

# Análisis y diseño de algoritmos: Práctica 1

Jorge Luis Huanca Mamani: jorge.huanca@ucsp.edu.pe

Octubre 2020

## 1. Problema 1

Utilizando las reglas de notación asintóticas, demuestre las expresiones son verdaderas o falsas, justificando con la propia demostración matemática y también con un breve comentario en caso sea necesario. Asuma que toda función presentada es positiva.

*Descripción :*

$$\blacksquare \max(f(n), g(n)) = \Theta(f(n) + g(n))$$

para  $\Theta(n)$  se debe tener dos asintotas gobernadas por dos constantes  $C1, C2$  y estas definirán si es superior o inferior, por lo tanto tenemos la siguiente inecuación.

$C1(f(n) + g(n)) \leq \max(f(n), g(n)) \leq C2(f(n) + g(n))$  donde  $C1$  es la constante que definirá la asintota inferior y  $C2$  definirá la asintota superior.

Solo basta con mirar el max donde se puede deducir que solo una función entrará a la desigualdad por lo que tiene sentido multiplicar por una fracción menor a uno a la suma de funciones y que dicha fracción sea el valor concreto de  $C1$ .

Entonces tenemos que de la parte max ya sea  $f(n)$  o  $g(n)$  entrará a la desigualdad, entonces si tenemos una parte grande y una pequeña para lo cual ambas estan multiplicados por una fracción menor a uno, cobraría sentido que si dividimos la parte mas grande en dos y por definición de max tendríamos una parte grande dividida en dos más la suma de una parte pequeña tambien dividida por dos y esto a su vez nunca sera mayor o igual a una parte grande ya sea  $f(n)$  o  $g(n)$  por lo tanto la constante  $C1$  tomaria el valor de  $\frac{1}{2}$

Ahora para la asintota superior necesitamos determinar la constante  $C2$

por lo que si nos fijamos en la parte derecha de la ecuación que es  $\max(f(n), g(n)) \leq C2(f(n) + g(n))$ , entonces solo basta con asumir que  $C2$  toma el valor de 1 para que nos quede una expresión de  $1 \leq j(n)$  donde  $j$  es una función ya sea  $f$  o  $g$  y dicha desigualdad siempre se cumpliría.

■  $(n + a)^b = \Theta(n^b) \mid a, b \in \mathbb{R}^+, b > 0$

Entonces tenemos lo siguiente:

Como se ve en la gráfica, se tiene las clásicas dos constantes  $C1$  y  $C2$  que

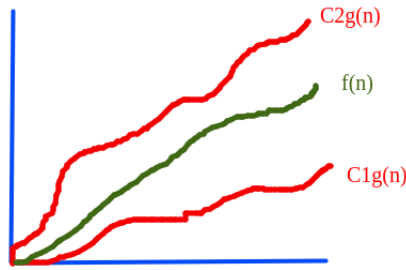


Figura 1: Representación Gráfica del planteamiento

determinan la función asintótica superior e inferior, para ello se expresa la siguiente desigualdad:

$$C1g(n) \leq f(n) \leq C2g(n)$$

donde  $f(n) = (n + a)^b$  y  $g(n) = n^b$  reemplazando

$$C1.n^b \leq (n + a)^b \leq C2.n^b$$

$$\frac{C1.n^b}{n^b} \leq \left(\frac{n+a}{n}\right)^b \leq \frac{C2.n^b}{n^b}$$

$$C1 \leq \left(\frac{n+a}{n}\right)^b \leq C2$$

de la primera parte  $C1 \leq \left(\frac{n+a}{n}\right)^b$

de la parte mas a la izquierda tenemos lo siguiente:

$$C1 \leq \left(\frac{n+a}{n}\right)^b$$

supongamos lo siguiente que  $n > 0$  y  $n = a$  tenemos

$$C1 \leq \left(\frac{a+a}{a}\right)^b$$

$$C1 \leq 2^b$$

ahora por la parte mas a la derecha tenemos lo siguiente

$$\left(\frac{n+a}{n}\right)^b \leq C2$$

por lo que con la misma premisa de  $n = a$  se tiene

$$2^b \leq C2$$

Por lo que concluimos que  $(n + a)^b$  está dominada firmemente por  $n^b$

- $2^{(n+1)} = O(2^n)$

Entonces tenemos lo siguiente:

