Algorithm analysis homework

Name: Nguyen Dang Loc   Student ID: SE160199   Class: SE1616

1. MATRIX MULTIPLICATIONS

*Matrix A size M * N, matrix B size N * P*

Input size: Matrix size / total number of elements (M * N + N * P)

Code in Java:

```java
int M = A.length, N = A[0].length, P = B[0].length;
int[][] C = new int[M][P];

for (int i = 0; i < M; i++)
    for (int j = 0; j < P; ++j) {
        C[i][j] = 0;
        for (int k = 0; k < N; ++k)
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

Basic operation: Arithmetic operations as

+ Multiplications: A[i][k] * B[k][j]

+ Additions: C[i][j] + ...

All of the 3 loops completely process the entire size (i range from 1 to n, the same to j, k). Multiplications are executed once in each iterate of the innermost loop.

The total number of multiplications is $M \cdot N \cdot P$. It depends on the size of the 2 matrices, not on the value of the elements, the best-case and worst-case complexity are *the same. ($C_{worst} = C_{best}$)*

Thus, we have:

$C(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} 1 = \sum_{i=1}^{n} \sum_{j=1}^{n} n = \sum_{i=1}^{n} n^2 = n^3.$

Hence, the overall time complexity is $\Theta(n^3)$.

→ Best case: $O(n^3)$; Worst case - summation for $C(n) = O(n^3)$.

2. GAUSSIAN ELIMINATION

Input size: Total number of elements in matrix A (N * (N+1))

Code in Java:

```
int N = A.length;
for (int i = 0; i < N - 1; i++)
  for (int j = i + 1; j < N; j++)
    for (int k = N; k >= i; --k)
      A[j][k] = A[j][k] - (A[i][k] * A[j][i] / A[i][i]);
```

Basic operation: Arithmetic operations are executed on each repetition of the innermost loop.

+ Multiplications: A[i][k] * A[j][i]
+ Divisions: A[j][i] / A[i][i]
+ Subtractions: A[j][k] - ...

Time complexity to process the expression: $A_{j,k} = A_{j,k} - (A_{i,k} \times A_{j,i} \div A_{i,i})$ is $O(1)$.

The number of operations depends on the size of 2 matrices (no condition to consider), the best-case and worst-case complexity are *the same. ($C_{worst} = C_{best}$)*

According to the algorithm, the calculation for the total number of times the basic operation gets executed is:

$$\begin{aligned}
C(n) &= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} \sum_{k=i}^{n} 1 \\
&= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} (n - i + 1) \\
&= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} (n + 1) - \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} (i) \\
&= (n + 1) \sum_{i=0}^{n-1} (n - i) - \sum_{i=0}^{n-1} i(n - i) \\
&= n(n + 1) \sum_{i=0}^{n-1} 1 - (n + 1) \sum_{i=0}^{n-1} i - (n \sum_{i=0}^{n-1} i - \sum_{i=0}^{n-1} i^2) \\
&= n^2(n + 1) - (n + 1)\frac{n(n-1)}{2} - (n\frac{n(n-1)}{2} - \frac{(n-1)n(2n-1)}{6}) \\
&= \frac{1}{2}n(n + 1)^2 - \frac{1}{6}n(n^2 - 1) \\
&= \frac{1}{3}(n^3 + 3n^2 + 2n) \\
&\approx \frac{n^3}{3}
\end{aligned}$$

Hence, the overall time complexity is $O(n^3)$.

→ Best case: $O(n^3)$; Worst case - summation for $C(n) = O(n^3)$

3. KNAPSACK PROBLEM

Input size: N - number of items, W - capacity of the knapsack.

Code in Java:

a) Pick an item at most 1 time

```
public static int knapsack01(int N, int W, int[] w, int[] v) {
  if (N <= 0 || W <= 0) return 0;
  int[][] f = new int[N + 1][W + 1];
  for (int i = 0; i <= W; i++) f[0][i] = 0;
  for (int i=0; i <= N; i++) f[i][0]=0;

  for (int i = 1; i <= N; i++)
    for (int j = 1; j <= W; j++) {
      if (j >= w[i])
        f[i][j] = Math.max(f[i-1][j], f[i-1][j - w[i]] + v[i]);
      else
        f[i][j] = f[i-1][j];
  }
  return f[N][W];
}
```

b) Pick an item many times

```
public static int knapsackN(int N, int W, int[] w, int[] v) {
  if (N <= 0 || W <= 0) return 0;
  int[][] f = new int[N + 1][W + 1];
  for (int i = 0; i <= W; i++) f[0][i] = 0;
  for (int i=0; i <= N; i++) f[i][0]=0;

  for (int i = 1; i <= N; i++)
    for (int j = 1; j <= W; j++) {
      if (j >= w[i])
        f[i][j] = Math.max(f[i-1][j], f[i][j - w[i]] + v[i]);
      else
        f[i][j] = f[i-1][j];
    }
  return f[N][W];
}
```

For both 2 cases:

Basic operation: Comparisons, Assignments, Arithmetic operations (Additions, Subtractions)

No matter what the condition in the comparisons is true or false, the assignment will always be processed. The assignment operation is the update of the current f[i][j], it is executed once in each iteration.

The number of basic operations depends only on the size of inputs (i.e. N - the number of items, W - capacity of knapsack), the best-case and worst-case complexity are *the same. ($C_{worst} = C_{best}$)*

Thus, we have:

$C(n) = \sum_{i=1}^{n} \sum_{j=1}^{w} 1 = n \times w$.

Hence, the overall time complexity is $O(nW)$.

→ Best case: $O(nW)$; Worst case - summation for $C(n) = O(nW)$.