**Project Report**

Emmanuel (Manny) David

Problem Definition

This problem is known as the Selection Problem. It's all about finding a particular item in a list. There are a bunch of numbers in a list, and the want is to figure out which number is the ith smallest. When the "size" of the problem is mentioned, it means how many numbers are in the list. The algorithms are going to look at lists ranging from 10,000 to 100,000 numbers, increasing by 10,000 each time, so, we're dealing with both big and small lists. This kind of problem-solving isn't just theoretical; it has real-world uses. For example, in databases, it's like searching for specific information. In stats, it's like finding the middle number or figuring out different rankings based on certain criteria.

Algorithms and RT Analysis

Alg1: Insertion Sort Pseudocode

insertion-sort(A){

for i = 2 to A.length

key = A[i]

j = i - 1

while j > 0 and A[j] > key

A[j + 1] = A[j]

j = j - 1

A[j + 1] = key

}

RT analysis: O(n2) - Using the Insertion Sort technique, ALG1 sorts the array and gives back the ith order statistic. The running time is O(n^2) in the worst-case scenario (when the array is in reverse order), but O(n) in the best-case (when the array is already sorted). Compared to more sophisticated sorting algorithms, it is less effective for larger datasets because the average-case running time is still O(n2).

Alg2: Merge Sort Pseudocode

Pseudocode: merge(A, p, q, r){

n1 = q-p + 1;

n2 = r-q;

// create arrays L[1..n1+1] and R[1..n2+1]

for i = 1 to n1

L[i] = A[p+i-1]

for j = 1 to n2

R[j] = A[q + j]

L[n1 + 1] = infinity

R[n2 + 1] = infinity

i = 1

j = 1

for k = p to r

if L[i] <= R[j]

A[k] = L[i]

i = i+1

else

A[k] = R[j]

j=j+1

}

RT Analysis: O(n Log2 n) - To sort the array and determine the ith order statistic, ALG2 uses the MergeSort algorithm. It is efficient in a range of situations because it consistently maintains an O(n log2 n) running time complexity. MergeSort exhibits stability and reliability for larger datasets, as evidenced by its worst-case, best-case, and average-case complexities remaining O(n log n) even in partially sorted arrays.

Alg3: Randomized Select Pseudocode

randomized-select(A,p,r,i)

if p==r:

return A[p]

q=randomized-partition(A,p,r)

k=q-p+1

if i==k:

return A[q]

elif i<k:

return randomized-select(A,p,q-1,i)

else:

return randomized-select(A,q+1,r,i-k)

RT analysis: O(n) - The Randomized-Select algorithm is used by ALG3 to determine the ith order statistic. The worst-case situation has an O(n2) running time if the pivot consistently partitions the array unevenly. However, ALG3 generally performs well, achieving O(n) time complexity because of the random pivot selection. This makes it a sensible option when attempting to strike a balance for the Selection Problem between worst-case and average-case efficiency.

Experiments Results

Alg1 Table

n TheoreticalRT EmpiricalRT (ms) Ratio PredictedRT

10000 100000000.00 1087.16 0.00001158 1158.09

20000 400000000.00 4342.79 0.00001109 4436.27

30000 900000000.00 9718.87 0.00001101 9904.61

40000 1600000000.00 17455.38 0.00001120 17915.17

50000 2500000000.00 27138.01 0.00001094 27350.21

60000 3600000000.00 38996.45 0.00001097 39494.05

70000 4900000000.00 52767.96 0.00001083 53059.20

80000 6400000000.00 69090.38 0.00001086 69533.18

90000 8100000000.00 87302.45 0.00001082 87629.50

Alg2 Table

n TheoreticalRT EmpiricalRT (ms) Ratio PredictedRT

10000 132877.12 15.25 0.00011425 15.56

20000 285754.25 31.89 0.00010825 34.13

30000 446180.25 47.83 0.00010600 48.92

40000 611508.50 64.83 0.00010513 66.15

50000 780482.02 82.81 0.00010512 84.98

60000 952360.49 99.71 0.00010424 100.41

70000 1126654.71 118.14 0.00010538 118.76

80000 1303016.99 136.33 0.00010399 137.47

90000 1481187.36 156.75 0.00010611 158.71 100000 1660964.05 174.48 0.00010354 177.76

Alg3 Table

n TheoreticalRT EmpiricalRT (ms) Ratio PredictedRT

10000 10000.00 0.69 0.00009739 0.97

20000 20000.00 1.88 0.00009835 2.90

30000 30000.00 1.82 0.00007244 2.66

40000 40000.00 1.78 0.00003629 2.83

50000 50000.00 3.36 0.00007954 6.15

60000 60000.00 4.38 0.00009081 6.28

70000 70000.00 6.26 0.00009503 7.87

80000 80000.00 5.51 0.00008711 7.95

90000 90000.00 8.47 0.00007248 16.36

100000 100000.00 10.41 0.00005429 15.55

A graph with orange dots

Description automatically generated

A graph with a line and dots

Description automatically generated

A graph with a line and a line

Description automatically generated

A graph with a line and dots

Description automatically generated with medium confidence

Conclusions

To compare the theoretical estimates with the real execution times, the ratios of empirical to theoretical running times for each algorithm was computed. These ratios, on average, were relatively consistent across the different input sizes. A predicted running time based on the maximum ratio observed in each algorithm's experiments computed as well. Overall, the theoretical and empirical results align reasonably well, with the theoretical estimates providing a good indication of the algorithms' performance. However, it's crucial to note that while theoretical analysis lays a foundation, real-world execution can introduce variations influenced by factors such as hardware and language-specific optimizations.