Practica Divide y Venceras

1. MaxMin

Para abordar este problema, utilizamos un simple divide y venceras, en el que cada iteración, el vector se divide en 2.

Calculamos la Eficiencia:

```
Para T(n) = 2 T(n/2) +1

Utilizando la fórmula maestra donde:

a = "Llamadas recursivas"
b = "Tamaño de los subproblemas"
p = "función de mezcla"

Para este caso:
  a = 2
  b = 2
  p = 1

Según la formula maestra:

  O(n^log^a(en base "b")) si a > b^p
  O(n^p*logn)             si a = b^p
  O(n^p)                  si a < b^p

En este caso 2 = 2 , y por tanto, O(n) = n*logn

En el mejor caso, en el que nos encontramos que sea 1 elemento tanto el máximo
como el mínimo, Ω(n) = 1.

Para el caso promedio, ya que al igual que en primer caso seguimos recorriendo
el vector, encontramos θ(n) = n*logn
```

2. MaxMin Matriz

En este caso el problema aborda una matriz y se utiliza un divide y venceras, en el cual, se divide el problema entre 4 por cada iteración.

Calculamos la Eficiencia:

```
Para T(n) = 2 T(n/2) +1

Utilizando la fórmula maestra donde:

a = "Llamadas recursivas"
```

```
b = "Tamaño de los subproblemas"
p = "función de mezcla"

Para este caso:
  a = 4
  b = 4
  p = 1

Según la formula maestra:

  O(n^p*logn) si a = b^p

  Ya que 4 = 4 , por tanto O(n) = n*logn

  El mejor caso, en el que nos encontramos que sea 1 elemento tanto el máximo
  como el mínimo, Ω(n) = 1.

  Para el caso promedio, ya que al igual que en primer caso seguimos recorriendo
  toda la matriz, encontramos θ(n) = n*logn
```

3. Zapataos / Pies

Este problema lo abordamos utilizando el algoritmo llamado "Pivote" junto al Divide y Venceras.

Calculamos la Eficiencia:

```
Para T(n) = 2 T(n/"2") + 2n

  "2"= Ya que suponemos que en el peor caso el vector se partira cerca de
       la mitad.
  "2n"= Ya que cada vez que llamamos a Partición se recorre el vector, al llamarlo
       2 veces, 2*n.

Utilizando la fórmula maestra donde:

a = "Llamadas recursivas"
b = "Tamaño de los subproblemas"
p = "función de mezcla"

Para este caso:
  a = 2
  b = 2
  p = 2

Según la fórmula maestra:
O(n^p) = a < b^p
O(n^2) = 2 < 4

Por tanto O(n^2).

Para el mejor caso, suponemos que ambos vectores tendran un único elemento y,
por tanto, Ω(n) = 1.
```

```
Para el caso promedio, solo cambiaría que el vector en vez de partirse por la mitad
se partieras en partes dispares, pero aun así se acabba recorriendo el vector entero
en ambos casos. Por tanto, θ(n) = n^2
```

4. Moda

Este algoritmo, tiene varias soluciones del cual hemos implementado dos:

Vector sin ordenar

En este caso estamos usando una función externa llamada *Frecuencia*

```
int Frecuencia(std::vector<int> &v, int p, int ini, int fin)
```

que es la encargada de devolver las n ocurrencias del vector (recorre desde i hasta n buscando las n ocurrencias del vector). Cuya eficiencia es de:

```
O(n)
```

y dentro de la moda estamos recorriendo hasta n el vector por lo tanto nuestro algoritmo, si juntamos las dos O tenemos una eficiencia de:

```
O(n²)
```

Vector ordenado

Basicamente creamos un vector desordenado y calculando la moda (inline), en un solo recorrido desde ini a fin tenemos un coste de:

```
O(n)
```

Pero como en este caso lo estamos ordenando antes de llamar la moda la complejidad de este algoritmo en promedio recorre la distancia de ini hasta fin, y realizara n*Log2(n) comparaciones y swaps en el vector, siendo n la distancia. Asi que la eficiencia es de:

```
O(nlogn)
```