Como se ve en la gráfica anterior, donde  $f(n) = 2^{n+1}$  y  $g(n) = 2^n$ , en

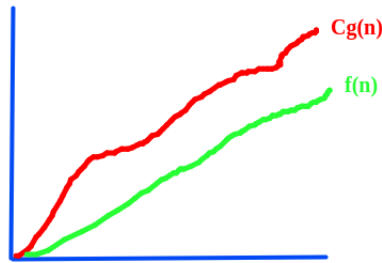


Figura 2: Representación Gráfica del planteamiento

esta ocasión solo nos pide hallar la constante  $C$  que determina la función asintótica superior para ello :

$$\begin{aligned} 2^{n+1} &\leq C2^n \\ 2^{n+1} &\leq C2^n \frac{2}{2} \\ 2^{n+1} &\leq \frac{C2}{2} 2^{n+1} \\ 1 &\leq \frac{C}{2} \\ 2 &\leq C \end{aligned}$$

Entonces para  $C = 2$  y  $n_0 \geq 0$  se tiene la función asintótica por lo tanto  $O(2^n)$  domina asintóticamente a  $2^{(n+1)}$ .

- $2^{2n} = O(2^n)$

Para este problema nos piden algo similar al ejercicio anterior que es probar que  $g(n) = 2^n$  domina asintóticamente a  $f(n) = 2^{2n}$  para ello se propone la siguiente inecuación.

$$\begin{aligned} 2^{2n} &\leq C2^n \text{ resolviendo} \\ \ln 2(2n) &\leq \ln C + \ln 2n \end{aligned}$$

$$2n \leq \ln C + n$$

como  $2n \leq \ln C$  nunca sucede como se muestra en la siguiente grafica.

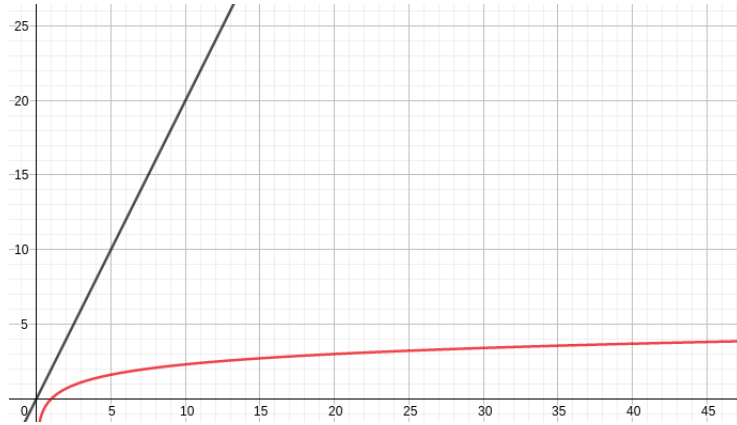


Figura 3: No existe un numero  $C$  que haga valer la desigualdad

por lo tanto esta afirmacion es falsa.

## 2. Problema 2

La recurrencia  $T(n) = 7T(\frac{n}{2}) + n^2$  representa la función de complejidad del algoritmo A. Un algoritmo alternativo A' para el mismo problema tiene la siguiente ecuación  $T'(n) = aT'(\frac{n}{4}) + n^2$ . Cuál es el mayor número entero  $a$ , que hace que el algoritmo A' sea asintóticamente más rapido que A.

*Descripción* : Describa detalladamente el proceso para encontrar el valor de  $a$  y de como respuesta  $a^2$ .

Entonces se tiene lo siguiente:

$$T(n) = 7T(\frac{n}{2}) + n^2 \text{ que representa al algoritmo A.}$$

$$T'(n) = aT'(\frac{n}{4}) + n^2 \text{ que representa al algoritmo A'.$$

Para hacer que  $A'$  sea asintóticamente más rapido que A, esto se interpreta que A es explicitamente mayor a  $A'$  lo que implica  $O(f(A')) < O(f(A))$ .

Entonces resolveremos con el teorema maestro las recurrencias de A y A'.

