BigInteger(const string &);

Complejidad en el peor de los casos: O(n), n es el tamaño de la cadena s. Ya que se itera sobre cada carácter de la cadena una vez.

BigInteger(BigInteger &big);

Complejidad en el peor de los casos: O(n), n es el tamaño del vector num.big. Ya que la operación que copia el vector, implica iterar sobre cada elemento del vector big y copiarlo en el nuevo vector num.big

void add(BigInteger &);

Complejidad en el peor de los casos: O(n), donde n es el tamaño máximo entre n.size y big.size(). Ya que el ciclo for itera tam veces.

void product(BigInteger &);

Complejidad en el peor de los casos: O(n^2), donde n es el tamaño de "big". Ya que contiene un ciclo for anidado que itera sobre "big" y sobre "a".

void substract(BigInteger &);

complejidad en el peor de los casos: O(n^2), ya que itera sobre el vector "big" y posteriormente elimina los 0 que se encuentran en la primera posición.

void quotient(BigInteger &);

complejidad en el peor de los casos: O(n^3) ya que posee un ciclo while y dentro de este se utiliza la operación substract que posee una complejidad de O(n^2)

void remainder(BigInteger &);

complejidad en el peor de los casos: O(n^3) ya que posee un ciclo while y dentro de este se utiliza la operación substract que posee una complejidad de O(n^2)

void pow(BigInteger &);

complejidad en el peor de los casos: O(n^1/2) ya que posee un ciclo while y dentro de este se va dividiendo el tamaño de "a".

string toString ();

complejidad en el peor de los casos: O(n) ya que recorre el BigInteger y lo guarda en una cadena digito a digito.

int sumarListaValores(vector <BigInteger> &);

complejidad en el peor de los casos: O(n^2) ya que recorre una lista de bigIntegers y aplica el operador + que tiene una complejidad O(n)

int MultiplicarListaValores(vector <BigInteger> &);

complejidad en el peor de los casos: O(n^3) ya que recorre una lista de bigIntegers y aplica el operador * que tiene una complejidad O(n^2)

int operator[](int);

esta operación tiene una complejidad de O(1);

BigInteger operator +(BigInteger &a);

Complejidad en el peor de los casos: O(n), donde n es el tamaño máximo entre n.size y big.size(). Ya que el ciclo for itera tam veces.

BigInteger operator - (BigInteger &a);

complejidad en el peor de los casos: O(n^2), ya que itera sobre el vector "big" y posteriormente elimina los 0 que se encuentran en la primera posición.

BigInteger operator * (BigInteger &a);

Complejidad en el peor de los casos: O(n^2), donde n es el tamaño de "big". Ya que contiene un ciclo for anidado que itera sobre "big" y sobre "a".

BigInteger operator / (BigInteger &a);

complejidad en el peor de los casos: O(n^3) ya que posee un ciclo while y dentro de este se utiliza la operación substract que posee una complejidad de O(n^2)

BigInteger operator % (BigInteger &a);

complejidad en el peor de los casos: O(n^3) ya que posee un ciclo while y dentro de este se utiliza la operación substract que posee una complejidad de O(n^2)

bool operator == (BigInteger &a);

Complejidad en el peor de los casos: O(n), ya que posee un ciclo que recorre el bigInteger para compararlos con los de l objeto actual

bool operator < (BigInteger &a);

Complejidad en el peor de los casos: O(n), ya que posee un ciclo que recorre el bigInteger para compararlos con los de l objeto actual

bool operator <= (BigInteger &a);

Complejidad en el peor de los casos: O(n), ya que posee un ciclo que recorre el bigInteger para compararlos con los de l objeto actual.