```
1: function SelectionSort(A[],n)
2:     if n < 1 then
3:         return
4:     end if
5:     for i = 1 to n − 1 do
6:         if A[i] > A[n] then
7:             Swap(A[i],A[n])
8:         end if
9:     end for
10:    SelectionSort(A[],n − 1)
11: end function
```

Caso Base

Comparamos el elemento i-esimo con el ultimo elemento y si el elemento i-esimo es más grande que el ultimo los cambiamos

Estos intercambios se harán continuamente y terminado se tendrá El maximo elemento del array en la posición final

En cada recursión estaremos encontrando el máx elemento de ese array que cada vez se esta acortando

Vamos a Analizarlo:

↳ $T(n)$ → tiempo de ejecución del selectionSort en un array de tamaño "n"

```
1: function SelectionSort(A[],n)
2:     if n < 1 then
3:         return
4:     end if
5:     for i = 1 to n − 1 do
6:         if A[i] > A[n] then
7:             Swap(A[i],A[n])
8:         end if
9:     end for
10:    SelectionSort(A[],n − 1)
11: end function
```

C

Constante C → $C \sum_{i=1}^{n-1} 1 = C(n-1)$

$T(n-1)$

$T(n) = C(n-1) + T(n-1)$

Caso Base: $T(0) = C$

$T(n) = C(n-1) + T(n-1)$

Recursionando la recursión

$T(n-1) = C(n-1-1) + T(n-1-1)$

$T(n-1) = C(n-2) + T(n-2)$

$T(n) = C(n-1) + [C(n-2) + T(n-2)]$

$T(n-2) = C(n-3) + T(n-3)$

$T(n) = C(n-1) + C(n-2) + [C(n-3) + T(n-3)]$

$T(n) = C(n-1) + C(n-2) + C(n-3) + T(n-3)$

$T(n) = C(n-1) + C(n-2) + \dots + C(n-k) + T(n-k)$ } Para algún $k \geq 1$

Obs:

Caso Base $T(0) = C$

Eligiremos un k tq $n-k=0$

$n = k$

$T(n) = \underbrace{C(n-1) + C(n-2) + \dots + C(n-n)}_{n \text{ terms.}} + T(0)$

$n-1, \ n-2, \ \dots, \ 1, \ 0$

$T(0) \to C$

↳ $T(n) = C[0 + 1 + \dots + n-2 + n-1]$ → Suma aritmetica

$T(n) = \frac{(n-1)(n)}{2} C + C \Rightarrow T(n) \in \Theta(n^2)$ //

```
1: function SelectMax(A[],n)
2:     if n = 1 then
3:         return A[1]
4:     end if
5:     for i = 1 to ⌊n/2⌋ do
6:         A[i] =Max(A[i],A[n − i + 1])
7:     end for
8:     x =SelectMax(A[],⌈n/2⌉)
9:     return x
10: end function
```

→ Función Recursiva, para obtener el maximo elemento de un array

↳ Caso Base $T(1) = c$

→ c } $c\left(\frac{n}{2}\right)$

→ $T\left(\frac{n}{2}\right)$

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

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

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

$$T(n) = \frac{cn}{2} + \frac{cn}{4} + \frac{cn}{8} + T\left(\frac{n}{8}\right)$$

$$\frac{1}{2}\sum_{i=0}^{k-1} \frac{cn}{2^i} + T\left(\frac{n}{2^k}\right)$$

Queremos que

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \lg n$

$$\frac{cn}{2}\sum_{i=0}^{k-1} \frac{1}{2^i} + T(1)$$ (c)

$$T(n) = c\frac{n}{2}\sum_{i=0}^{k-1}\left(\frac{1}{2}\right)^i + c$$

Acotaremos esto a una suie geometrica

$$\sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^i = \frac{1}{1-1/2}$$

$$T(n) \leq cn\sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^i + c$$

$$cn\left(\frac{1}{1-1/2}\right) + c$$

$$T(n) \leq 2cn + c \} \; T(n) \in O(n)$$

Inferior

$cn(1) + c \leq Tn \} \; T(n) \in \Omega(n)$

$\therefore T(n) \in \Theta(n)$ □