**Estructura de datos proyecto(reporte de complejidad)**

**Rui Yu Lei Wu**

**Código: 8978227**

La implementación elegida para el TAD BigInteger fue de vectores de STL.

**Constructores:**

En este caso el TAD BigInteger cuenta con tres constructores.

1. Constructor por defecto: Este constructor inicializa un vector vacío denominado vec. La complejidad de esta operación es O(1), debido a que inicializar un vector tiene coste constante.
2. Constructor String: Este constructor inicializa un vector el cual contiene en la posición 0 el número 1 o 0 correspondiente al signo del número, seguido de eso cada posición corresponde a un número **“n”** , en el cual cada número representa a un número del string. En este caso el vector alberga el numero dado la vuelta. La complejidad de esta operación es O(n) donde n es el número de elementos en el string.
3. Constructor copia: Este constructor copia los elementos de un BigInteger y los alberga en otro nuevo, esto sin modificar el BigInteger original. La complejidad de esta operación es O(n) donde n representa el número de elementos del BigInteger original, esto debido a que lo único que hace este constructor es copiar los elementos de un vector y copiarlos a otro.

**Operaciones:**

El TAD BigInteger cuenta con seis operaciones principales.

1. Operación add: Esta operación es la encargada de sumar dos números de n dígitos, esta operación contiene un ciclo for que va desde i hasta el final del BigInteger más pequeño, seguido de esto si el tamaño de ambos BigIntegers es diferente entra en otro ciclo que itera hasta que el residuo sea igual a cero y i igual al tamaño del BigInteger más grande. En este caso la complejidad de esta implementación es O(n) donde n es el tamaño del BigInteger más grande.
2. Operación substract: Esta operación es la encargada de restar dos números de n dígitos, esta operación tiene un concepto muy parecido a la operación add, un ciclo que va desde i hasta el tamaño del BigInteger con menos dígitos, y si el tamaño de ambos BigInteger es diferente entra en otro ciclo el cual terminar cuando i es igual al tamaño del BigInteger más grande. La complejidad de esta implementación al igual que la operación add es O(n) donde n es el tamaño del BigInteger más grande.
3. Operación Product: Esta operación es la encargada de multiplicar dos números (BigInteger “a”, “n”) de n dígitos, la operación “product” esta conformada por dos ciclos anidados los cuales recorren los dos números que se quieren multiplicar, dentro de estos dos ciclos existe una función encargada de insertar ceros a una instancia BigInteger la cual va ser usada para albergar de cada secuencia de la multiplicación, para después sumar este BigInteger denominado “parcial” a otro que albergara el resultado. La complejidad de esta operación debido a su doble ciclo anidado es de O(n\*m) donde n es el numero de elementos en el BigInteger “a” y m el numero de elementos del BigInteger “n”. Debido a esto se puede concluir que esta operación es O(n2).
4. Operación División(remainder,quotient): En este caso existe un operación privada que realiza ambas operaciones simultáneamente, se trata de una división con restas sucesivas en la cual se resta n veces el divisor al dividendo hasta que el dividendo sea menor al divisor, esto si el dividendo es mayor al divisor. La complejidad de esta operación es O(n2).
5. Operación pow: La operación pow es la encargada de elevar un BigInteger “a” a un numero entero, su funcionamiento consta de un for que itera desde 0 hasta n/2, de esta forma se trata de reducir la cantidad de multiplicaciones que se pretenden desde un inicio, dentro de este ciclo se invoca la operación product con coste n2, esto para realizar las multiplicaciones correspondientes. La complejidad de esta operación es O(\*n2).

**Operaciones adicionales**

Operadores de comparación:

1. Operador ==: Este operador es el encargado de comparar si dos instancias BigInteger son iguales. La complejidad de esta operación es O(n) donde n es el numero de elementos de uno de los BigIntegers, esto en el caso que sean iguales.
2. Operador <:

Para el caso a<b:

Este operador es el encargado de indicar si se cumple o no esta afirmación.

La complejidad de esta operación es O(n) en el peor caso, que es cuando a

y b son iguales, para este caso n representa el tamaño de “a”, debido a que ambos tamaños son iguales.

1. Operador <=: Este operador se encarga de determinar si “b” es mayor o igual a “a”. La complejidad de esta operación es O(n), donde n es el numero de elementos de uno de los BigIntegers, esto debido a que el peor caso es que sean iguales.
2. Operación tostring: Esta operación se encarga de convertir una instancia BigInteger a un string. La complejidad de esta operación es O(n) donde n es el numero de elementos en el BigInteger. Esto debido a que se recorre el BigInteger hasta llegar a la posición 0 del número.
3. SumarListaValores: Esta operación se encarga de sumar todos los valores de una lista de BigInteger. La complejidad de esta operación es O(n\*m) donde n es el numero de BigIntegers en la lista y m el numero de elementos del mayor BigInteger en cada interación. Esto debido a que se recorre la lista mediante iteradores, hasta llegar al final de la misma, y cada iteración se realiza una suma que tiene costo O(n).
4. MultiplicarListaValores: Esta operación se encarga de multiplicar todos los valors de una lista de BigInteger. La complejidad de esta operación es O(n \*m2) donde n es el numero de BigIntegers en la lista y m2 se refiere a cada multiplicación realizada. Esto debido a que se recorre la lista hasta llegar al final, y cada iteración se realiza una multiplicación con coste O(n2)