Programming Problem: Matrix Chain Multiplication

Naive Strategy

```
import java.io.*;
import java.util.ArrayList;
import java.util.Scanner;

public class naive {

      public static Long MultiplyMatrices(ArrayList<Integer>
matrixDimData) {
            Long numOfMultiplications = (long)0;
            while (matrixDimData.size() != 2) {
                  Long val1 = (long)matrixDimData.get(0); Long val2 =
(long)matrixDimData.get(1); Long val3 = (long)matrixDimData.get(2);
                  Long loopVal1 = Math.multiplyExact(val1, val2); Long
loopVal2 = Math.multiplyExact(loopVal1, val3);
                  numOfMultiplications =
Math.addExact(numOfMultiplications, loopVal2);
                  matrixDimData.remove(1);
            }
            return numOfMultiplications;
      }

      public static void main(String[] args) throws IOException {
            // TODO Auto-generated method stub
            long startTime = System.currentTimeMillis();
            int numOfMatrices = 0;
            int fileLine = 1;
            ArrayList<Integer> matrixDimData = new
ArrayList<Integer>();
            String line = new String();
            Scanner sc = new Scanner(System.in);
            while(true) {
                  try {
                        line = sc.nextLine();
                        if(line.length() == 0)
                              break;
                        if (fileLine == 1) {
                        if (line.length() == 0) {
                              System.out.println("Invalid input
format.");
                              break;
                        }
                        else {
                              numOfMatrices = Integer.parseInt(line);
                        }
                        fileLine++;
                        }
                        else {
                              if (line.length() == 0) {
                                    break;
                              }
                              else {
                                    String[] strArr = line.split(" ");
                                    for (int i = 0; i < strArr.length;
i++) {

      matrixDimData.add(Integer.parseInt(strArr[i]));
                                    }
                              }
                        }
                  }
                  catch(Exception e) {
                        break;
                  }
            }
            sc.close();
            if (matrixDimData.size() != numOfMatrices+1) {
                  System.out.println("Input format inavlid");
            }
            else {
                  Long numOfMultiplications =
MultiplyMatrices(matrixDimData);
                  System.out.println(numOfMultiplications);
            }
            long endTime = System.currentTimeMillis();
            System.out.println(endTime - startTime+ " milliseconds");
      }

}
```

__**Greedy Strategy**__

```
import java.math.BigInteger;
import java.util.Scanner;

public class greedy {
	static boolean debug = false;
	public static BigInteger greedyCalcCost(int start, int end, int[]
array) {
		BigInteger cost = BigInteger.ZERO;
		//BigInteger min  = BigInteger.valueOf(100000000);
		if(end == start+1) {
			return cost;
		}
		BigInteger min =
BigInteger.valueOf(array[start]).multiply(BigInteger.valueOf(array[sta
rt+1])).multiply(BigInteger.valueOf(array[end]));
		int k = -1;
		for(int i = start+1; i< end; i++) {

	if(BigInteger.valueOf(array[start]).multiply(BigInteger.valueOf(a
rray[i])).multiply(BigInteger.valueOf(array[end])).max(min).equals(min
)) {
				k = i;
				min =
BigInteger.valueOf(array[start]).multiply(BigInteger.valueOf(array[i])
).multiply(BigInteger.valueOf(array[end]));
			}
		}

		if(debug)
			System.out.println(min);
		if(debug)
			System.out.println("DEBUG: start: "+start+"; end:
"+end+"; k: "+k);
		cost = min.add(greedyCalcCost(start, k,
array)).add(greedyCalcCost(k, end, array));
		return cost;
	}

	public static void main(String[] args) {
		Scanner sc =  new Scanner(System.in);
		int num = sc.nextInt();
		int[] arr = new int[num+1];
		for(int i=0; i<= num; i++) {
			arr[i] = sc.nextInt();
		}

		System.out.println(greedyCalcCost(0, num, arr).toString());
	}

}
```

DP Strategy

```
import java.io.IOException;
import java.util.Scanner;

public class dp {

      public static Long MultiplyMatrices(int[] matrixDimData) {
            Long dpTable[][] = new
Long[matrixDimData.length][matrixDimData.length];
            for (int i = 1; i < matrixDimData.length; i++) {
                  dpTable[i][i] = (long) 0;
            }
            for (int i = matrixDimData.length - 1; i >= 1; i--) {
                  for (int j = i + 1; j < matrixDimData.length; j++) {
                        Long minCost = Long.MAX_VALUE;
                        for (int k = i; k < j; k++) {
                              Long dimensionCost1 = (long)
Math.multiplyExact(matrixDimData[i-1], matrixDimData[k]); Long
dimensionCost2 = (long) Math.multiplyExact(dimensionCost1,
matrixDimData[j]);
                              Long currentCost1 =
Math.addExact(dpTable[i][k], dpTable[k+1][j]); Long currentCost2 =
Math.addExact(currentCost1, dimensionCost2);
                              minCost = Math.min(minCost, currentCost2);
                        }
                        dpTable[i][j] = minCost;
                  }
            }
            return dpTable[1][matrixDimData.length-1];
      }

      public static void main(String[] args) throws
NumberFormatException, IOException {
            // TODO Auto-generated method stub
            long startTime = System.currentTimeMillis();

            Scanner sc =  new Scanner(System.in);
            int numOfMatrices = sc.nextInt();
            int[] matrixDimData = new int[numOfMatrices+1];
            for(int i=0; i<= numOfMatrices; i++) {
                  matrixDimData[i] = sc.nextInt();
            }
            sc.close();
            System.out.println(MultiplyMatrices(matrixDimData));
            long endTime = System.currentTimeMillis();
            System.out.println(endTime - startTime+ " milliseconds");
      }

}
```