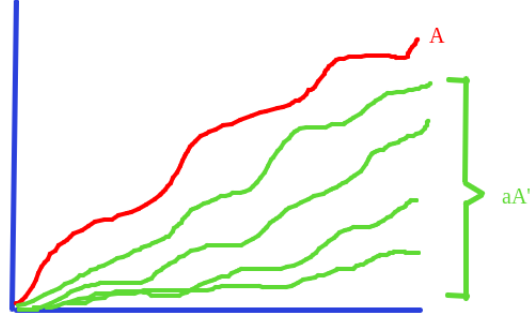


Figura 4: Representación Gráfica del planteamiento

1.  $T(n) = 7T(\frac{n}{2}) + n^2$   
donde  $a = 7, b = 2, f(n) = n^2$   
entonces :  $n^{\log_b a} \implies n^{\log_2 7} (*)$ .  
resolviendo el logaritmo tenemos  $n^{2,80735..}$ .  
Por el Teorema Maestro en el caso I tenemos lo siguiente:  
 $f(n) = O(n^{\log_b a - \epsilon})$  para  $\epsilon > 0$   
 $f(n) = O(n^{2,80735 - 0,80735})$  para  $\epsilon \approx 0,80735$   
entonces concluimos que  $T(n) = \Theta(n^{2,80735})$
2.  $T'(n) = aT'(\frac{n}{4}) + n^2$   
donde  $a = a, b = 4, f(n) = n^2$   
entonces :  $n^{\log_b a} \implies n^{\log_4 a} (*)$ .
3. Por lo tanto como definimos al inicio  $A$  tiene que ser explícitamente mayor a  $A'$  para esto usamos los  $(*)$  y tenemos lo siguiente.

$$\begin{aligned} \log_4 a &< \log_2 7 \\ \frac{\log_2 a}{\log_2 4} &< \log_2 7 \\ \log_2 a &< 2\log_2 7 \\ \log_2 a &< \log_2 49 \end{aligned}$$

Entonces para cumplir dicha desigualdad el valor máximo de  $a$  es 48 por lo que  $a = 48$

Para  $a = 48$  tenemos lo siguiente:

$$T'(n) = 48T'(\frac{n}{4}) + n^2, \text{ esto lo resolvemos con el teorema maestro.}$$

$$\text{donde } a = 48, b = 4, f(n) = n^2$$

$$\text{entonces: } n^{\log_b a} \implies n^{\log_4 48} \implies n^{2,79248}$$

Por el Teorema Maestro en el caso I tenemos lo siguiente:

$$f(n) = (n^{2,74248}) \implies T(n) = (n^{2,74248})$$

### 3. Problema 3

Dada las siguientes ecuaciones de recurrencia, resolver utilizando expansión de recurrencia y teorema maestro. En ambos casos colocar todo el desarrollo no obviar partes. Si en caso no se pueda aplicar el teorema maestro, explique el motivo. Asuma que  $T(n)$  es constante para  $n < 2$ .

*Descripción :*

- $T(n) = 4T(\frac{n}{2}) + n$   
Usando expansión de recurrencia:

$$T(n) = \frac{n}{2} = 4(4T(\frac{n}{2^2}) + \frac{n}{2}) + n$$

$$T(n) = \frac{n}{2} = 4^2T(\frac{n}{2^2}) + 3n$$

$$T(n) = \frac{n}{2^2} = 4^2(4T(\frac{n}{2^3}) + \frac{n}{2^2}) + 3n$$

$$T(n) = \frac{n}{2^2} = 4^3T(\frac{n}{2^3}) + 7n$$

$$T(n) \frac{n}{2^{i-1}} = 4^iT(\frac{n}{2^i}) + (2^i - 1)n$$

$$\log_2 n = i$$

$$4^{\log_2 n} + (2^{\log_2 n} - 1)n$$

$$n^2 + n^2 - n$$

$$2n^2 - n$$

Asumiendo:

$$C1 = 1, C2 = 5, n_0 = 2$$

entonces

$$2n^2 - n = \Theta(n^2)$$

por lo tanto concluimos:

$$T(n) = 4T(\frac{n}{2}) + n \text{ esta firmemente dominada por } \Theta(n^2)$$

Usando el Teorema Maestro:

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

donde:

$$a = 4, b = 2, f(n) = n \text{ entonces}$$

$$n^{\log_b a} = n^{\log_2 4} = \Theta(n^2)$$

Recae en el Caso 1 del Teorema Maestro:

$$f(n) = \Theta(n^{\log_b a - \epsilon}) \text{ para cualquier } \epsilon > 0$$

Entonces concluimos que:

$$T(n) = \Theta(n^2)$$

Que reafirma la conclusion expuesta con la expansión de recurrencia.

