Reporte de laboratorio 4

Laura Rincón Riveros - B55863
Esteban Vargas Vargas - B16998
Grupo 3

5 de octubre de 2016

Índice

---

1. Introducción

En el presente laboratorio se realizó una revisión bibliográfica para poder sintetizar los conceptos de problemas con complejidad NP, NP-duros y NP-completos. Asimimso se realizó una búsqueda sobre problemas clásicos de las complejidades mencionadas.

En la segunda parte del laboratorio se analizó un código fuente proporcionado por el profesor y se obtuvo su función de tiempo de ejecución y su complejidad O; adicionalmente se elaboraron gráficas para representar con facilidad los datos.

2. Desarrollo

2.1. Conceptos de complejidad de problemas

2.1.1. Problemas NP

Un problema *NP* se define como un problema que se puede resolver mediante un algoritmo con una función de tiempo de ejecución no polinomial en una máquina determinista.

2.1.2. Problemas NP-duros

Si se tiene un conjunto de problemas *NP*, un problema es *NP-duro* si la duración de todos esos problemas *NP* se puede transformar mediante un factor polinomial en la duración del problema *NP-duro*.

2.1.3. Problemas NP-completos

Un problema es NP-completo si se puede reducir con una función polinomial en una función polinomial también; y además la verificación de su instancia también es polinomial.

2.2. Problemas clásicos

2.2.1. Problemas NP

- El problema de factorizar números en una multiplicación de números primos: cuando la cifra es grande el problema es intratable.
- El problema de isomorfismo de grafos: consiste en la determinación de si dos grafos con el mismo número de vértices y aristas son isomorfos o no. No se conoce si es resoluble en tiempo polinómico o si es *NP-completo*.

2.2.2. Problemas NP-duros

- Problema de la suma de subconjuntos: dado un conjunto de enteros, la pregunta es si existe algún subconjunto en él cuya suma sea cero. (Problema también es *NP-completo*).
- El problema del agente viajero: dado un viajero y un conjunto de ciudades, el problema es encontrar cuál es la distancia más corta posible en la que el viajero puede visitar todas y volver a su punto de origen.

2.2.3. Problemas NP-completos

- El problema SAT de la satisfactibilidad de la lógica proposicional. (Fue el primer problema demostrado ser NP-completo, por S.A. Cook en 1971).
- El problema de *Clique*: es un problema de decisión de si un grafo contiene un *clique* de al menos un tamaño k. (Un clique se le dice a un subgrafo con todos los vértices conectados entre ellos).

2.3. Programa TicTacToe

2.3.1. Explicación

El programa en **ttt.src** contiene el algoritmo de un juego de TicTacToe para dos jugadores. El cuál se basa en una matriz cuadrada, cada jugador ingresa una posición y el programa verifica si el valor asignado para ese jugador está en una columna, fila o digonal completa. Si se cumple lo anterior se determina el ganador o se continúa hasta que alguno gane.

2.3.2. Función de tiempo de ejecución

```
public class TicTacToe {

    int[][] matrix;

    /** Initialize your data structure here. */
    public TicTacToe(int n) {
        matrix = new int[n][n]; //(1) asignación
    }

    /** Player {player} makes a move at ({row}, {col}).
        @param row The row of the board.
        @param col The column of the board.
        @param player The player, can be either 1 or -1.
        @return The current winning condition, can be either:
                0: No one wins.
                1: Player 1 wins.
                -1: Player 2 wins. */
    int move(int row, int col, int player) {
        matrix[row][col]=player; //(1) asignacion, (1) acceso a memoria

        //check row
        boolean win=true; //(1) asignacion
        for(int i=0; i<matrix.length; i++){ // [(1) asignación, (1) comparación, (1) incremento] se ejecuta n veces
            if(matrix[row][i]!=player){  // (1) acceso a memoria, (1) comparación
                win=false; //(1) asignación
                break; // (1) intrucción
            }
        }

        if(win) return player; // (1) comparación, (1) instrucción
```

(a) Función move

```
int main()
{
    int player = -1; //(1) asignación

    boolean end = false; //(1) asignación

    int n = 3; //(1) asignación
    int r, c, result; //(1) asignación

    TicTacToe ttt = new TicTacToe(n); //(1) declaración
    while(!end) //se ejecuta n^2
    {
        player *= -1 //(1) asignación, (1) multiplicación
        r =  randomInt(0, 3);//(1) función randomInt, (1) asignación
        c =  randomInt(0, 3);//(1) función randomInt, (1) asignación

        result = move(player, r, c); // función move, (1) asignación
        if(result != 0) //(1) comparación
        {
            print("player " + result + " won."); //(1) función print
            end = true; //(1) asignación
        }
    }
    return 0; //(1) instrucción
}
```

(b) Main

Figura 1: Conteo para obtener función del tiempo

[1] Se procedió a contar cada intrucctivo para cuantificar el tiempo de ejecución de todo el programa.

Por lo que, se puede observar en la Figura 1.a se encuentra la declaración de las variables y la función de move. Para el caso de la función del progama, las verificaciones como se repiten se obtiene lo siguiente:

$$T_1(n) = 1 + 1 + 4 * (1 + [(1 + 1 + 1)n * (1 + 1 + 1 + 1)]) + 1 + 1 + 1 + 1 = 48n + 10 \qquad (1)$$

Si agregamos los intrucctivos del main y la inicialización de las variables a (1), de la Figura 1.b:

$$T(n) = 1+1+1+1+1+1+[n^2*(1+1+1+1+1+1+48n+10+1+1+1+1)]+1 = 48n^3+20n^2+7 \quad (2)$$

La función de tiempo dada por (2) pertenece a todo el programa. Y se puede observar en la Figura 2, la segunda gráfica.

