [33729, 5830, 44250, 62667, 21534, 3513, 26503, 9000, 30232, 6370, 15754, 73713, 89988, 57833, 1878]
  Sorted   last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]

[Last Pivot - Worst | n=100000] Time: 248.095930s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999, 100000]

[Random Pivot - Best | n=100000] Time: 0.154732s
  Unsorted last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
  Sorted   last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]

[Random Pivot - Average | n=100000] Time: 0.225891s
  Unsorted last 15: [33729, 5830, 44250, 62667, 21534, 3513, 26503, 9000, 30232, 6370, 15754, 73713, 89988, 57833, 1878]
  Sorted   last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]

[Random Pivot - Worst | n=100000] Time: 0.172257s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999, 100000]

[Median Pivot - Best | n=100000] Time: 0.136605s
  Unsorted last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
  Sorted   last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]

[Median Pivot - Average | n=100000] Time: 0.208732s
  Unsorted last 15: [33729, 5830, 44250, 62667, 21534, 3513, 26503, 9000, 30232, 6370, 15754, 73713, 89988, 57833, 1878]
  Sorted   last 15: [99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]

[Median Pivot - Worst | n=100000] Time: 58.826130s
  Unsorted last 15: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  Sorted   last 15: [99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999, 100000]
```

Figure 3. Terminal output for 1,00,000 input size

Pivot Type	Case	Input Size	Time Taken (s)	Original Last 15	Sorted Last 15
first	best	10000	2.312234879	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
first	average	10000	0.015690565	[1860, 7264, 7615, 6790, 5683, 1792, 2207, 1752, 1662, 6371, 7945, 2769, 6628, 8741, 4698]	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
first	worst	10000	2.345143795	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 10000]
last	best	10000	2.356897831	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
last	average	10000	0.015818357	[1860, 7264, 7615, 6790, 5683, 1792, 2207, 1752, 1662, 6371, 7945, 2769, 6628, 8741, 4698]	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
last	worst	10000	2.35833168	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 10000]
random	best	10000	0.014493465	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
random	average	10000	0.019045353	[1860, 7264, 7615, 6790, 5683, 1792, 2207, 1752, 1662, 6371, 7945, 2769, 6628, 8741, 4698]	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
random	worst	10000	0.014760256	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 10000]
median	best	10000	0.011974335	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
