Trabajo Autónomo - Estructuras de datos

Nombre: Ariel Alejandro Calderón
Tema: Análisis de Algoritmos: Encontrar f(n) y O(n)
Fecha: 24/02-2025

1. Estructura Condicional Anidada

Código:

```
if (n1 > n2) {
    if (n1 > n3) {
        if (n2 > n3) {
            mostrar(n1, n2, n3);
        }
    }
}
```

Análisis:

- Las estructuras `if` son evaluaciones condicionales que se ejecutan en **tiempo constante**.
- Cada `if` se evalúa de forma **independiente** sin bucles ni recursiones.

Función de tiempo:

f(n) = c (constante)

Notación Big O:

O(1)

---

2. Algoritmo de Ordenamiento (Tipo Bubble Sort)

Código:

```
for (i = 0; i < n - 1; i++) {
    for (j = i + 1; j < n; j++) {
        if (V[i] > V[j]) {
            aux = V[i];
            V[i] = V[j];
            V[j] = aux;
        }
    }
}
```

Análisis:

- El **primer for** se ejecuta **(n - 1)** veces.
- El **segundo for** se ejecuta en promedio (n - i) veces.
- Esto genera una **serie aritmética**:
  (n - 1) + (n - 2) + ... + 1 = n(n - 1)/2

Función de tiempo:

f(n) = n(n - 1)/2

Notación Big O:

$O(n^2)$

---

3. Inserción Binaria con Desplazamientos

Código:

```
for (i = 1; i < n; i++) {
    temp = v[i];
    Izq = 0;
    Der = i - 1;
    while (Izq <= Der) {
        Medio = (Izq + Der) / 2;
        if (temp < v[Medio]) {
            Der = Medio - 1;
        } else {
            Izq = Medio + 1;
        }
    }
    for (j = i - 1; j >= Izq; j--) {
        v[j + 1] = v[j];
    }
    v[Izq] = temp;
}
```

Análisis:

- **Búsqueda binaria:** O(log n) por iteración.
- **Desplazamientos:** En el peor caso, O(n) por iteración.
- **Total:** Se realizan n iteraciones, cada una con búsqueda binaria y desplazamiento lineal.

Función de tiempo:

f(n) = n(log n + n)

Notación Big O:

$O(n^2)$

---

4. Ordenamiento por Mezcla (Merge Sort)

Código Resumido:

```
void ordenarMezcla(TipoEle A[], int izq, int der) {
    if (izq < der) {
        int centro = (izq + der) / 2;
        ordenarMezcla(A, izq, centro);
        ordenarMezcla(A, centro + 1, der);
        intercalar(A, izq, centro, der);
    }
}
```

Análisis:

- **División del arreglo:** Se realizan log n divisiones.
- **Intercalación:** Se hace en tiempo lineal O(n) en cada nivel de recursión.

Función de tiempo:

f(n) = n log n

Notación Big O:

O(n log n)

---

5. Búsqueda Lineal

Código:

```
for (i = j = 0; i < N; i++) {
    if (array[i] == elemento) {
        solucion[j] = i;
        j++;
    }
}
```

Análisis:

- El **for** recorre todos los elementos del arreglo.
- La búsqueda es **lineal**.

Función de tiempo:

f(n) = n

Notación Big O:

O(n)

---

6. Recorrido de ArrayList

Código:

```
public String recorrido2(ArrayList it, String msg) {
    int i = 0;
    String r = msg + "\n";
    while (i < it.size()) {
        r += "\n" + it.get(i).toString();
        i++;
    }
    return r;
}
```

Análisis:

- **Recorre todo el ArrayList:** n iteraciones.
- **Acceso a elementos:** O(1).

Función de tiempo:

f(n) = n

Notación Big O:

O(n)

---

7. Recorrido en Árbol Binario por Nivel

Código:

```
public void imprimirNivel(NodoArbol pivote, int nivel2) {
    if (pivote != null) {
        niveles[nivel2] = pivote.valor + ", " + ((niveles[nivel2] != null) ?
niveles[nivel2] : "");
        imprimirNivel(pivote.derecha, nivel2 + 1);
        imprimirNivel(pivote.izquierda, nivel2 + 1);
    }
}
```

Análisis:

- **Recorrido completo del árbol:** Cada nodo se visita una vez.
- Para un árbol con n nodos:

Función de tiempo:

f(n) = n

Notación Big O:

O(n)

---

8. Inserción en Lista Enlazada por Referencia

Código:

```
public void insercionReferencia(Nodo nuevo, int pos) {
    if (this.cab == null) this.insercionCab(nuevo);
    else {
        if (pos <= 0) this.insercionCab(nuevo);
        else if (pos >= n) this.insercionFinal(nuevo);
        else {
            Nodo p = this.cab;
            for (int i = 0; i < pos; i++) {
                p = p.getSig();
            }
            nuevo.setSig(p);
            this.n++;
        }
    }
}
```

Análisis:

- **Peor caso:** Insertar al final requiere recorrer la lista entera.
- **Recorrido:** O(n).

Función de tiempo:

f(n) = n

Notación Big O:

O(n)

---

9. Construcción de Árbol desde Archivo (Adivina)

Código Resumido:

```
public Adivina() {
    raiz = d.deserializeNodo("animal.txt");
}
```

Análisis:

- **Lectura del archivo:** Depende del número de líneas/nodos n.
- **Construcción del árbol:** Cada línea se procesa una vez.

Función de tiempo:

f(n) = n

Notación Big O:

O(n)

---

10. Menú Interactivo con Scanner

Código:

```
while (ope != 3) {
    System.out.println("Menu");
    ope = scc.nextInt();
    switch (ope) {
        case 1: a1 = l.construir(a1); break;
        case 2: System.out.println(a1.toString()); break;
    }
}
```

Análisis:

- **Menú:** Ejecuta operaciones simples (constantes).
- El número de iteraciones **depende del usuario**, pero cada operación es O(1).

Función de tiempo:

f(n) = c

Notación Big O:

O(1)