TD3: Algoritmos y Estructuras de Datos

Segundo Semestre de 2024

Clase práctica: Complejidad algorítmica

Ejercicio 1

Para el siguiente programa

1. definir el tamaño de entrada n,
2. identificar el peor caso de ejecución del programa,
3. escribir la función de costo temporal $T(n)$ asociada al peor caso,
4. proponer un órden de complejdiad $\mathcal{O}$ para $T(n)$,
5. mostrar que $T(n)$ pertenece al órden de complejidad propuesto.

```cpp
int suma(vector<int> &v) {
    int suma = 0;
    for (int i = 0: i < v.size(); i++) {
        suma = suma + v[i];
    }

    return suma;
}
```

Ejercicio 2

Para el siguiente programa

1. definir el tamaño de entrada n,
2. identificar el peor caso de ejecución del programa,
3. escribir la función de costo temporal $T(n)$ asociada al peor caso,
4. proponer un órden de complejdiad $\mathcal{O}$ para $T(n)$,
5. mostrar que $T(n)$ pertenece al órden de complejidad propuesto.

```
int sumaMientrasMenor(vector<int> &v, int max) {
    int suma = 0;
    int i = 0;
    while (i < v.size() && v[i] <= max) {
        suma = suma + v[i];
        i = i + 1;
    }

    return suma;
}
```

Ejercicio 3

Para el siguiente programa

1. definir el tamaño de entrada n,
2. identificar el peor caso de ejecución del programa,
3. escribir la función de costo temporal $T(n)$ asociada al peor caso,
4. proponer un órden de complejdiad $\mathcal{O}$ para $T(n)$,
5. mostrar que $T(n)$ pertenece al órden de complejidad propuesto.

```cpp
int sumaDoblePares(vector<int> &v) {
    int suma = 0;
    int i = 0;
    while (i < v.size()) {
        if(v[i] % 2 == 0) {
            int j = 0;
            while (j < v.size()) {
                suma = suma + v[j];
                j = j + 1;
            }
        }
        i = i + 1;
    }

    return suma;
```

Ejercicio 3 bis

Para el siguiente programa

1. definir el tamaño de entrada n,
2. identificar el peor caso de ejecución del programa,
3. escribir la función de costo temporal $T(n)$ asociada al peor caso,
4. proponer un órden de complejdiad $\mathcal{O}$ para $T(n)$,
5. mostrar que $T(n)$ pertenece al órden de complejidad propuesto.

```cpp
int sumaDoblePares(vector<int> &v) {
    int suma = 0;
    int i = 0;
    while (i < v.size()) {
        if(v[i] % 2 == 0) {
            int j = 0;
            while (j < i) { // antes: j < v.size()
                suma = suma + v[j];
                j = j + 1;
            }
        }
        i = i + 1;
    }

    return suma;
```

Análisis de funciones

Indique si la siguiente afirmación es verdadera o falsa. Justifique su respuesta haciendo uso de la definición de $\mathcal{O}$.

1. $n \in \mathcal{O}\left(n \log(n)\right)$
2. $n^2 - 4n + 2 \in \mathcal{O}(n^2)$
3. $2n^2 + n \in \mathcal{O}(n)$

Complejidad de algoritmos recursivos

Para el siguiente programa

1. definir el tamaño de entrada *n*,
2. identificar el peor caso de ejecución del programa,
3. escribir la ecuación de recurrencia T(n) asociada al peor caso,
4. usar el árbol de recursión para dar una fórmula cerrada para T(n),
5. usando la fórmula cerrada, proponer un orden de complejidad $\mathcal{O}$ para T(n) y demostrar que pertenece.

```
minimoAux(v, 0, |v|)

int minimoAux(vector<int> &v, int desde, int hasta) {
    if (desde - hasta <= 1) {
        return v[desde];
    }
    int mitad = (desde+hasta)/2;
    int sol1 = minimoAux(v, desde, mitad);
    int sol2 = minimoAux(v, mitad, hasta);
    return min(sol1, sol2);
}
```

Algunas preguntas

Dada la ecuación de recurrencia del ejercicio anterior, ¿qué ocurré con el árbol de recursión cuando cambiamos cada uno de los siguientes elementos?

1. La cantidad de subproblemas.
2. El tamaño de los subproblemas.
3. El costo de conquer (la porción no recursiva).