Problem 0.1. Solve the matrix problem
\[
\begin{bmatrix}
1 & 2 & -1 \\
0 & 1 & 3 \\
-1 & 2 & 1
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c
\end{bmatrix}
= \begin{bmatrix}
2 \\
11 \\
6
\end{bmatrix}.
\] (The answer vector should be integers.)

Solution. I will use an augmented matrix and solve by row reduction. (My augmented matrix will not have the normal vertical line.)

\[
\begin{bmatrix}
1 & 2 & -1 & 2 \\
0 & 1 & 3 & 11 \\
-1 & 2 & 1 & 6
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 2 & -1 & 2 \\
0 & 1 & 3 & 11 \\
0 & 4 & 0 & 8
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 2 & -1 & 2 \\
0 & 1 & 3 & 11 \\
0 & 0 & 1 & 3
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 2 & 0 & 5 \\
0 & 1 & 0 & 2 \\
0 & 0 & 1 & 3
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 2 \\
0 & 0 & 1 & 3
\end{bmatrix}
\]

and thus \((a, b, c) = (1, 2, 3)\). □

Problem 0.2. Though I don’t expect you to understand the specifics, briefly explain what “block matrix operations” are and why they can be useful computationally. (I’m only looking for a paragraph or two.)

Solution. In a block matrix algorithm, the matrices are split into pieces called blocks, and all operations are done on this blocks instead of individual entries of the matrix. (It turns out that most algorithms can be easily modified to accommodate this.) While the programming is often a bit more complicated, there are two main advantages.

First, because memory access is often slower than computation, block methods can be designed to minimize memory access delays, speeding up the computations. (More detail than necessary: it can change memory accesses to roughly \(O(n^2)\) for a computation of \(O(n^3)\), so the memory delays become unimportant for large \(n\).)

Second, it is inherently parallelizable. Each piece of the block calculation can be sent to a different processor, then combined together later. □

Problem 0.3. Explain what it means for an algorithm to be \(O(n^{1.5})\). As part of that, what happens if you double the size of the matrices involved?

Solution. If an algorithm is \(O(n^{1.5})\), that means that, in doing the algorithm, the computer will perform about \(cn^{1.5}\) flops, where \(c\) is some constant, and “about"
means that we are ignoring terms that have lower power than $n^{1.5}$. That means that if you double the size of the matrices involved (i.e., double $n$), the amount of flops will roughly scale by $2^{1.5} \approx 2.83$. □