Análisis de clase hexadecimal

hexadecimal::hexadecimal(long valor)

Línea	Instrucción	Complejidad espacial	Complejidad temporal
1	`int i, temp = 0;`	O(1)	O(1)
2	`for(i = 0; i < 100; i++) hexa[i] = '0';`	O(100)	O(100)
3	`i = 99;`	O(1)	O(1)
4	`while(valor > 0) {`	O(1)	O(log16(valor))
5	`temp = valor % 16;`	O(1)	O(1)
6	`if(temp < 10) {`	O(1)	O(log16(valor))
7	`hexa[i] = temp + 48;`	O(1)	O(1)
8	`i--;`	O(1)	O(1)
9	`}`	O(1)	O(log16(valor))
10	`else {`	O(1)	O(log16(valor))
11	`hexa[i] = temp + 55;`	O(1)	O(1)
12	`i--;`	O(1)	O(1)
13	`}`	O(1)	O(log16(valor))
14	`valor = valor / 16;`	O(1)	O(1)
15	`}`	O(1)	O(log16(valor))

Explicación:

- La complejidad espacial del constructor es **O(100)** debido a la inicialización del array `hexa` con 100 caracteres.
- La complejidad temporal del constructor es **O(log(valor))** debido al bucle `while` que se ejecuta un número de veces proporcional al logaritmo en base 16 del valor inicial.
- Las demás instrucciones del constructor tienen una complejidad espacial y temporal de **O(1)**.

hexadecimal hexadecimal::operator+(const hexadecimal &op2) const

Línea	Instrucción	Complejidad espacial	Complejidad temporal
1	`hexadecimal temp;`	O(1)	O(1)
2	`int sum = 0;`	O(1)	O(1)
3	`int acarreo = 0;`	O(1)	O(1)
4	`for (int i = 99; i >= 0; i--) {`	O(1)	O(100)
5	`int digit1 = (hexa[i] >= '0' && hexa[i] <= '9') ? hexa[i] - '0' : hexa[i] - 'A' + 10;`	O(1)	O(1)
6	`int digit2 = (op2.hexa[i] >= '0' && op2.hexa[i] <= '9') ? op2.hexa[i] - '0' : op2.hexa[i] - 'A' + 10;`	O(1)	O(1)
7	`sum = digit1 + digit2 + acarreo;`	O(1)	O(1)
8	`if (sum >= 16) {`	O(1)	O(1)
9	`acarreo = sum / 16;`	O(1)	O(1)
10	`sum = sum % 16;`	O(1)	O(1)
11	`} else {`	O(1)	O(1)
12	`acarreo = 0;`	O(1)	O(1)
13	`}`	O(1)	O(1)
14	`if (sum < 10) {`	O(1)	O(1)
15	`temp.hexa[i] = sum + '0';`	O(1)	O(1)
16	`} else {`	O(1)	O(1)
17	`temp.hexa[i] = sum - 10 + 'A';`	O(1)	O(1)
18	`}`	O(1)	O(1)
19	`}`	O(1)	O(100)

20	`return temp;`	O(1)	O(1)

Explicación:

- La complejidad espacial del operador `+` es **O(100)** debido a la creación del array temporal `temp`.
- La complejidad temporal del operador `+` es **O(100)** debido al bucle `for` que se ejecuta 100 veces.
- Las demás instrucciones del operador `+` tienen una complejidad espacial y temporal de **O(1)**.

Complejidad espacial y temporal del operador de salida `<<` sobrecargado

Línea	Instrucción	Complejidad espacial	Complejidad temporal
1	`ostream& operator<<(ostream &salida, const hexadecimal &num)`	O(1)	O(1)
2	`int i;`	O(1)	O(1)
3	`for(i = 0; (num.hexa[i] == '0') && (i < 100); i++);`	O(1)	O(100)
4	`/*ignora ceros a la izquierda*/`	O(1)	O(1)
5	`if(i == 99)`	O(1)	O(1)
6	`salida << 0;`	O(1)	O(1)

7	`else`	O(1)	O(1)
8	`for( ; i < 100; i++)`	O(1)	O(100 - i)
9	`salida << num.hexa[i];`	O(1)	O(1)
10	`}`	O(1)	O(1)
11	`return salida;`	O(1)	O(1)

Explicación:

- La complejidad espacial del operador de salida `<<` es **O(1)**.
- La complejidad temporal del operador de salida `<<` es **O(100)** en el peor de los casos, cuando el número hexadecimal no tiene ceros a la izquierda. En el mejor de los casos, cuando el número hexadecimal tiene muchos ceros a la izquierda, la complejidad temporal es **O(1)**.