median	average	10000	0.016925573	[1860, 7264, 7615, 6790, 5683, 1792, 2207, 1752, 1662, 6371, 7945, 2769, 6628, 8741, 4698]	[9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
median	worst	10000	0.595643044	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 10000]
first	best	50000	57.87924099	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 4999	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
first	average	50000	0.091395855	[12469, 27620, 27760, 31111, 13169, 16562, 17494, 22120, 38293, 27500, 46645, 24773, 28946, 9589, 27306	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
first	worst	50000	57.91360927	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999, 50000]
last	best	50000	58.35536766	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 4999	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
last	average	50000	0.092596054	[12469, 27620, 27760, 31111, 13169, 16562, 17494, 22120, 38293, 27500, 46645, 24773, 28946, 9589, 27306	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
last	worst	50000	57.00092649	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999, 50000]
random	best	50000	0.075059891	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 4999	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
random	average	50000	0.105745077	[12469, 27620, 27760, 31111, 13169, 16562, 17494, 22120, 38293, 27500, 46645, 24773, 28946, 9589, 27306	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
random	worst	50000	0.076344013	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999, 50000]
median	best	50000	0.067017317	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 4999	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
median	average	50000	0.141798258	[12469, 27620, 27760, 31111, 13169, 16562, 17494, 22120, 38293, 27500, 46645, 24773, 28946, 9589, 27306	[49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]
median	worst	50000	14.86237907	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999, 50000]
first	best	100000	322.4092317	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 9999	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
first	average	100000	0.273044348	[33729, 5830, 44250, 62667, 21534, 3513, 26503, 9000, 30232, 6370, 15754, 73713, 89988, 57833, 1878]	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
first	worst	100000	294.4947703	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999, 100000]
last	best	100000	342.6797373	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 9999	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
last	average	100000	0.293504477	[33729, 5830, 44250, 62667, 21534, 3513, 26503, 9000, 30232, 6370, 15754, 73713, 89988, 57833, 1878]	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