[1]Revisar el archivo **ttt.src** comentado y contado

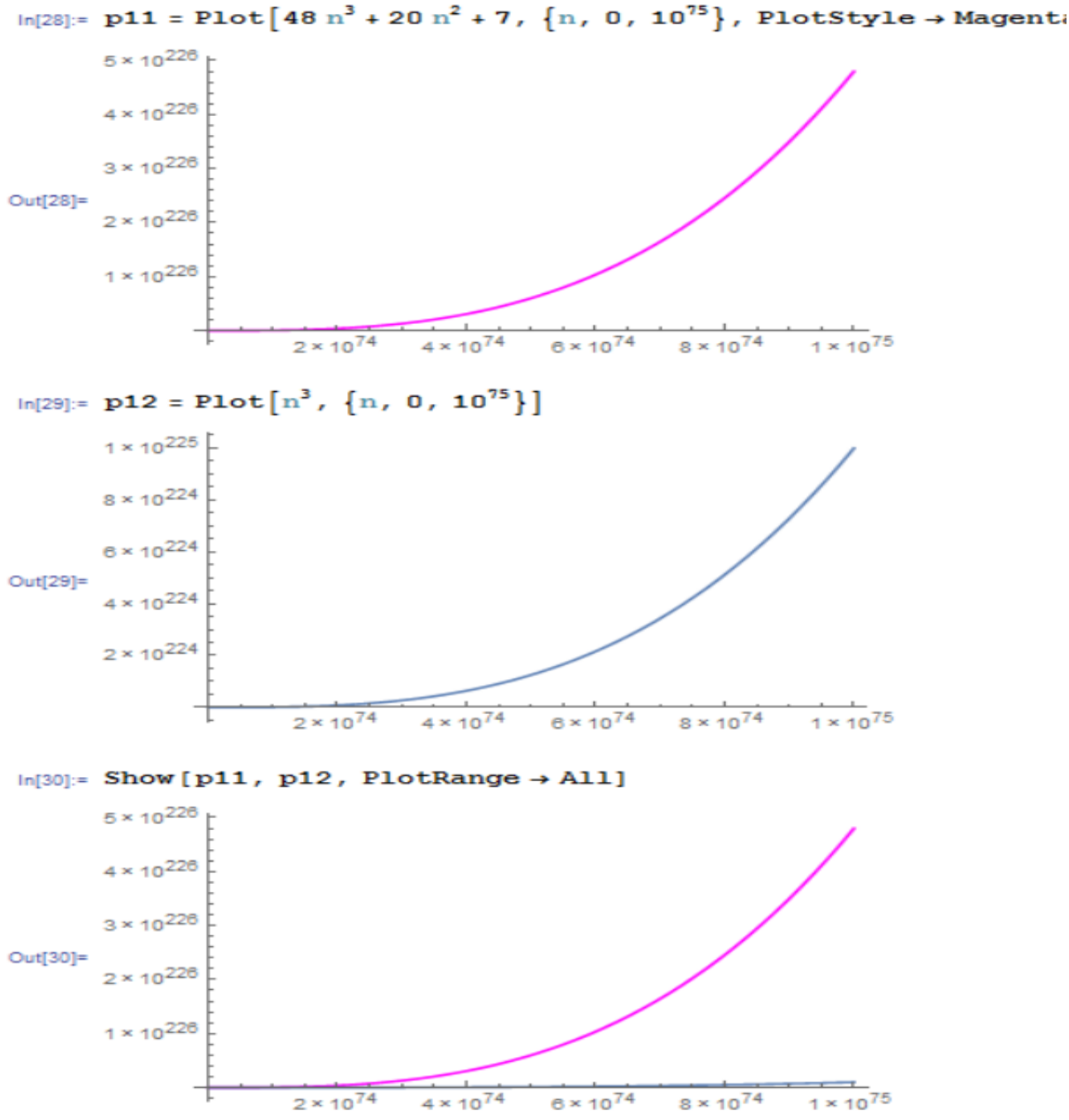

Figura 2: Conteo para obtener función del tiempo

2.3.3. Función de complejidad O

Si se simplifica la función de tiempo, se puede obtiene la función de complejidad $O(n^3)$, que se puede representar en la Figura 2, la primera gráfica. También se puede ver que O(n) acota inferiormente a T(n), en la tercera gráfica de la Figura 2

3. Conclusiones

- Se describieron y explicaron los conceptos de complejidades NP, NP-dura y NP-completos.
- Se estudiaron los problemas clásicos de las distintas complejidades NP.
- Se obtuvo la función de tiempo y comlejidad del programa **ttt.src**
- Se graficaron las funciones de tiempo y complejidad del programa proporcionado.

Referencias

[1] Pérez, M.Sancho,F. (2003). *Máquinas moleculares basadas en ADN*. Sevilla, España: Secretariado de Publicaciones de la Universidad de Sevilla.

[2] *Complejidad-ProblemasNP-Completos*. Algoritmos y estructuras de datos III. Recuperado de $https://www.dc.uba.ar/materias/aed3/2014/2c/teorica/handout_compl.pdf$.