1. (35 points) Given two sorted lists A and B, merge(A, B) will return a new list consisting of elements from A and B with cost \(O(|A| + |B|)\), where \(|X|\) is the length of X, i.e., the number of elements in list X. Please design an efficient algorithm (as fast as you can) which merge n sorted lists into a single list by calling merge(A, B), where the lists \(S = \{L_1, L_2, \ldots, L_n\}\) have various sizes: for \(1 \leq i < n\), \(|L_i| = 2^i\), and \(|L_n| = 2\). Thus, the total number of elements in \(S\) is \(2^n\). Please analyze the complexity of your algorithm in terms of \(n\).

Answer: The following algorithm will merge \(S\) into one sorted list:

\[
\text{Proc \ mergeAll (} \{L_1, L_2, \ldots, L_n\} \text{)}
\]

L1: \(R = \text{merge}(L_1, L_n)\)
L2: for \(i\) from 2 to \(n - 1\) do \(R = \text{merge}(R, L_i)\)
L3: \text{return } R

Complexity analysis:
Merge at L1 costs 4; for each \(i\) at L2, merge costs \(2^{i+1}\), so the total cost from merge is \(4 + 2^3 + 2^4 + \ldots + 2^n = 4(2^{n-1} - 1) = O(2^n)\).
If you use Huffman coding algorithm, the sequence of merges will be the same, but you will pay extra \(O(n \log n)\) for priority queue operations.

2. (35 points) Given an \(m \times m\) matrix \(M\) and a positive integer \(n\), please design an efficient algorithm (as fast as you can) which computes \(M^n\), where \(M^1 = M\) and \(M^{i+1} = M \times M^i\). Please analyze the complexity of your algorithm in terms of \(m\) and \(n\).

Answer: Using the idea of Divide-and-Conquer, we may save time on matrix multiplication and power computation.

The fast matrix multiplication algorithm is called Strassen’s Algorithm, which divides the multiplication of \(m \times m\) matrices into 7 multiplications of matrices \(m/2\) by \(m/2\) matrices and the complexity is reduced from \(O(m^3)\) to \(O(m^\log_27)\). The details are given on Lecture Notes

The fast power computation is given below:

\[
\text{Proc \ fastPower}(M, n) \\
L1: \text{if } n == 1 \ \text{return } M; \\
L2: B = \text{fastPower}(M, n/2); \\
L3: B = \text{StrassenMatrixMulti}(B, B); \\
L4: \text{if } (n \text{ is even}) \ \text{return } B; \\
L5: \text{else return } \text{StrassenMatrixMulti}(B, M);
\]

Complexity analysis:
For each recursive call of fastPower, n is reduced to half (n/2 returns the floor of n/2), so there are at most \( \log(n) \) calls. Each call of fastPower calls StrassenMatrixMulti at most twice. Since the complexity of StrassenMatrixMulti is \( O(m^{\log_27}) \), so the total cost is \( O(\log(n)m^{\log_27}) \).

3. (30 points) Given two strings \( X[1..n] \) and \( Y[1..m] \), the minimal edit distance \( D[i,j] \) of \( X[1..i] \) and \( Y[1..j] \), where \( 0 \leq i \leq n \) and \( 0 \leq j \leq m \), is the minimal cost of changing \( X[1..i] \) into \( Y[1..j] \) by either deleting a letter, adding a letter, or replacing a letter in \( X[1..i] \). For example, the cost of changing \( X[1..5] = “steal” \) into \( Y[1..6] = “staple” \) is 3, as we need a sequence of 3 operations on \( X \): “delete e at \( X[3] \)”, “insert p after \( X[4] \)”, and “insert e after \( X[5] \)”. The cost here is simply the total number of operations of deleting, adding or replacing a letter. Let \( D[0,0] = 0 \) (need 0 additions), \( D[0,1] = 0 \) (need 0 deletions), and in general,

\[
\begin{align*}
&\text{if } x_i = y_j, \text{ then } D[i,j] = D[i-1,j-1] \text{ (this is a match, no operation at } X[i]) \\
&\text{else } D[i,j] = \min \{ D[i-1,j], D[i,j-1], D[i-1,j-1] \} + 1 \text{ (need one operation)}
\end{align*}
\]

Suppose the matrix \( D[0..n, 0..m] \) is already computed for you. Please design an efficient algorithm for finding a sequence of the least number of operations (of deleting, adding, or replacing) for changing \( X[1..n] \) into \( Y[1..m] \). Please analyze the complexity of your algorithm in terms of \( m \) and \( n \).

**Answer:** Since the matrix \( D[0..n, 0..m] \) is already computed, we use it to print out a sequence of operations on \( X \) by calling printOps(n, m), where printOps is given as follows:

```java
Proc printOps(i, j)
L1: if (i == 0) { if (j > 0) print(“insert”, Y[1..j], “at the beginning of X”); }
L2: else if (j == 0) print(“delete”, X[1..i], “from the beginning of X”);
L3: else if (X[i] == Y[j]) printOps(i-1, j-1);
L4: else if (D[i-1, j-1]+1 == D[i, j]) {
L5: printOps(i-1, j-1);
L6: print(“replace”, X[i], “by”, Y[j], “at X[”, i, “]”);
L7: }
L8: else if (D[i, j-1]+1 == D[i, j]) {
L9: printOps(i, j-1);
L10: print(“insert”, Y[j], “after X[”, i, “]”);
L11: }
L12: else { /* (D[i-1, j]+1 == D[i, j]) */
L13: printOps(i-1, j);
L14: print(“delete”, X[i], “at X[”, i, “]”);
L15: }
```

**Complexity analysis:**
The complexity of printOps(n, m) is \( O(n+m) \) because each recursive call decreases \( n \) or \( m \) or both by 1 and there is at most one recursive call in the execution of printOps.