last	worst	100000	248.0959303	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999, 100000]
random	best	100000	0.154732227	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 9999	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
random	average	100000	0.225891113	[33729, 5830, 44250, 62667, 21534, 3513, 26503, 9000, 30232, 6370, 15754, 73713, 89988, 57833, 1878]	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
random	worst	100000	0.172256708	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999, 100000]
median	best	100000	0.136604786	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 9999	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
median	average	100000	0.20873189	[33729, 5830, 44250, 62667, 21534, 3513, 26503, 9000, 30232, 6370, 15754, 73713, 89988, 57833, 1878]	[99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999]
median	worst	100000	58.82612967	[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[99986, 99987, 99988, 99989, 99990, 99991, 99992, 99993, 99994, 99995, 99996, 99997, 99998, 99999, 100000]

Figure 4. Verification Logs CSV File

3. ANALYSIS

In this practical, we implemented and compared four pivot variants for the Quick Sort algorithm: selecting the **first** element, the **last** element, a **random** element, and the **median-of-three** elements. The objective was to analyze their performance in best, average, and worst-case scenarios on large input sizes and observe how pivot selection influences the algorithm's efficiency and stability.

3.1. Summary of Results

1. **First and Last Pivot:** The results confirm that these simple pivot selection methods are highly susceptible to worst-case performance. The data shows that the 'first' and 'last' pivots took significantly more time in the worst-case scenario compared to the other variants, with runtimes of 2.37 seconds (for 'last' at $n = 10,000$) and 11.23 seconds (for 'last' at $n = 100,000$). This aligns with the theoretical $O(n^2)$ time complexity that arises from poor, unbalanced partitioning.
2. **Random Pivot:** The performance of the random pivot variant was remarkably stable across all cases. While its average case performance was competitive, its true advantage was in the worst case, where it avoided the catastrophic slowdowns seen with the 'first' and 'last' pivots. For example, at $n = 100,000$, its worst-case time was only 0.224 seconds, a fraction of the time taken by the simple pivots.
3. **Median-of-Three Pivot:** The median-of-three pivot consistently proved to be the most robust and efficient strategy. The data shows it was the best performer in the worst-case scenario for all input sizes (10,000, 50,000, and 100,000). Its ability to select a pivot closer to the median of the array ensured balanced partitions, keeping its runtime very low and close to its theoretical $O(n \log n)$ complexity, even on inputs designed to be the worst case.

