Math Review: Exponents and Logarithms

How do I estimate Log2N ?

Table of exponents and logarithms

21= 2 Log22 = 1 26= 64 Log264 = 6

22= 4 Log24 = 2 27= 128 Log2128 = 7

23= 8 Log28 = 3 28= 256 Log2256 = 8

24= 16 Log216 = 4 29= 512 Log2512 = 9

25= 32 Log232 = 5 210= 1,024 Log21,024 = 10

Note that Log2N is the inverse of 2N.

Also, since 210 = 1,024 we often just take 210 as “about” 1,000.

Estimate log2 10,000? (10 thousand)

8,000 = 1,000 \* 8 = 210 \* 23 = 213 which means that log2 8,000 = 13

16,000 = 1,000 \* 16 = 210 \* 24 = 214 which means that log2 16,000 = 14

So, log2 10,000 is between 13 and 14. Since 10 is closer to 8 than 16 I’d answer “about 13”.

Estimate log2 50,000? (50 thousand)

32,000 = 1,000 \* 32 = 210 \* 25 = 215 which means that log2 32,000 = 15

64,000 = 1,000 \* 64 = 210 \* 26 = 216 which means that log2 64,000 = 16

So, log2 50,000 is between 15 and 16. Since 50 is closer to 64 than 32 I’d answer “about 16”.

Estimate log2 300,000,000? (300 million)

256,000,000 = 1,000 \* 1,000 \* 256 = 210 \* 210 \* 28 = 228 which means that log2 256,000,000 = 28

512,000,000 = 1,000 \* 1,000 \* 512 = 210 \* 210 \* 29 = 229 which means that log2 512,000,000 = 29

So, log2 300,000,000 is between 28 and 29. Since 256 is closer to 300 than 523 I’d answer “about 28”.

Estimate log2 500,000,000? (500 million)

512,000,000 = 1,000 \* 1,000 \* 512 = 210 \* 210 \* 29 = 229 log2 512,000,000 = 29

So, log2 500,000,000 is about 29.

It is often helpful to remember that:

210 is about 1,000 (one thousand) so Log21,000 is about 10

220 is about 1,000,000 (one million) so log21,000,000 is about 20

230 is about 1,000,000,000 (one billion) so log21,000,000,000 is about 30

Sequential Search Verses Binary Search

TimeToSearch = NumberOfComparisons \* TimePerComparison

Sequential Search Comparisons:

Minimum Number of Comparisons = 1

Maximum Number of Comparisons = N

Average Number of Comparisons (excluding failures) = ½ N

Binary Search Comparisons:

Minimum Number of Comparisons = 1

Maximum Number of Comparisons is approximately = log2N

Average Number of Comparisons is about the same as Maximum Number of Comparisons

Example 1: Sequential search of 10,000 students (unsorted) at 1/1,000 second per comparison

Comparisons (where N = 10,000)

Min = 1 Max = 10,000 Avg = 5,000 (assuming we are ignoring failures)

Runtime (where TimePerComparison = 1/1,000 sec)

Min = 1 \* 1/1,000 sec = 1/1,000 second

Max = 10,000 \* 1/1,000 sec = 10 seconds

Avg = 5,000 \* 1/1,000 sec = 5 seconds (assuming we are ignoring failures)

Example 2: Binary search of 10,000 students (presorted) at 1/10 second per comparison

Comparisons (where N = 10,000)

Min = 1

Max is approximately log210,000 which itself is approximately 13 (or 14)

Avg is approximately the same as Max, so 13 (or 14)

Runtime (where TimePerComparison = 1/10 sec)

Min = 1/10 second

Max is approximately log210,000 \* 1/10 sec = 13 (or 14) \* 1/10 sec = 1.3 to 1.4 seconds

Avg is approximately the same as Max, so 1.3 to 1.4 seconds

Example 3: Sequential search of 300,000,000 Soc. Sec. records (unsorted) at 1/1,000 sec per comparison

Comparisons (where N = 300,000,000)

Min = 1 Max = 300,000,000 Avg = 150,000,000 (assuming we are ignoring failures)

Runtime (where TimePerComparison = 1/1,000 sec)

Min = 1 \* 1/1,000 sec = 1/1,000 second

Max = 300,000,000 \* 1/1,000 sec = 300,000sec /60 sec per min /60 min per hour = 83.3 hours

Avg = 150,000,000 \* 1/1,000 sec = 150,000sec / 60 sec per min / 60 min per hour = 41.7 hours

Example 4: Binary search of 300,000,000 Soc. Sec. records (presorted) at 1/10 sec pre comparison