$$\blacksquare \quad T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

Usando expansión de recurrencia:

$$T(n) = \frac{n}{2} = 4\left(4T\left(\frac{n}{2^2}\right) + \left(\frac{n}{2}\right)^2\right) + n^2$$

$$T(n) = \frac{n}{2} = 4^2 T\left(\frac{n}{2^2}\right) + 2n^2$$

$$T(n) = \frac{n}{2^2} = 4^2 \left(4T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^2\right) + 2n^2$$

$$T(n) = \frac{n}{2^2} = 4^3 T\left(\frac{n}{2^3}\right) + 3n^2$$

$$T(n) = \frac{n}{2^{i-1}} = 4^i T\left(\frac{n}{2^i}\right) + in^2$$

$$\log_2 n = i$$

$$4^{\log_2 n} + (\log_2 n)n^2$$

$$n^2 + \log_2 n(n^2)$$

$$2n^2 - n$$

Asumiendo:

$$C1 = 1, C2 = 5, n_0 = 10$$

entonces

$$2n^2 + (\log_2 n)n^2 = \Theta(n^2 \log n)$$

por lo tanto concluimos:

$$T(n) = 4T(\frac{n}{2}) + n^2 \text{ esta firmemente dominada por } \Theta(n^2 \log n)$$

Usando el Teorema Maestro:

$$T(n) = 4T(\frac{n}{2}) + n^2$$

donde:

$$a = 4, b = 2, f(n) = n^2 \text{ entonces}$$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

Recae en el Caso 2 del Teorema Maestro:

Entonces concluimos que:

$$T(n) = \Theta(n^2 \log n)$$

Que reafirma la conclusion expuesta con la expansión de recurrencia.

- $T(n) = 4T(\frac{n}{2}) + n^3$  Usando expansión de recurrencia:

$$T(n) = \frac{n}{2} = 4(4T(\frac{n}{2^2}) + (\frac{n}{2})^3) + n^3$$

$$T(n) = \frac{n}{2} = 4^2 T(\frac{n}{2^2}) + \frac{3n^3}{2}$$

$$T(n) = \frac{n}{2^2} = 4^2(4T(\frac{n}{2^3}) + (\frac{n}{2^2})^3) + \frac{3n^3}{2}$$

$$T(n) = \frac{n}{2^2} = 4^3 T(\frac{n}{2^3}) + \frac{n^3}{4} + \frac{3n^3}{2}$$

$$T(n) = \frac{n}{2^2} = 4^3 T(\frac{n}{2^3}) + \frac{7n^3}{2^2}$$

$$T(n) \frac{n}{2^{i-1}} = 4^i T(\frac{n}{2^i}) + \frac{(2^i - 1)n^3}{2^{i-1}}$$

$$\log_2 n = i$$



$$4^{\log_2 n} + \frac{(2^{\log_2 n} - 1)n^3}{2^{\log_2 n - 1}}$$

$$n^2 + \frac{(n-1)n^3}{2^{\log_2 n - 1}}$$

$$2n^3 + n^2 - 2$$

Asumiendo:

$$C1 = 5, C2 = 1, n_0 = 10$$

entonces

$$2n^3 + n^2 - 2 = \Theta(n^3)$$

por lo tanto concluimos:

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3 \text{ esta firmemente dominada por } \Theta(n^3)$$

Usando el Teorema Maestro:

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

donde:

$$a = 4, b = 2, f(n) = n^3 \text{ entonces}$$

$$n^{\log_b a} = n^{\log_2 4} \text{ para un } \epsilon = \frac{1}{2}$$

Recae en el Caso 3 del Teorema Maestro:

$$n^3 = \Omega(n^{\log_2 4 + \epsilon})$$

$$n^3 = \Omega(n^{2 + \frac{1}{2}})$$

$$\text{si } C < 1$$

$$4\left(\frac{n}{2}\right)^2 \leq Cn^3$$

$$\frac{n^3}{2} \leq Cn^3$$

sea  $C = \frac{1}{2}$  cumple con la propiedad

Entonces concluimos que:

$$T(n) = \Theta(n^3)$$

Que reafirma la conclusion expuesta con la expansión de recurrencia.