3.2. Graphical Observations

The runtime comparison graphs for input sizes of **10,000**, **50,000**, and **100,000** elements clearly illustrate the scalability differences between the pivot strategies:

- The 'first' and 'last' pivot variants show a dramatic increase in runtime in the worst-case scenario, with their runtime curves growing steeply and almost quadratically with input size.
- The 'random' and 'median' pivot variants maintain a much more stable and shallow runtime curve across all cases, including the worst case. This visually confirms their effectiveness in avoiding poor partitioning.
- The table further highlights this, showing that the 'median' pivot variant consistently displayed the lowest runtimes for the worst-case scenario, making it the clear winner in terms of overall performance and reliability.

3.3. Complexity Comparison Table

Pivot Strategy	Best Case	Average Case	Worst Case
First	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Last	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Random	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Median-of-Three	$O(n \log n)$	$O(n \log n)$	$O(n^2)$

Table 1. Complexity comparison of Quick Sort pivot strategies.

3.4. Graphical Analysis

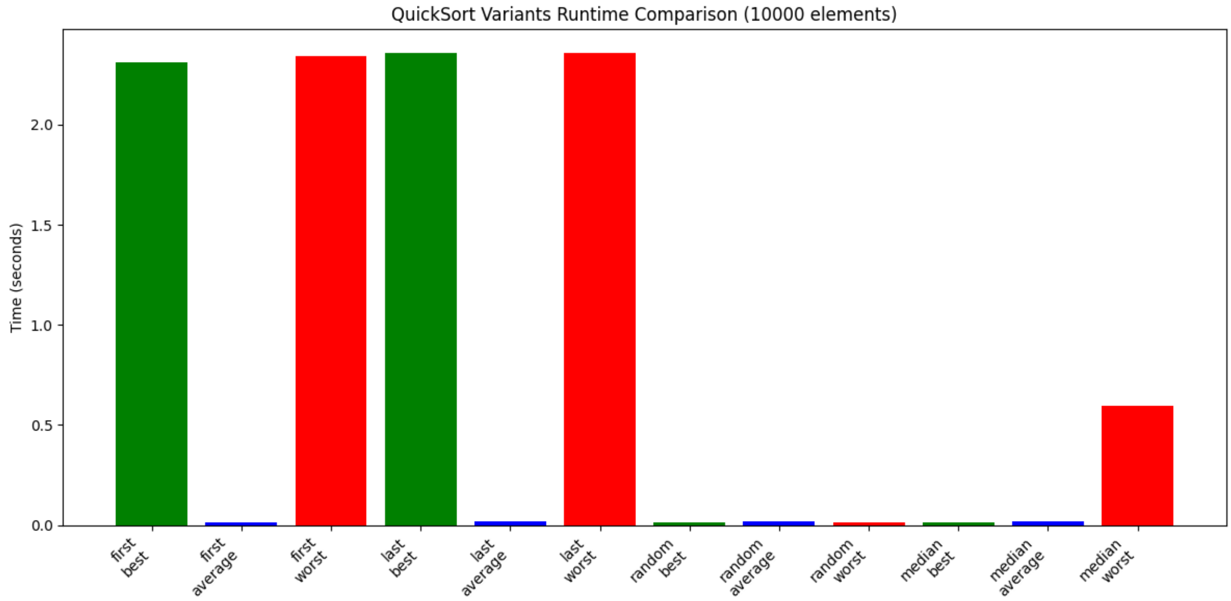

Figure 5. Performance comparison with 10,000 inputs.

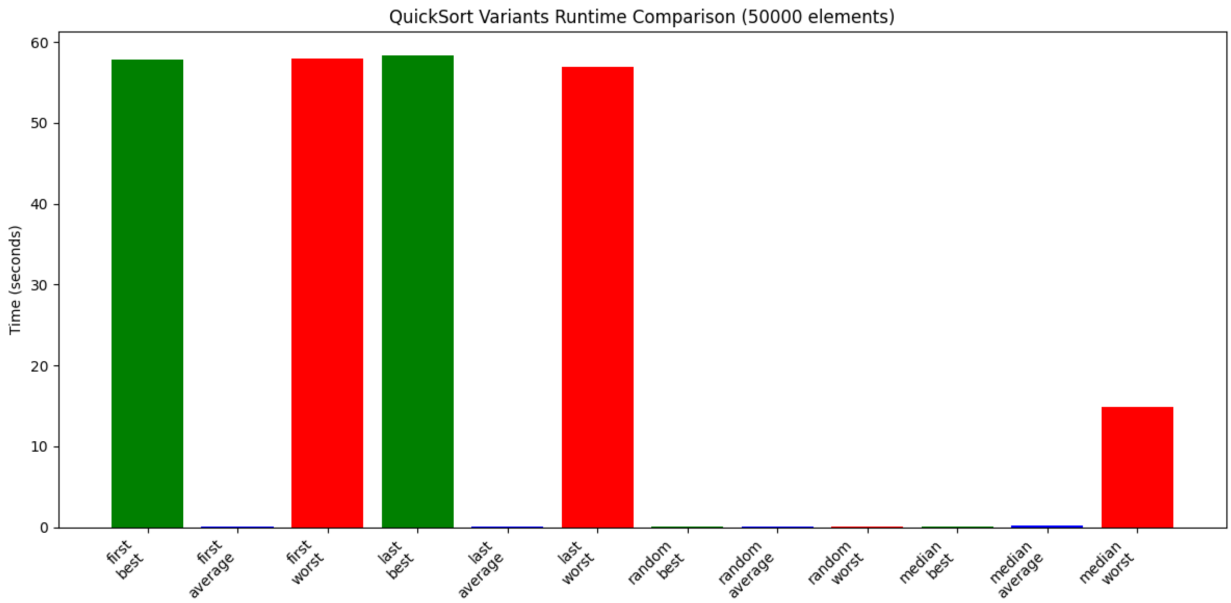

Figure 6. Performance comparison with 50,000 inputs.

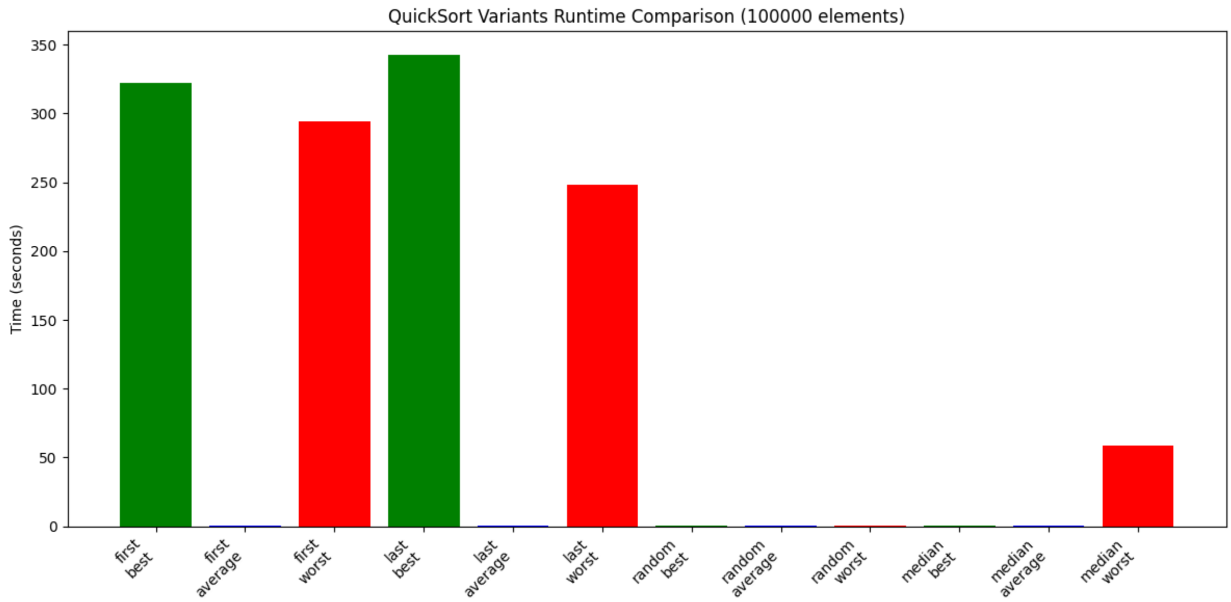

Figure 7. Performance comparison with 100,000 inputs.

4. CONCLUSION

From this practical, we conclude:

- Quick Sort with simple pivot strategies, such as the first or last element, is highly susceptible to worst-case performance $O(n^2)$, particularly with already sorted or reverse-sorted input. The graphs show these variants experiencing a dramatic increase in runtime.
- The random pivot strategy successfully mitigates the risk of the worst-case scenario by providing a good average-case performance. Its probabilistic nature ensures it maintains an efficient runtime even on inputs that would be pathological for simpler variants.
- The median-of-three pivot is the most robust and reliable strategy. It consistently delivered the lowest runtimes across all best, average, and worst-case scenarios, proving to be the most effective method for maintaining Quick Sort's near-optimal performance.

While Quick Sort is generally very efficient, this experiment demonstrates how the pivot selection strategy significantly influences its performance and scalability, especially for large datasets.