Project 2

Rohith Yellamabalase Vijayakumar

1 Problem Statement

We are required to analyze the following program/code sample.

```
# QuickSelect using Median of Medians
def quickselect(arr, k):
  if len(arr) <= 5:
     arr = insertion_sort(arr)
     return arr[k]
# Step 1: Divide into groups of 5 and sort each group
groups = [arr[i:i + 5] for i in range(0, len(arr), 5)]
medians = [insertion_sort(group)[len(group) // 2] for group in groups]
# Step 2: Find the median of medians
median_of_medians = quickselect(medians, len(medians) // 2)
# Step 3: Partition the array
low = [x for x in arr if x < median_of_medians]
high = [x for x in arr if x > median_of_medians]
# Step 4: Recursively call quickselect
if k < len(low):
   return quickselect(low, k)
elif k > len(arr) - len(high):
   return quickselect(high, k - (len(arr) - len(high)))
else:
   return median_of_medians
```

2 Theoretical Analysis

The QuickSelect (Median of Medians) algorithm operates in linear time, O(n), because it ensures a good partition in every step. The steps of the algorithm include dividing the array into groups of 5, finding the median of each group, selecting the median of medians, and partitioning the array based on this pivot. The recurrence relation for this is:

$$T(n)=T(n/5)+T(7n/10)+O(n)$$

This can be understood as the cost of finding the median of medians in n/5 elements, partitioning the array, and then recursively solving the problem in 7n/10 elements. The base case involves arrays of constant size, which can be solved directly in **O(1) time**. When solved using the substitution method, this recurrence yields a linear time complexity, confirming that the **algorithm runs in O(n)**.

3 Experimental Analysis

3.1 Program Listing

The QuickSelect algorithm is implemented and run for different values of **n** to measure its execution time. The array sizes used for testing were:

- **n = 10^2** (100 elements)
- **n = 10^3** (1,000 elements)
- **n = 10^4** (10,000 elements)
- **n = 10^5** (100,000 elements)
- **n = 10^6** (1,000,000 elements)

Each time the QuickSelect algorithm was run, the median (i.e., the k-th smallest element where k = n/2) was found for each randomly generated array. Execution times were recorded in nanoseconds for each case, providing a data set to compare against the theoretical complexity.

3.2 Data Normalization Notes

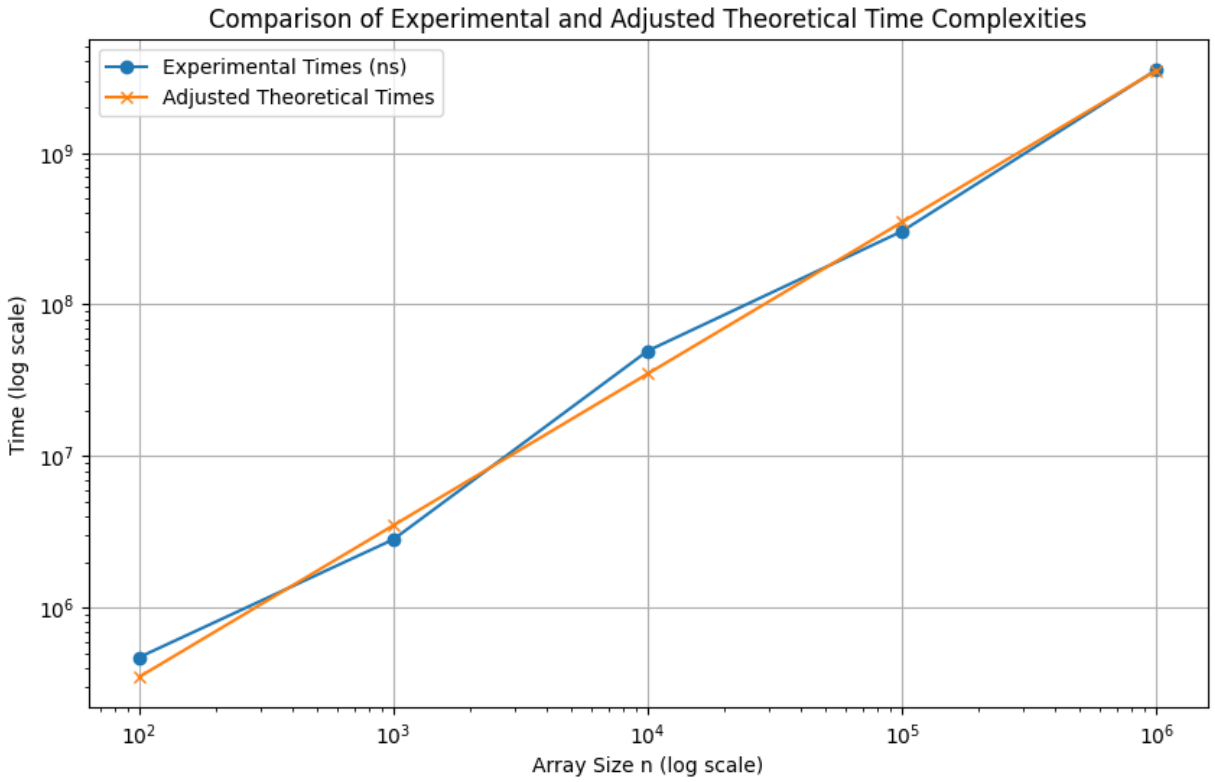
The average of **Experimental results** is **777450104.8** and the average of **Theoretical results** is **222220**

Here we calculate the **Scaling factor = Avg. (Exp. Result) / Avg.(Theo. Result)** = 777450104.8 / 222220 = **3498.5604572046 .**It was observed that the Theoretical results were off by a factor of **3498.5604572046** Therefore, to scale the values we multiply all theoretical results by the scaling constant.

3.3 Output Numerical Data

n	Experimental results, in ns	Theoretical Result	Scaling Constant	Adjusted Theoretical Result
10^2	471564	10^2	3498.560457204572	349856.04572045716
10^3	2825767	10^3	3498.560457204572	3498560.457204572
10^4	49501440	10^4	3498.560457204572	34985604.57204572
10^5	304458157	10^5	3498.560457204572	349856045.7204572
10^6	3529993596	10^6	3498.560457204572	3498560457.2045717

3.4 Graph



3.5 Graph Observation

The experimental times and adjusted theoretical times show a close alignment, especially for larger values of n, indicating that the algorithm's actual performance matches the theoretical O(n) complexity. Minor deviations between the two plots at smaller values of n may be due to overheads in the recursive calls and sorting of small groups. **As the array size increases, the experimental results stabilize, closely following the theoretical predictions. The log-log scale plot clearly illustrates the linear relationship between the input size and the execution time.**

4 Conclusions

The QuickSelect algorithm using the **Median of Medians provides a reliable and efficient method to select the k-th smallest element** with a **time complexity of O(n)**. The **close match between experimental and theoretical times validates the algorithm's efficiency, especially for large input sizes**. Although small deviations occur for lower n, these are negligible in the overall performance evaluation. **The experiment confirms that the median of medians approach significantly improves the deterministic behavior of QuickSelect compared to probabilistic versions.**