- $T(n) = T(\frac{9n}{10}) + n$
- $T(n) = 16T(\frac{n}{4}) + n^2$

Usando expansión de recurrencia:

$$T(n) = \frac{n}{4} = 16(16T\frac{n}{4^2} + (\frac{n}{4})^2) + n^2$$

$$T(n) = \frac{n}{4} = 16^2T(\frac{n}{4^2}) + 2n^2$$

$$T(n) = \frac{n}{4^2} = 16^2(16T\frac{n}{4^3} + (\frac{n}{4^2})^2) + 2n^2$$

$$T(n) = \frac{n}{4^2} = 16^3T(\frac{n}{4^3}) + 3n^2$$

$$T(n) \frac{n}{2^{i-1}} = 16^iT(\frac{n}{4^i}) + in^2$$

$$\log_4 n = i$$

$$16^{\log_4 n} + \log_4 nn^2$$

$$n^2 + \log_4 nn^2$$

$$2n^3 + n^2 - 2$$

Asumiendo:

$$C1 = \frac{1}{2}, C2 = 3, n_0 = 16$$

entonces

$$2n^3 + n^2 - 2 = \Theta(n^3)$$

por lo tanto concluimos:

$$T(n) = 16T(\frac{n}{4}) + n^2 \text{ esta firmemente dominada por } \Theta(n^2 \log_4 n)$$

Usando el Teorema Maestro:

$$T(n) = 16T(\frac{n}{4}) + n^2$$

donde:

$$a = 16, b = 4, f(n) = n^2 \text{ entonces}$$

$$\log_b a = \log_2 16 = 2$$

Recae en el Caso 2 del Teorema Maestro:

Entonces concluimos que:

$$T(n) = \Theta(n^2 \log n)$$

Que reafirma la conclusion expuesta con la expansión de recurrencia.

■  $T(n) = 7T(\frac{n}{3}) + n^2$  Usando expansión de recurrencia:

$$T(n) = \frac{n}{3} = 7(7T(\frac{n}{3^2}) + (\frac{n}{3})^2) + n^2$$

$$T(n) = \frac{n}{3} = 7^2 T(\frac{n}{3^2}) + \frac{7n^2}{3^2} + n^2$$

$$T(n) = \frac{n}{3^2} = 7^2 (7T(\frac{n}{3^3}) + (\frac{n}{3})^2) + \frac{7n^2}{3^2} + n^2$$

$$T(n) = \frac{n}{3^2} = 7^3 T(\frac{n}{3^3}) + \frac{7n^2}{3^3} + \frac{7n^2}{3^2} + n^2$$

$$T(n) \frac{n}{3^{i-1}} = 7^i T(\frac{n}{4^i}) + in^2$$

por lo tanto concluimos:

$$T(n) = 7T(\frac{n}{3}) + n^2 \text{ esta firmemente dominada por } \Theta(n^2)$$

Usando el Teorema Maestro:

$$T(n) = 7T(\frac{n}{3}) + n^2$$

donde:

$$a = 7, b = 3, f(n) = n^2 \text{ entonces}$$

$$\log_b a = \log_3 7 = 1,7712$$

Recae en el Caso 3 del Teorema Maestro:

$$n^2 = \Omega(n^{\log_3 7 + \epsilon}) \text{ para } \epsilon = 0,1$$

si ,  $C < 1$

$$7\left(\frac{n}{3}\right)^2 \leq Cn^2$$

$$\frac{7n^2}{9} \leq Cn^3$$

sea  $C = 0,9$  cumple con la propiedad

Entonces concluimos que:

$$T(n) = \Theta(n^2)$$

Que reafirma la conclusion expuesta con la expansión de recurrencia.

- $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$
- $T(n) = 4T\left(\frac{n}{2}\right) + n^2 \log n$

Usando el Teorema Maestro:

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2 \log n$$

donde:

$a = 4, b = 2, f(n) = n^2 \log n$  entonces

$$n^{\log_b a} = n^{\log_2 4} \text{ para un } \epsilon = \frac{1}{2}$$

Recae en el Caso 3 del Teorema Maestro:

$$n^3 = \Omega(n^{\log_2 4 + \epsilon})$$

$$n^3 = \Omega(n^{2 + \frac{1}{2}})$$

si ,  $C \leq 1$

$$4\left(\frac{n}{2}\right)^2 \leq Cn^2 \log n$$

$$\frac{n^3}{2} \leq Cn^3$$

sea  $C = \frac{1}{2}$  cumple con la propiedad

Entonces concluimos que:

$$T(n) = \Theta(n^2 \log n)$$

- $T(n) = T(n - 1) + n$
- $T(n) = \sqrt{n} + 1$   
No existe recursividad no se puede aplicar El teorema maestro ni la recurrencia de expansion

