Part IV:

Working out the computational complexity for Case A

As x and B are both integer-valued $N \times N$ matrices, the number of elementary operations (elops) required to multiply x and B together is $N^2(2N-1)$ where N corresponds to the size of the matrices. As there are k terms, $kN^2(2N-1)$ elops are required for multiplication.

The number of elops required to work out x^{i_j} is:

$\sum_{j=1}^{k}(j-1)N^2(2N-1)$, which is equivalent to $N^2(2N-1)\sum_{j=1}^{k}(j)$ - $N^2(2N-1)\sum_{j=1}^{k}(1)$.

Evaluating this gives $N^2(2N-1)\big(0.5k(k+1)\big) - kN^2(2N-1)$.

The number of elops required to sum all the terms together in the polynomial is $N^2(k-1)$.

Hence, the total number of elops for case A is $kN^2(2N-1) + N^2(2N-1)\big(0.5k(k+1)\big) - kN^2(2N-1) + N^2(k-1)$, which simplifies down to $N^2(k-1) + 0.5k^2N^2(2N-1) + 0.5kN^2(2N-1)$.

Working out the computational complexity for Case B

As HornerSparseUnsorted requires the index array to be sorted Merge Sort, there are $klog(k)$ comparisons.

There is also $k-1$ elops to determine the power of the matrix x as the power of x is defined as $i_{k-1} - i_k$ with the first x being simply to the power of the first element in the index (hence its $k-1$ elops rather than k).

The number of elops required to compute BruteForceSearch is $0.5k(k+1)$ as BruteForceSearch has to find every element in the array B at some point so has to compare $k + (k-1) + (k-2) + \cdots + 3 + 2 + 1$ elements.

As the index list i is a permutation of the numbers 1,2,3,..,k then after the Merge Sort the index list will always give x^1 as $i_{k-1} - i_k$ will always be 1. Hence the number of elops required for FastPower is always 0.

As x and B are both integer-valued $N \times N$ matrices, the number of elementary operations (elops) required to multiply x and B together is $N^2(2N-1)$ where N corresponds to the size of the matrices. As there are k terms, $kN^2(2N-1)$ elops are required for multiplication.

The number of elops required to sum all the terms together in the polynomial is $N^2(k-1)$.

Hence, the total number of elops for Case B: $k - 1 + kN^2(2N-1) + N^2(k-1) + klog(k) + 0.5k(k+1)$

Working out the computational complexity for Case C

As x and B are both integer-valued $N \times N$ matrices, the number of elementary operations (elops) required to multiply x and B together is $N^2(2N-1)$ where N corresponds to the size of the matrices. As there are k terms, $kN^2(2N-1)$ elops are required for multiplication.

The number of elops required to sum all the terms together in the polynomial is $N^2(k-1)$.

The number of elops required to compute the matrix powers of x via the FastPower algorithm can be evaluated as $C(k) \leq kN^2(2N-1)(2+2log_2(k))$ where $C(k)$ represents the number of multiplicative operations computed by FastPower.

Hence, the total number of elops for Case C $\leq kN^2(2N-1)(2+2log_2(k)) + N^2(k-1) + kN^2(2N-1)$.

Comparing Cases A,B and C:

For Case A it can be deduced that $O(k^2N^3)$ and for Case B, $O(kN^3)$ and for C it is found that $O(kN^3log_2(k))$. Therefore, for large values of k and N Case A will have more elops and thus will be less efficient than both Cases B and C. As the total number of elops in Case C $\leq kN^2(2N-1)(2+2log_2(k)) + N^2(k-1) + kN^2(2N-1)$ and the total number of elops for Case B $= k - 1 + kN^2(2N-1) + N^2(k-1) + klog(k) + 0.5k(k+1)$. When both k and N are large values it can be concluded that for large values of k and N Case B will be more efficient than Case C, as for large k and N, $kN^2(2N-1)\big(2+2log_2(k)\big)$ is greater than $k - 1 + klog(k) + 0.5k(k+1)$. Hence, Case B is the most efficient, then Case C then Case A.