```
/*
Función BigInteger
Entrada: ...
Salida: ...
Descripción: Esta funcion crea un big integer el cual como no recibe nada lo crea por defecto con un 0.

*/

BigInteger::BigInteger(){
    big.push_back(0);
}
```

La complejidad del push_back() de los vectores es constante amortizado, en teminos de o grande seria O(1)

```
/*
Función BigInteger
Entrada: Un string.
Salida: ...
Descripción: Esta función crea un BigInteger que tiene como numeros internos cada elemento del string ingresado.

*/

BigInteger::BigInteger(string cad){
    if(cad[0] == '-'){
        esPositivo = false;
    }
    else{
        esPositivo = true;
    }

    for(int i = cad.size() - 1; i >= 0; i--){
        if(esPositivo || i != 0){
            big.push_back(cad[i] - '0');
        }
    }
}
```

La complejidad es de O(n) siendo n la cantidad de caracteres que tiene la cadena, la complejidad es O(n) porque debe recorrer toda la cadena.

```
/*
Función BigInteger
Entrada: Un objeto BigInteger.
Salida: ...
Descripción: Esta función toma como parametro una costante BigInteger para poder copiar el parametro ingresado a un nuevo BigInteger.

*/

BigInteger::BigInteger(const BigInteger& big){ //el const la verdad lo puse porque al compilar la consola me marcaba error y me decia alg
    this->esPositivo = big.esPositivo;
    for(int i = 0; i < big.big.size(); i++){
        this->big.push_back(big.big[i]);
    }
}
```

La complejidad es O(n) siendo n el tamaño del vector, la complejidad es n porque debe recorrer todo el vector.

```
void BigInteger::add(BigInteger& big){
    int i = 0;
    int sum;
    int sobra = 0;

    if(this->esPositivo == big.esPositivo){
        while(i < this->big.size() || i < big.big.size() || sobra){
            if(i >= this->big.size()){
                this->big.push_back(0);
            }
            sum = this->big[i] + sobra;
            if (i < big.big.size()) {
                sum += big.big[i];
            }
            this->big[i] = sum % 10;
            sobra = sum / 10;
            i++;
        }
```

```
    }
    else if(this->esPositivo == true && big.esPositivo == false){
        big.esPositivo = true;
        if(*this == big){
            this->big.clear();
            this->big.push_back(0);
        }
        else{
            substract(big);
            big.esPositivo = false;
        }
    }
    else if(this->esPositivo == false && big.esPositivo == true){
        big.esPositivo = false;
        if(*this == big){
            this->big.clear();
            this->big.push_back(0);
            this->esPositivo = true;
        }
        else{
            substract(big);
            big.esPositivo = true;
        }
    }
}
```

Siendo n el tamaño del vector

```
/*
Función operator+
Entrada: Un objeto BigInteger.
Salida: Un BigInteger.
Descripción: Esta función retorna suma de dos BigIntegers(Sobrecarga del operador +).

*/

BigInteger BigInteger::operator+(BigInteger& big){
    BigInteger ans(*this);
    ans.add(big);
    return ans;
}
```

Siendo n la cantidad de elementos del vector

```
/*
Función toString
Entrada: ...
Salida: Un string.
Descripción: Esta función retorna un string que corresponde a los elementos del objeto BigInteger al cual se le realiza la funcion.

*/

string BigInteger::toString(){
    string res = "";
    if(!esPositivo){
        res += '-';
    }

    for(int i = big.size() - 1; i >= 0; i--){
        char digit = '0' + big[i];
        res += digit;
    }
```

Siendo n la cantidad de elementos del vector

```
    return res;
}
```

O(n) = n

```
void BigInteger::product(BigInteger& big){

    if(esPositivo == false && big.esPositivo == false){
        esPositivo = true;
    }
    else if(esPositivo == true && big.esPositivo == false){
        esPositivo = false;
    }
    else if(esPositivo == false && big.esPositivo == true){
        esPositivo = false;
    }
    else{
        esPositivo = true;
    }

    int sobra = 0;
    int suma = 0;
    BigInteger temp;

    for(int i = 0; i < big.big.size() || i < this->big.size(); i++){
        temp.big.push_back(0);
    }

    for(int i = 0; i < big.big.size(); i++){
        for(int s = 0; s < this->big.size(); s++){
            suma += this->big[s] * big.big[i];
            sobra = suma / 10;
            suma = suma % 10;
            temp.big[s + i] += suma;
            suma = sobra;
        }
        temp.big.push_back(0);

        if(sobra != 0){
            temp.big[i + this->big.size()] += sobra;
            sobra = 0;
            suma = 0;
        }
    }
    for(int i = 0; i < temp.big.size(); i++){
        if(temp.big[i] > 9){
            sobra = temp.big[i] / 10;
            suma = temp.big[i] % 10;
            temp.big[i] = suma;
            temp.big[i + 1] += sobra;
        }
    }
    int i = 0;
    int s = temp.big.size() - 1;
    while(temp.big[s] == 0){
        temp.big.pop_back();
        s = temp.big.size() - 1;
    }
    this->big = temp.big;
```