## 4. Problema 4

Se sabe que, para resolver el problema de ordenación de números, el algoritmo Mergesort posee una complejidad de  $\Theta(n \log n)$  y el Insertion Sort  $O(n^2)$ . Sin embargo, los factores constantes del algoritmo Insertion lo tornan más veloz para vectores de tamaño  $n$  pequeños. Por ende tiene sentido realizar una combinación de ambos, y utilizar el algoritmo de inserción cuando los problemas se tornen lo suficientemente pequeños. Considere la siguiente modificación del Mergesort:  $\frac{n}{k}$  sub listas de tamaño  $k$  son ordenadas utilizando el algoritmo de inserción, y luego combinadas utilizando el mecanismo del Mergesort (*Merge*), siendo  $k$  la variable a ser determinada.

*Descripción :*

- Presente el algoritmo, y en anexo coloque el código

Tenemos el siguiente algoritmo:

*mergeSortO* que tiene como entrada *Array, min, max, k*

$n \leftarrow \text{tamaño del Array}$

$\text{sin} \leq k$

entonces *MergeSortO*(*Array, min, max*)

caso contrario :

.  $\text{mid} \leftarrow (\text{max} - \text{min})/2$

. *MergeSortO*(*min, mid, limit*)

. *MergeSortO*(*mid + 1, max, limit*)

. *Merge*(*min, max, mid*)

Usamos una clase que engloba todos los metodos descritos en el algoritmo y la llamamos *PowerSort*

A continuación se muestra los codigos para los métodos de *Merge* e *InsertionSort*.

Cabe resaltar que más adelante se modificara los métodos de *Merge* y *MergeSort* les agregamos un distintivo *original* para no tener errores de compilación.

Todo el código está disponible en mi repositorio personal de GitHub [4]

```
template<typename adapter>
void PowerSort<adapter>::mergeSort0( int min, int max, int limit){
    if ((max - min + 1) <= limit){
        insertionSort( min, max);
    }
    else{
        int mid = (max + min) / 2;
        mergeSort0(min, mid, limit);
        mergeSort0(mid + 1, max, limit);
        mergeOriginal(min, max, mid);
    }
}
```

```

template <typename adapter>
void PowerSort<adapter>::mergeOriginal(int min, int max, int mid){
    int * temp = new int [max + 1];
    int i = min;
    int j = mid + 1;

    for (int index = min; index <= max; index++) {
        if (cmp(i,mid) && (j > max || cmp(arr[i], arr[j]))) {
            temp[index] = arr[i];
            i = i + 1;
        }
        else{
            temp[index] = arr[j];
            j = j + 1;
        }
    }

    for (int index = min ; index <= max ; index++)
        arr[index] = temp[index];
}

```

Figura 5: Código del Merge Original

```

template<typename adapter>
void PowerSort<adapter>::insertionSort( int min, int max) {
    int key;
    for (int j = min + 1; j <= max; j++){
        key = arr[j];
        int i = j - 1;

        while (i >= min && arr[i] > key){
            arr[i + 1] = arr[i];
            i--;
        }
        arr[i + 1] = key;
    }
}

```

Figura 6: Código del InsertionSort

Todo el código está disponible en mi repositorio personal de GitHub [4]

- En la práctica (realice un análisis estadístico) cual es el valor de  $k$ .
- Muestre que las  $\frac{n}{k}$  sub-listas, cada una de tamaño  $k$ , pueden ser ordenadas por el algoritmo Insertion en el peor caso en  $O(nk)$ .
- Muestre que las listas pueden ser combinadas en el peor caso en tiempo  $O(n \log(\frac{n}{k}))$ .
- Sea  $A = [1..n]$  un vector de números distintos. Si se cumple que  $i < j$  y  $A[i] > A[j]$ , entonces el par  $(i, j)$  es llamado de “inversión” del vector A. Modifique el algoritmo del Mergesort para encontrar las inversiones de un vector. En anexo del mismo problema coloque el código.

```

template<typename adapter>
void PowerSort<adapter>::mergeModified( int min, int max, int mid ){
    int i = min, j = mid + 1, k = 0, k1=k;
    int* temp = new int[max - min + 1];

    while ( i <= mid && j <= max ){
        if (cmp(arr[i] , arr[j]) && i < j){
            temp[k++] = arr[i++];
        }
        else
            k1++;
        temp[k++] = arr[j++];
    }

    while ( i <= mid )
        temp[k++] = arr[i++];

    while ( j <= max )
        temp[k++] = arr[j++];

    for ( i = min; i <= max; i++)
        arr[i] = temp[i - min];

    cout << endl;
    cout << k1 << " este es el numero de Inversiones " << endl;

    return;
}

