

UNIVERSIDAD DE ORIENTE

NÚCLEO DE ANZOÁTEGUI

DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS

CÁTEDRA: Estructura de Datos

SECCIÓN 01

**Big 0**

Profesor: Integrantes:

Gabriela Veracierta. Br. Luis Agreda C.I: 26.886.935

Br.Gabriel Trujillo C.I 25.244.508

Barcelona, febrero de 2022

**Pregunta: diga con sus propias palabras ¿Por qué es importante considerar la eficiencia en el diseño de algoritmos?**

Es importante porque el hecho de crear una solución a un problema no requiere que solo funcione y sea sencilla, esta debe tener un rendimiento óptimo. Tomando en cuenta que desde que existen una inmensa cantidad de procesadores y maquinas computacionales, al momento de tomar en cuenta el rendimiento y la complejidad del programa, no se puede medir en tiempo, así que se mide la cantidad de datos y cómo se comporta ante el algoritmo conforme crecen los datos, y para eso está la notación Big O. Para portar eficiencia a la hora de diseñar algoritmos y que así se utilice la menor cantidad de espacio y se procese en el menor tiempo posible.

**Calcular la función O grande de eficiencia de las siguientes funciones:**

1. f(n) = n (n+1)/2 = 1/2 n2 + 1/2 n

Después de resolver la ecuación, y teniendo en cuenta el mayor valor, que en dado caso sería tomar en cuenta el peor caso posible, su función O grande es O(n2).

1. f(n) = ajnk + aj-1nk-1 + ... + a2n2 + a1n + a0

dejando un lado los coeficientes, se puede determinar que la eficiencia de la función es O(nk)

**Determinar la complejidad de la función:**

**a**.-

double mayor(double x, double y){

if (x > y) // 0(1)

return x;

else

return y;

}

Cualquier función se considera Big 0 de 1 0 constante siempre y cuando no sea un ciclo o recursión o llamadas a otra función (que no sea constante) por ende el resultado su complejidad es O(1).

**b**.- void escribeVector(doublex[], int n)

{

int j; //o(1)

for (j = 0; j < n; j++) // 0(n)

{

cout << x[j]; // O(1)

}

}

La complejidad del ejercicio es O(n) debido que repite una sentencia constante n veces se podría decir que 0(n) + 0(1), eligiendo el peor escenario la complejidad seria: O(n)

**c**.- double suma(double d[], int n){

int k ; // O(1)

k = s = 0; // O(1)

while (k < n) //O(n)

{

s += d[k];

if (k == 0)

k = 2;

else

k \*= 2; //0(log n)

}

return s;

}

La complejidad del ejercicio es: 0(log n) debido que la variable del ciclo se va multiplicando se convierte en (log n).

**d**.-void traspuesta(float d[][M], int n)

{

int i, j; // O(1)

for (i = n - 2; i > 0; i--){ //O(n)

for (j = i + 1; j < n; j++){ //O(n2)

float t;

t = d[i][j];

d[i][j] = d[j][i];

d[j][i] = t;

}

}

}

La complejidad del ejercicio es O(n2) debido que repite una sentencia constante n veces durante dos ciclos, sobre sentencia constantes.

**e**-. Escribir un algoritmo que indique si una palabra leída del teclado es un palíndromo. Un palíndromo (capicúa) es una palabra que se lee igual en ambos sentidos como “radar”. Calcule su complejidad.

void palindromo(string palabra){

int i=0,j=0; // 0(1)

bool palindromo = false;

for(i=0;i<palabra.length();i++){ // 0(n)

for(j=palabra.length()-1;j>=0;j--){ // 0(n2)

if(palabra[i]==palabra[j]){ // 0(1)

palindromo = true;

}

else{

palindromo = false;

}

}

}

if (palindromo == false) // 0(1)

cout<<"no es palindromo"<<endl;

else

cout <<"es palindromo"<<endl;

}

La complejidad del ejercicio es O(n2)

**f**-. Escribir un algoritmo que cuente el número de ocurrencias de cada letra en una palabra leída como entrada. Por ejemplo, “Mortimer” contiene dos “m”, una “o”, dos “r”, una “y”, una “t” y una “e”. Calcule su complejidad.

void Concurrencia(string palabra){

string letras="";

int contador,i;

bool rep; //O(1)

for( char c : palabra ){ // 0(n)

contador =0;

for( i=0; i<=palabra.size(); i++){ // 0(n^2)

if( c == palabra[i] || c== palabra[i] +32|| c == palabra[i] -32){

contador++;

}

}

for( char j : letras){ // 0(n^2)

rep = false; //O(1)

if ( c == j) //O(1)

rep = true;

}

letras = c + letras; //O(1)

if(rep==false) //O(1)

cout <<"La letra " <<c<<" se repite: "<<contador<<" veces"<<endl; //O(1)

}

}

Dado que hay más de un ciclo, después de quitar los coeficientes y elegir la complejidad mayor. La función tiene una complejidad de O(n2).