**Taller algoritmos y C++**

Análisis de complejidad por conteo en algoritmos de fuerza bruta para determina el peor caso:

1. Exponente por fuerza bruta:

**Nota:** suponiendo que exponente = n entonces:

int exponenciacion (int base, int exponente) {

int resultado = 1;------------------------------------------------- **1**

for (int i = 1; i <= exponente; i++){ -------------------------- **n+1**

resultado \*= base; --------------------------------------- **n**

}

return resultado; ------------------------------------------------- **1**

}

**Nota:** Añadiendo la asignación que está en el ciclo for de la variable i se agrega **1** paso más, además de un incremento en la misma variable de **n** pasos.

T(n) = 1 + (n + 1) + n + 1 + 1 + n = 3n + 4

T(n) = 2n + 4 entonces este algoritmo en el peor caso tiene una complejidad lineal representada de la forma O(n).

2. Búsqueda secuencial:

**Nota:** suponiendo que tamanio = n entonces:

int busqueda\_secuencial (int array[], int tamanio, int elemento ) {

int i = 0; ------------------------------------------------------------- **1**

while (i < tamanio) { ---------------------------------------------- **n + 1**

if (array[i] == elemento) { -------------------------------- **n**

return i; ---------------------------------------------- **1**

}

i++; ---------------------------------------------------------- **n**

}

return -1; ------------------------------------------------------------ **0**

}

T(n) = 1 + (n+1) + n + 1 + n + 0 = 3n + 3

T(n) = 3n + 3 entonces este algoritmo en el peor caso tiene una complejidad lineal representada de la forma O(n).

3. Ordenamiento burbuja:

**Nota:** suponiendo que tamanio = n entonces:

void ordenamiento\_burbuja (int arreglo[], int tamanio){

for (int i = 0; i <= tamanio - 2; i++) { ---------------------------- **n**

for (int j = 0; j <= tamanio - 2 - i; j++) {------------------ []- **1**  
 if ( arreglo[j+1] < arreglo[j]) { -------------------- []- **2**

int tmp = arreglo[j]; ------------------------- **2n**

arreglo[j] = arreglo[j+1]; ------------------- **2n**

arreglo[j+1] = tmp; -------------------------- **2n**

}

}

}

}

**Nota:** se añaden las asignaciones de los ciclos for, ósea, 1 paso por la asignación de la variable i y 1 paso por la asignación de la variable j

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

T(n) = + + 7n - 1  
  
T(n) = + + 7n – 1

T(n) = + 7n – 1 = + n + 7n – 1 = + 8n -1

T(n) = + 8n – 1 entonces este algoritmo en el peor caso tiene una complejidad cuadrática representada de la forma O().

4. Ordenamiento de selección:

**Nota:** suponiendo que tamanio = n entonces:

void ordenamiento\_seleccion (int arreglo2[], int tamanio){

for (int i = 0; i <= tamanio - 2; i++) { ------------------------------- **n**

int min = i; ------------------------------------------------------- **n - 1**

for (int j = i + 1; j <= tamanio - 1 ; j++) { --------------- []- **1**

if (arreglo2[j] < arreglo2[min]) { ----------------- []- **2**

min = j; ---------------------------------------- **(n - 1)\*(n – 1)**

}

}

int tmp = arreglo2[min]; ----------------------------------------- **n - 1**

arreglo2[min] = arreglo2[i]; ------------------------------------- **n - 1**

arreglo2[i] = tmp; ------------------------------------------------- **n - 1**

}

}

**Nota:** se añaden las asignaciones de los ciclos for, ósea, 1 paso por la asignación de la variable i y 1 paso por la asignación de la variable j

T(n) = n + (n – 1) + [ ]- 1 + [ ]- 2 + [(n - 1)\*(n – 1)] +(n – 1)..

..+ (n – 1) + (n – 1) + 1 + 1

T(n) = 5n – 5 + [] + [] + [-2n + 1]

T(n) = 5n – 5 + + n + – 2n + 1

T(n) = 2 + 4n - 4 entonces este algoritmo en el peor caso tiene una complejidad cuadrática representada de la forma O().

5. Emparejamiento de cadenas:

int emparejamiento\_cadenas (string texto, string linea){

int n = texto.length(); ------------------------------------------------- **1**

int m = linea.length(); ------------------------------------------------ **1**

for (int i = 0; i < n-m; i++) { ---------------------------------------- **n – m + 1**

int j = 0; -------------------------------------------------------- **n - m**

while ( j < m && linea[j] == texto[i + j]) { --------------- **m**(**n – m)**

j++; ------------------------------------------------------ **m**

}

if ( j == m ) { --------------------------------------------------- **1**

return i; -------------------------------------------------- **1**

}

}

return -1; --------------------------------------------------------------- **0**

}

**Nota:** se añaden las asignaciones del ciclo for, ósea, 1 paso por la asignación de la variable i.

T(n) = 1 + 1 + n – m + 1 + n – m + mn - + m + 1 + 1 + 0 + 1

T(n) = + mn + 2n - m + 6 entonces este algoritmo en el peor caso tiene una complejidad cuadrática representada de la forma O().