1.How big of an array can I sort using merge-sort within one minute?

Merge sort runs in O(n log n) time because at each “level” of recursion it does an O(n) merge, and there are log n levels from repeatedly splitting the array in half.
A rough estimate of speed is that a program written in C or C++ would perform 10^8 high level operations such as addition, multiplication, comparison, reading memory, etc. in a second. Python, being an interpreted language, would be about 10 times slower (10^7). Therefore, if I run this code on Python, in one minute I am able to do 10^7 * 60 high level operations, i.e 6 * 10^8 , while in C or C++ I can do 6* 10^9 high level operations.
So the equation to solve for Python is roughly n * log2(n) = 6 * 10^8. This equation cannot be solved algebraically in a simple closed form, but we can approximate the solution numerically to n = 2.44 * 10 ^ 7 and so for C or C++ it is n = 2.44 * 10^8.
These are very rough ballpark figures and actual performance can vary significantly depending on hardware, compiler optimizations, memory speeds, and other implementation details.
So I proceed with an empirical research.
Empirical results in Python:
Sorting 1,000,000 elements took ~5 seconds on a local machine, and ~4.4–5 seconds on Google Colab.
Sorting 10,000,000 elements took ~65 seconds on Google Colab (very close to 1 minute).
Sorting 15,000,000 elements took ~100 seconds on Google Colab.
Sorting 24,400,000 elements took ~173 seconds on Google Colab.
On a local machine, attempting 24,400,000 elements caused memory constraints (swapping). This slowed the process considerably because the system had limited free RAM
Empirical results in C++:
Sorting 1,000,000 elements: ~0.14 seconds on a local machine, ~0.23 seconds on Google Colab.
Sorting 24,400,000 elements: ~4.26 seconds locally, ~6.73 seconds on Colab.
Sorting 244,000,000 elements: ~49.26 seconds locally, ~70.63 seconds on Colab.
Sorting 300,000,000 elements locally: ~67–86 seconds.
But Sorting 200,000,000 elements on Colab took ~57 seconds which is very close.

2. Gaussian elimination on an n-by-n matrix runs in time O(n^3). How long would it take to perform it on a matrix of size 2000x2000?

Gaussian elimination takes about O(n^3) time, so for a 2000 by 2000 matrix, that’s roughly 2000^3 = 8 × 10^9 operations. If Python handles about 10^7 operations per second, it would take around 800 seconds, which is over 13 minutes. In C or C++, which can do about 10^8 operations per second, it would take around 80 seconds. These are just rough estimates since actual speed depends on hardware and optimizations, but it gives a good idea of the difference.
So I conduceted an empirical research.
Results in Python:
On a local machine, 2000×2000 elimination took ~363 seconds (~6 minutes).
On Google Colab, 2000×2000 elimination took ~268 seconds (~4.5 minutes).

This can be explained because not every “high-level” operation is equally expensive,
some vectorized operations or hardware optimizations may come into play,
real compilers/interpreters can do partial optimizations behind the scenes.
Results in C++:
A 2000×2000 matrix took ~0.80 seconds on a local machine—far faster than the naive 80-second estimate while on Cola bit took ~3.6 seconds.
Optimizations in the C++ compiler (like -O2) can greatly reduce overhead.CPU hardware instructions (e.g., vectorization, caching) make numeric loops more efficient than a simple “1 operation = 1 CPU cycle” approximation.