Siendo n la cantidad de elementos del vector

$T(n) = 6n^2 + 6n + 8$

peor caso

$O(n^2)$

```
/*
Función operator*
Entrada: Un objeto BigInteger.
Salida: Un objeto BigInteger.
Descripción: Esta función retorna un BigInteger el cual es el resultado de multiplicar el BigInteger
al cual se le aplica la funcion por el BigInteger de los parametros(Sobrecarga del operador *).

*/

BigInteger BigInteger::operator*(BigInteger& big){
    BigInteger ans(*this);
    ans.product(big);
    return ans;
}
```

n^2 $T(n) = n^2 + 2 \leftarrow O(n^2)$

Siendo n la cantidad de elementos del vector

```
/*
Función pow
Entrada: Un objeto BigInteger.
Salida: ...
Descripción: Esta función multiplica el BigInteger al cual se le aplica la funcion por si mismo n cantidad de veces que corresponde a un numero
ingresado en los parametros.

*/

void BigInteger::pow(int num){
    BigInteger temp(*this);
    for(int i = 0; i < num - 1; i++){
        product(temp);
    }
}
```

$T(n) = mn^2 + 1$

$O(m \cdot n^2)$

Siendo n la cantidad de elementos de vector y m la cantidad de veces que itera el ciclo o por decirlo de otra manera la cantidad de veces que se multiplica el mismo.

m

$n^2 \cdot m$

```
/*
Función operator==
Entrada: Un objeto BigInteger.
Salida: Un valor booleano.
Descripción: Esta función retorna un valor booleano que dice si dos BigIntegers son iguales o no lo son.

*/

bool BigInteger::operator==(BigInteger& big){
    bool ans = true;
    if(this->esPositivo == big.esPositivo){
        if(this->big.size() != big.big.size()){
            ans = false;
        }
        else{
            for(int i = 0; i < this->big.size() && ans; i++){
                if(this->big[i] != big.big[i]){
                ans = false;
                }
            }
        }
    }
    else if(this->esPositivo == false && big.esPositivo == true){
        ans = false;
    }
    else if(this->esPositivo == true && big.esPositivo == false){
        ans = false;
    }

    return ans;
}
```

$T(n) = 3n + 8 \leftarrow O(n)$

Siendo n la cantidad de elementos del vector

```
bool BigInteger::operator<=(BigInteger& big){
    bool ans = true;
    bool flag = true;
    if(this->esPositivo == true && big.esPositivo == true){
        if(this->big.size() < big.big.size()){
            ans = true;
        }
        else if(this->big.size() > big.big.size()){
            ans = false;
        }
        else{
            for(int i = this->big.size() - 1; i >= 0 && ans && flag; i--){
                if(this->big[i] > big.big[i]){
                    ans = false;
```

```
            }
            else if(this->big[i] < big.big[i]){
                ans = true;
                flag = false;
            }
            else if(this->big[i] == big.big[i]){
                ans = true;
            }
        }
    }
}
else if(this->esPositivo == false && big.esPositivo == true){
    ans = true;
}
else if(this->esPositivo == true && big.esPositivo == false){
    ans = false;
}
else{
    if(this->big.size() < big.big.size()){
        ans = false;
    }
    else if(this->big.size() > big.big.size()){
        ans = true;
    }
    else{
        for(int i = this->big.size(); i >= 0 && ans && flag; i--){
            if(this->big[i] > big.big[i]){
                ans = true;
                flag = false;
            }
            else if(this->big[i] < big.big[i]){
                ans = false;
                flag = false;
            }
            else if(this->big[i] == big.big[i]){
                ans = true;
            }
        }
    }
}
```