```

Figura 7: Código del Merge Modificado



## 5. Problema 5

Para los problemas presentados, cree un código en C++, presente la complejidad del algoritmo del cual es instancia el código presentado.

*Descripción :*

- Desarrolle un programa “Recursivo” para generar la descomposición de un número entero positivo, en la suma de todos los posibles factores. La presentación de los factores debe estar ordenada de mayor a menor. Por ejemplo para  $n = 5$

5  
4 + 1  
3 + 2  
3 + 1 + 1  
2 + 2 + 1  
2 + 1 + 1 + 1  
1 + 1 + 1 + 1 + 1

A continuación el código para resolver el problema:

Ingresando un valor  $n = 5$  como en el problema se obtiene el mismo resultado, con el código compilado y ejecutado.

```
#include <bits/stdc++.h>
using namespace std;

void display(int* arr, int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i];
        if (i < n - 1)
            cout << "+";
    }
    cout << endl;
}

void combination(int n, int i=0, int arrSize=1) {
    static int* arr = new int[arrSize];
    if(n == 0)
        display(arr, i);
    else if (n > 0) {
        int k;
        for (k = arrSize; k > 0; k--) {
            arr[i] = k;
            combination(n - k, i + 1, k);
        }
    }
}

int main(){
    int n = 5;
    combination(n,0,n);
    return 0;
}
```

Figura 8: Código del problema 5 parte 1

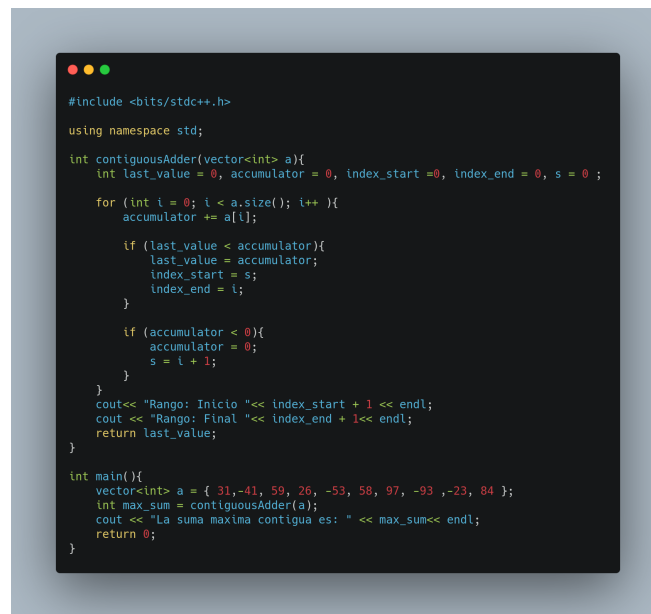
```
Actividades Terminal
(base) + ADA/C++/Practica 1 / master$ g++ practica.cpp -o Main
(base) + ADA/C++/Practica 1 / master$ ./Main
5
4+1
3+2
3+1+1
2+2+1
2+1+1+1
1+1+1+1+1
(base) + ADA/C++/Practica 1 / master$
```

Figura 9: Código compilado y ejecutado del problema 5 parte 1

- Dado un vector con  $n$  números enteros, determine la encontrada en un subvector contiguo de ese vector. Si todos los números fueran negativos asuma que la suma es 0. Por ejemplo en el siguiente vector  $A = [31, 41, 59, 26, 53, 58, 97, 9323, 84]$  la mayor suma de elementos contiguos es 187 que se encuentra en el rango de 3 a 7. El algoritmo del código a presentar debe tener complejidad lineal.

La complejidad de este algoritmo es de  $O(n)$ .