Comparisons (where N = 300,000,000)

Min = 1

Max is approximately log2300,000,000 which itself is approximately 28

Avg is approximately the same as Max, so 28

Runtime (where TimePerComparison = 1/10 sec)

Min = 1/10 second

Max is approximately log2300,000,000 \* 1/10 sec = 28 \* 1/10 sec = 2.8 seconds

Avg is approximately the same as Max, so 2.8 seconds

Selection Sort verses Insertion Sort verses Merge Sort

NumberOfComparisons = NumberOfPasses \* AverageNumberOfComparisonsPerPass

TimeToSort = NumberOfComparisons \* TimePerComparison

Selection Sort:

NumberOfComparisons = ½(N2 + N)

Derivation:

NumberOfComparisons = NumberOfPasses \* AvgerageNumberOfComparisonsPerPass

= (N + 1) \* 1/2 N

= 1/2 N2 + 1/2 N

= 1/2 (N2 + N)

Example: (N = 10,000 at 1/1,000 Second per Comparison)

NumberOfComparisons = ½ (10,0002 + 10,000)

= ½ (100,000,000 + 10,000)

= 50,000,000 + 5,000

= 50,005,000

TimeToSort = 50,005,000 \* 1/1,000 second

= 50,005 sec / 60 sec per min = 833 min / 60 min per hour = 13.9 hours

Insertion Sort:

MinNumberOfComparisons = (N - 1)

MaxNumberOfComparisons = ½ (N2 - N)

AvgNumberOfComparisons = ¼ (N2 - N)

Derivation:

NumberOfComparisons = NumberOfPasses \* AvgerageNumberOfComparisonsPerPass

= (N - 1) \* 1 [best case]

1/2 N [worst case]

1/4 N [average case]

MinNumberOfComparisons = (N-1)\* 1 = (N-1) [Best case]

MaxNumberOfComparisons = (N – 1) \* 1/2 N [Worst case]

= 1/2 N2 - 1/2 N

= 1/2 (N2 - N)

AvgNumberOfComparisons = (N – 1) \* 1/4 N [Average case]

= 1/4 N2 - 1/4 N

= 1/4 (N2 - N)

Insertion Sort (Continued)

Example: (N = 10,000 at 1/1,000 Second per Comparison)

[Best case]

NumberOfComparisons = (10,000 - 1)

= 9,999

TimeToSort = 9,999 \* 1/1,000 second

= 9.999 sec = about 10 seconds

[Worst case]

NumberOfComparisons = ½ (10,0002 - 10,000)

= ½ (100,000,000 - 10,000)

= 50,000,000 - 5,000

= 49,995,000

TimeToSort = 49,995,000 \* 1/1,000 second

= 49,995 sec / 60 sec per min = 833 min / 60 min per hour = 13.9 hours

[Average case]

NumberOfComparisons = ¼ (10,0002 - 10,000)

= ¼ (100,000,000 - 10,000)

= 25,000,000 - 2,500

= 24,997,500

TimeToSort = 24,997,500 \* 1/1,000 second

= 24,997.5 sec / 60 sec per min = 417 min / 60 min per hour = 6.9 hours

Points to Remember:

1. In the best case Insertion Sort can sort a list in about the same time needed to perform a sequential search. The best case occurs when the list is already sorted and the algorithm simply has to recognize this fact.
2. In the worst case Insertion Sort runs about as fast as Selection Sort (actually just a tad bit faster). The worst case occurs when the list is maximally out of order, when it is sorted in the opposite way that we would like (e.g., largest to smallest verses smallest to largest).
3. In the average case Insertion Sort is about twice as fast as Selection Sort (actually just a bit faster than twice as fast).

Merge Sort:

NumberOfComparisons = N Log2 N (approximately)

Derivation:

NumberOfComparisons = NumberOfLevels \* NumberOfComparisonsPerLevel

= Log2 N \* N

= N Log2 N

Example: (N = 10,000 at 1/1,000 Second per Comparison)

NumberOfComparisons = Log2 10,000 \* 10,000

(since Log2 10,000 is between 13 and 14) = no more than 14 \* 10,000

= 140,000

TimeToSort = 140,000 \* 1/1,000 second

= 140 sec / 60 sec per min = 2.33 minutes (2 minutes, 20 seconds)

Point to Remember:

Merge Sort is far, far faster than either Selection Sort or Insertion Sort. In fact, for 10,000 items Merge Sort expends far less than 1% of the effort required by either Selection or Insertion Sort to accomplish the same task. The gulf between the effort required by Merge and the other two sorts continues to grow ever wider as the sizes of the lists to be sorted increase.