BINARY\_INTERVAL (v, n, leftInterval, rightInterval)

//this function takes a vector and its length and the margins of the interval.

//uses the binary\_search method to find if there are an values in the mentioned interval

//in the vector. It returns true or false based on that

leftMargin<-1 O(1)

rightMargin<-n O(1)

//set the margins where to look at the beginning and end of the vector

while leftMargin<=rightMargin \*

//continue until there is less than an element to check

m<-(leftMargin+rightMargin)/2; O(log n)

//set the middle

if v[m]>=leftInterval and v[m]<=rightInterval O(log n)

//if the mid value is in the interval, return true

return true O(1)

else if v[m]>rightInterval O(log n)

//if the mid value is greater than the most right point of the interval

//look in the right side of the middle point

rightMargin<-m-1 O(log n)

else if v[m]<leftInterval O(log n)

//if the mid value is less than the most left point of the interval

//look in the left side of the middle point

leftMargin<-m+1 O(log n)

//if there is no values from the interval in the vector, return false

return false O(1)

O (n): O (6log n+4) = O (log n)

\*the exit condition depends on the left and right margins. Given that log n that comes from always finding the middle point between the two, everything that is in the while loop has a O (log n) complexity.

Big O has a logarithmic complexity of O (log n)