

# Cálculo del Tiempo de Ejecución

## Optimizando algoritmos

*Problema: encontrar el valor de la suma de la sub-secuencia de suma máxima*

# Problema de la subsecuencia de suma máxima


Dada una secuencia de números enteros, algunos negativos:

$$a_1, a_2, a_3, \dots, a_n$$

encontrar el valor máximo de la  $\sum_{k=i}^j a_k$

Por convención, la suma es cero cuando todos los enteros son negativos.

Ej.: -2, 11, -4, 13, -5, -2 la respuesta es 20



# Suma de la sub-secuencia de suma máxima

## Versión 1: $O(n^3)$

```
public final class MaxSumTest
{
    /* Cubic maximum contiguous subsequence sum algorithm. */
    public static int maxSubSum1( int [ ] a )
    {
        int maxSum = 0;
        for( int i = 0; i < a.length; i++ )
            for( int j = i; j < a.length; j++ )
            {
                int thisSum = 0;
                for( int k = i; k <= j; k++ )
                    thisSum += a[ k ];
                if( thisSum > maxSum )
                    maxSum = thisSum;
            }
        return maxSum;
    }
}
```

# Suma de la sub-secuencia de suma máxima

## Versión 2: $O(n^2)$

```
public final class MaxSumTest
{
    /* Quadratic maximum contiguous subsequence sum algorithm. */
    public static int maxSubSum1( int [ ] a )
    {
        int maxSum = 0;
        for( int i = 0; i < a.length; i++ )
            for( int j = i; j < a.length; j++ )
            {
                int thisSum = 0;
                for( int k = i; k <= j; k++ )
                    thisSum += a[ k ];
                if( thisSum > maxSum )
                    maxSum = thisSum;
            }
        return maxSum;
    }
}
```

The diagram illustrates the execution of the nested loops in the `maxSubSum1` method. A red line connects the `for( int i = 0; i < a.length; i++ )` loop to a box containing `int thisSum = 0;`, indicating that `thisSum` is reset to 0 at the start of each iteration of the `i` loop. Another red line connects the `for( int k = i; k <= j; k++ )` loop to a box containing `thisSum += a[ k ];`, indicating that `thisSum` is updated with the value of `a[k]` for each iteration of the `k` loop.

# Suma de la sub-secuencia de suma máxima

## Versión 2: $O(n^2)$

```
public final class MaxSumTest
{
    /* Quadratic maximum contiguous subsequence sum algorithm. */
    public static int maxSubSum2( int [ ] a )
    {
        int maxSum = 0;
        for( int i = 0; i < a.length; i++ )
            int thisSum = 0;
            for( int j = i; j < a.length; j++ )
            {
                thisSum += a[ j ];
                if( thisSum > maxSum )
                    maxSum = thisSum;
            }
        return maxSum;
    }
}
```

# Suma de la sub-secuencia de suma máxima

## Versión 3: $O(n \cdot \log n)$

/\*\* Solución recursiva:

\* Explicar la resolución detallada y graficamente

\*/

## Suma de la sub-secuencia de suma máxima - Versión 3: $O(n \cdot \log n)$

```
private static int maxSumRec( int [ ] a, int left, int right )
{
    if( left == right ) // Base case
        if( a[ left ] > 0 )
            return a[ left ];
        else
            return 0;
    int center = ( left + right ) / 2;
    int maxLeftSum = maxSumRec( a, left, center );
    int maxRightSum = maxSumRec( a, center + 1, right );

    int maxLeftBorderSum = 0, leftBorderSum = 0;
    for( int i = center; i >= left; i-- )
    {
        leftBorderSum += a[ i ];
        if( leftBorderSum > maxLeftBorderSum )
            maxLeftBorderSum = leftBorderSum;
    }
    int maxRightBorderSum = 0, rightBorderSum = 0;
    for( int i = center + 1; i <= right; i++ )
    {
        rightBorderSum += a[ i ];
        if( rightBorderSum > maxRightBorderSum )
            maxRightBorderSum = rightBorderSum;
    }
    return max3( maxLeftSum, maxRightSum, maxLeftBorderSum + maxRightBorderSum );
}
```

# Suma de la sub-secuencia de suma máxima

## Versión 3: $O(n \cdot \log n)$

```
/**
 * Driver for divide-and-conquer maximum contiguous
 * subsequence sum algorithm. */

public static int maxSubSum3( int [ ] a ) {
    return maxSumRec( a, 0, a.length - 1 );
}

/* END */

/**      * Return maximum of three integers.      */

private static int max3( int a, int b, int c ) {
    return a > b ? a > c ? a : c : b > c ? b : c;
}
```



## Versión 3: Función de Tiempo de Ejecución

$$T(n) = \begin{cases} \text{cte}_1 & n = 1 \\ 2 * T(n/2) + n + \text{cte}_2 & n > 1 \end{cases}$$

2 llamadas recursivas

Buscar la sub-secuencia de suma máxima de cada mitad que incluye los elementos centrales

# Suma de la sub-secuencia de suma máxima

## Versión 4: $O(n)$

```
/** Linear-time maximum contiguous subsequence sum
    algorithm. */
public static int maxSubSum4( int [ ] a )
{
    int maxSum = 0, thisSum = 0;
    for( int j = 0; j < a.length; j++ )
    {
        thisSum += a[ j ];
        if( thisSum > maxSum )
            maxSum = thisSum;
        else if( thisSum < 0 )
            thisSum = 0;
    }
    return maxSum;
}
```