NICOLLE RUIZ QUINTERO.

ESTRUCTURA DE DATOS - INFORME COMPLEJIDAD TAD BIGINTEGERS.

NOTAS GENERALES: N SE REFIERE A LA CANTIDAD DE ELEMENTOS.

//CONSTRUCTORAS.

BigInteger(const string& elem);

La complejidad del constructor BigInteger es O(n). Ya que en todos los casos debe recorrer cada posicion del string para convertirlo a entero y después guardarlo en el vector.

BigInteger(const BigInteger& elem2);

la complejidad de la operación sería O(n). Esto se debe a que se copian los elementos del vector valor de elem2 al vector valor del objeto actual, lo cual implica recorrer y copiar cada elemento del vector.

//MODIFICADORAS

void add(BigInteger& elem);

Su complejidad es O(n) ya que utiliza la sobrecarga del + para realizar esta operación.

void product(BigInteger& elem);

Su complejidad es O(n^2) ya que utiliza la sobrecarga del * para realizar esta operación.

void substract(BigInteger& elem);

Su complejidad es O(n) ya que utiliza la sobrecarga del - para realizar esta operación.

void quotient(BigInteger& elem);

Su complejidad es O(n) ya que utiliza la sobrecarga del /para realizar esta operación.

void remainder(BigInteger& elem);

Su complejidad es O(n) ya que utiliza la sobrecarga del % para realizar esta operación.

void pow(int elem);

Su complejidad es O(n^3) ya que primero tenemos el ciclo for, y dentro de este esta la multiplicacion haciendo que aumente la complejidad.

static BigInteger sumarListaValores(deque<BigInteger> bigg);

Su complejidad es O(m*n) ya que tenemos el bucle for que itera sobre los elementos de la lista y suma que internamente tambien hace esta iteracion, nos deja como resultado el tamaño de la lista(m) * el tamaños de los vectores(n).

static BigInteger multiplicarListaValores(deque<BigInteger> bigg);

Su complejidad es O(n^2) ya que primero esta la iteracion de la lista O(n) y despues la multiplicacion que tiene una complejidad O(n^2), y siendo esta la de mayor compejida lo convierte en O(n^2)

string toString();

Su complejidad es O(n) ya que el ciclo while itera el total de elementos del vector.

//ANALIZADORAS

int tamano();

Su complejidad es O(1), ya que no es necesario recorrer todo el vector para conocer el tamaño gracias a la funcion size(), la cual ya conoce internamente su tamaño.

void signo();

Su complejidad es O(1) ya que no lleva ningun ciclo, todas las operaciones que hace las hace 1 vez y no necesita de recorrer nada.

//SOBREGARGA DE OPERADORES:

BigInteger operator+(BigInteger& elem);

Su complejidad es O(n) ya que el ciclo for itera la cantidad de elementos del vector BigInteger, ademas las otras operaciones se realizan o de forma constante o de n pero al no ser anidadas estas no aumentan su complejidad.

BigInteger operator-(BigInteger& elem);

Su complejidad es O(n) ya que Se realiza un bucle para iterar sobre los elementos de los vectores valor y elem.valor . Para cada elemento, se realizan operaciones básicas como restas y comparaciones. Estas operaciones tienen una complejidad constante, O(1), y se realizan para cada elemento del vector, lo que resulta en una complejidad de O(n).

BigInteger operator*(BigInteger& elem);

Su complejidad es O(n^2) ya que cuenta con un operaciones en ciclos anidados sobre los elementos de los vectores.

BigInteger operator/(BigInteger& elem);

Su complejidad es O(n) ya que esta es dominada por el bucle while a, con una complejidad lineal en función del tamaño de los vectores.

BigInteger operator%(BigInteger& elem);

Su complejidad es O(n) ya que la depende del tamaño de los vectores.

bool operator==(BigInteger& elem);

la complejidad del operador == sobrecargado está dominada por la comparación elemento por elemento de los vectores valor, lo cual tiene una complejidad lineal en función del tamaño de los vectores. Por lo tanto, la complejidad total del operador == es O(n).

bool operator<(BigInteger& elem);

comparación elemento por elemento de los vectores valor, lo cual tiene una complejidad lineal en función del tamaño de los vectores. Por lo tanto, la complejidad total del operador < es O(n).

bool operator<=(BigInteger& elem);

comparación elemento por elemento de los vectores valor, lo cual tiene una complejidad lineal en función del tamaño de los vectores. Por lo tanto, la complejidad total del operador <= es O(n).

bool operator>(BigInteger& elem);

comparación elemento por elemento de los vectores valor, lo cual tiene una complejidad lineal en función del tamaño de los vectores. Por lo tanto, la complejidad total del operador > es O(n)

bool operator>=(BigInteger& elem);

comparación elemento por elemento de los vectores valor, lo cual tiene una complejidad lineal en función del tamaño de los vectores. Por lo tanto, la complejidad total del operador >= es O(n).

bool operator!=(BigInteger& elem);

Su complejidad es O(n) ya que hace la misma operación del == pero devuelve el bool como false.