A continuación el resultado con el mismo ejemplo del enunciado.



```
#include <bits/stdc++.h>

using namespace std;

int contiguousAdder(vector<int> a){
    int last_value = 0, accumulator = 0, index_start = 0, index_end = 0, s = 0 ;

    for (int i = 0; i < a.size(); i++){
        accumulator += a[i];

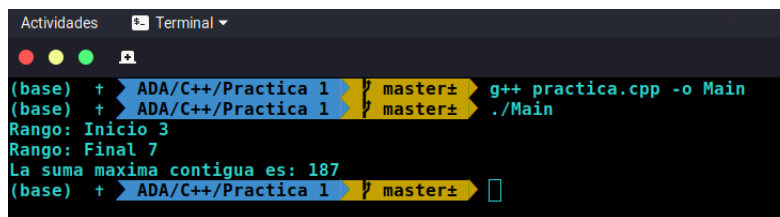
        if (last_value < accumulator){
            last_value = accumulator;
            index_start = s;
            index_end = i;
        }

        if (accumulator < 0){
            accumulator = 0;
            s = i + 1;
        }
    }

    cout<< "Rango: Inicio "<< index_start + 1 << endl;
    cout<< "Rango: Final "<< index_end + 1<< endl;
    return last_value;
}

int main(){
    vector<int> a = { 31,-41, 59, 26, -53, 58, 97, -93 ,-23, 84 };
    int max_sum = contiguousAdder(a);
    cout << "La suma maxima contigua es: " << max_sum<< endl;
    return 0;
}
```

Figura 10: Código del problema 5 parte 2



```
Actividades Terminal
(base) + ADA/C++/Practica 1 master± g++ practica.cpp -o Main
(base) + ADA/C++/Practica 1 master± ./Main
Rango: Inicio 3
Rango: Final 7
La suma maxima contigua es: 187
(base) + ADA/C++/Practica 1 master±
```

Figura 11: Código compilado y ejecutado del problema 5 parte 2

## 6. Conclusiones

A medida que se avanzaba con el trabajo era imperativo dar un vistazo a [2] y [3], ya con una idea sólida se podría abarcar cualquier pregunta o al menos la que requería menos esfuerzo para lograra avanzar en el trabajo, sin duda algunos conceptos en una primera vista quedan algo confusos como era el caso del Teorema Maestro, una de las dificultades es determinar cuando usar un caso para un determinado problema, es decir que requisitos debía cumplir un problema para aplicar un caso.

Una dificultad mayor fue el analisis y diseño del algoritmo de ordenamiento *MergeSort* combinado con el otro algoritmo de ordenamiento *InsertionSort*, por lo que no solamente se tenia que recurrir a las diapositivas [2] y [3] si no que también al libro [5] recomendado en clase, para poder entenderlo y lograr hacer un analisis lo más certero posible.

Ya con un poco de experiencia en los ejercicios anteriores sobre todo con el algoritmo de ordenamiento usando dos algoritmos *MergeInsertion*, la siguiente dificultad fue toparme con el Problema 5 primera parte, sin duda en un primer vistazo no se muestra intimidante, pero a medida que se va pensando en una posible solución y por la naturaleza especificada recursivo y esto confundía más, por que no solo era determinar los factores descompuestos del número de entrada, si no que también era imperativo tener un orden y ese orden era que dicha descomposición tendria que estar hecha de mayor a menor.

Para el último ejercicio se presento un algoritmo de orden lineal  $O(n)$  para tomar la mayor suma de un sub array de elementos contiguos, este no presento muchos problemas.

Para los programas que se pedían se utilizó un transformador de código a imagen [1] y para presentar el presente trabajo se usa un editor de Latex Online como lo es Overleaf.

## Referencias

- [1] carbon app. Create and share beautiful images of your source code in carbon.
- [2] Rensso Víctor Hugo Mora Colque. Análisis y diseño de algoritmos, 2020.
- [3] Antonio Alfredo Ferreira Loureiro. Projeto e análise de algoritmos análise de complexidade, 2020.

- [4] Jorge Luis Huanca Mamani. <https://github.com/jluisdeveloper/ADA/>. (ADA Repository). 2020.
- [5] Ronal L. Rivest Clifford Stein Thomas H. Cormen, Charles E. Leiserson. *Introduction to Algorithms*. 2020.