Siendo n la cantidad de elementos del vector
Hay que aclarar que todas lals complejidades presentadas son en los peores casos, donde sea necesario siempre recorrer todo el vector.

```
bool BigInteger::operator<(BigInteger& big){
    bool ans = true;
    bool flag = true;
    if(this->esPositivo == true && big.esPositivo == true){
        if(this->big.size() < big.big.size()){
            ans = true;
        }
        else if(this->big.size() > big.big.size()){
            ans = false;
        }
        else if(*this==big){
            ans = false;
        }
        else{
            for(int i = this->big.size() - 1; i >= 0 && ans && flag; i--){
                if(this->big[i] > big.big[i]){
                    ans = false;
                }
                if(this->big[i] < big.big[i]){
                    ans = true;
                    flag = false;
                }
            }
        }
    }
    else if(this->esPositivo == true && big.esPositivo == false){
        ans = false;
    }
```

```
    else if(this->esPositivo == false && big.esPositivo == true){
        ans = true;
    }
    else{
        if(this->big.size() < big.big.size()){
            ans = false;
        }
        else if(this->big.size() > big.big.size()){
            ans = true;
        }
        else if(*this==big){
            ans = false;
        }
        else{
            for(int i = this->big.size() - 1; i >= 0 && ans && flag; i--){
                if(this->big[i] > big.big[i]){
                    ans = true;
                    flag = false;
                }
                if(this->big[i] < big.big[i]){
                    ans = false;
                    flag = false;
                }
            }
        }
```

Siendo n la cantidad de elementos del vector

$T(n) = 15n - 6 \leftarrow O(n)$

```
/*
Función remainder
Entrada: Un objeto BigInteger.
Salida: ...
Descripción: Esta función se encarga de sacar el residuo que se genera de la division de dos BigInteger.

*/

void BigInteger::remainder(BigInteger& big){
    int cnt = 0;
    int bandera = 0;
    vector<int> tempo;
    this->esPositivo = true;
    big.esPositivo = true;

    BigInteger temp(*this);
    BigInteger mult;
    BigInteger dos("2");
    BigInteger ten("10");
    BigInteger cero("0");

    if(big == dos){
        bandera = 1;
        if(this->big[0] % 2 == 0){
            this->big.clear();
            this->big.push_back(0);
        }
        else{
            this->big.clear();
            this->big.push_back(1);
        }
    }
    else if(big == ten){
        bandera = 1;
        int save = this->big[0];
        this->big.clear();
        this->big.push_back(save);
    }
    if(bandera == 0){
        temp.quotient(big);
        mult = temp * big;
        substract(mult);
    }
}
```

$T(n) = n^3 \log n + n^2 + 3n + 22$

$O(n^3 \log n)$

n es la cantidad de elementos del vector

```
/*
Función operator%
Entrada: Un objeto BigInteger.
```

```
 Salida: Un objeto BigInteger.
 Descripción: Esta función se encarga de retonar el residuo que se genera de la division de dos BigInteger(sobrecarga del operador %).

 */

BigInteger BigInteger::operator%(BigInteger& big){
    BigInteger temp(*this);
    temp.remainder(big);
    return temp;
}
```

Siendo n la cantidad de elementos del vector

$n^3 \log n$

$T(n) = n^3 \log n + n + 1$

$O(n^3 \log n)$

```
/*
Función multiplicarListaValores
Entrada: Una lista que contiene objetos BigInteger.
Salida: Un objeto BigInteger.
Descripción: Esta función se encarga de sacar la multiplicacion de todos los objetos BigInteger de una lista y los retorna en un solo objeto
BigInteger.

*/

BigInteger BigInteger::multiplicarListaValores(list<BigInteger>& lista){
    BigInteger temp("1");

    for(std::list<BigInteger>::iterator it = lista.begin(); it != lista.end(); ++it){
        temp = temp * (*it);
    }
    return temp;
}
```

Siendo m la cantidad de elementos de la lista

siendo n la cantidad de elementos del vector

$n^2 \cdot m$

$T(n) = n^2 \cdot m + m + 1$

$O(n^2 m)$

```
/*
Función sumarListaValores
Entrada: Una lista que contiene objetos BigInteger..
Salida: Un objeto BigInteger.
Descripción: Esta función se encarga de sacar la suma de todos los objetos BigInteger de una lista y los retorna en un solo objeto BigInteger.

*/

BigInteger BigInteger::sumarListaValores(list<BigInteger>& lista){
    BigInteger temp("0");

    for(std::list<BigInteger>::iterator it = lista.begin(); it != lista.end(); ++it){
        temp = temp + (*it);
    }
    return temp;
}
```

Siendo m la cantidad de elementos de la lista

siendo n la cantidad de elementos del vector

$m \cdot n^3$

$T(n) = mn^3 + m + 2$

$O(n) = mn^3$

```
void BigInteger::quotient(BigInteger& big){
    bool temporal;
    BigInteger cnt("0");
    int i;
    int bandera;
    char tm;
    string res = "";
    else{
        BigInteger dos("2");
        BigInteger uno("1");
        BigInteger ten("10");
        BigInteger cero("0");
        if(big == dos){
            divisionPorDos();
        }
```

```
        else{
            bandera = 0;
            BigInteger tempoSuma;
            if((big + big) <= temp){
                tempoSuma = big + big;
                cnt = cnt + dos;
            }
            else{
                bandera = 1;
                cnt = cnt + uno;
            }
            while(tempoSuma <= temp && bandera == 0){
                if((tempoSuma + tempoSuma) <= temp){
                    tempoSuma = tempoSuma + tempoSuma;
                    cnt = cnt + cnt;
                }
                else{
                    if((tempoSuma + big) <= temp){
                        tempoSuma = tempoSuma + big;
                        cnt = cnt + uno;
                    }
                    else{
                        bandera = 1;
                    }
                }
            }
            *this = cnt;
            this->esPositivo = temporal;
        }
    }
}
```

$T(n) = 10n^3 + 6n + 2 \in O(n^3)$

Siendo n la cantidad de elementos del vector

```
/*
Función operator/
Entrada: Un objeto BigInteger.
Salida: Un objeto BigInteger
Descripción: Esta función se encarga de realizar la division entera de dos BigInteger y retorna el resultado del mismo(Sobrecarga del operador /).

*/

BigInteger BigInteger::operator/(BigInteger& big){
    BigInteger temp;
    temp = *this;
    temp.quotient(big);
    return temp;
}
```

$T(n) = n^3 + 3$

$O(n^3)$

Siendo n la cantidad de elementos del vector

```
/*
Función divisionPorDos
Entrada: ...
Salida: ...
Descripción: Esta función se realiza cuando tratas de dividir un objeto BigInteger por dos y te realiza la divison.
```

```
*/


void BigInteger::divisionPorDos(){
    if(!big.empty()){
        int acarreo = 0;
        for(int i = big.size() - 1; i >= 0; --i){
            int digito = big[i];
            big[i] = (digito + acarreo) / 2;
            acarreo = (digito % 2) * 10;
        }
    }
    while (!big.empty() && big.back() == 0) {
        big.pop_back();
    }
}
```

Siendo n la cantidad de elementos del vector

$T(n) = 3n + 5$

$O(n)$

```
void BigInteger::substract(BigInteger& big){
    bool flag = true;
    bool flag2 = false;
    if(this->esPositivo == true && big.esPositivo == true){
        BigInteger temp(big);
        if(*this < big){
            this->esPositivo = false;
        }
        if(*this == big){
            this->big.clear();
            this->big.push_back(0);
            flag2 = true;
        }
        else{
            if(this->esPositivo){
                int i = 0;
                int s;
                while(i < this->big.size() || i < big.big.size()){
                    if(i < big.big.size()){
                        if(this->big[i] >= big.big[i]){
                            this->big[i] = this->big[i] - big.big[i];
                        }
                        else{
                            flag = true;
                            for(s = i + 1; s < this->big.size() && flag; s++){
                                if(this->big[s] > 0){
                                    this->big[s] = this->big[s] - 1;
                                    flag = false;
                                }
                                else if(this->big[s] == 0){
                                    this->big[s] = 9;
                                }
                            }
                            this->big[i] += 10 - big.big[i];
                        }
                    }
                    i++;
                }
            }
            else{
                int i = 0;
                while(i < this->big.size() || i < temp.big.size()){
                    if(i < this->big.size()){
                        if(temp.big[i] >= this->big[i]){
                            temp.big[i] = temp.big[i] - this->big[i];
                        }
                        else{
```

```
                        flag = true;
                        for(int s = i + 1; s < temp.big.size() && flag; s++){
                            if(temp.big[s] > 0){
                                temp.big[s] = temp.big[s] - 1;
                                flag = false;
                            }
                            else if(temp.big[s] == 0){
                                temp.big[s] = 9;
                            }
                        }
                        temp.big[i] += 10 - this->big[i];
                    }
                }
                i++;
            }
        }
    }
    if(!this->esPositivo){
        this->big = temp.big;
    }

    if(!flag2){
        int s = this->big.size() - 1;
        while(this->big[s] == 0){
            this->big.pop_back();
            s = this->big.size() - 1;
        }
    }
```

$T(n) = 4n^2 + 11 \leftarrow O(n^2)$

Siendo n la cantidad de elementos del vector

```
/*
Función operator-
Entrada: Un objeto BigInteger.
Salida: Un objeto BigInteger.
Descripción: Esta función retorna un objeto BigInteger que es el resultado de la resta de dos objetos BigInteger(Sobrecarga del operador -).

*/


BigInteger BigInteger::operator-(BigInteger& big){
    BigInteger ans(*this);
    ans.substract(big);
    return ans;
}
```

$T(n) = n^2 + n + 1$

$O(n^2)$

Siendo n la cantidad de elementos del vector