**Addition function:**

The function takes two strings (s1,s2) then it declare a string called result and this string will return in the end. Then it declare a variable called carry from char data type to use it in the addition operation when there is a carry.

Then there is declaration for two variables called l1, l2 from int data type and make them equal the two strings size.

Then there is a condition if the first string size less than or equal to the second string

It will go into loop loops from first string size minus one till 0 there is declaration to variables I and j. they equal to strings size minus one. Each time, they decrement.

Inside the loop, there is a declaration to a variable called sum from int. it sums the two strings bit by bit plus the carry if exists. There is a solution we used to turn the string to integer just when we add which is minus ‘0’ from the string. Then we push back the sum modulus 10 to take a number less than 10 and we turn it to string type by adding ‘0’

After this there is a line that makes the carry equal to sum/10 (0 or 1). When the first loop finish, the function will go into a second for loop which add the rest of the remaining bits in the bigger string. There is a declaration for a variable i (the difference between l2 and l1 (strings size) minus 1) which the loop will start from till zero. Each time, i decrements. This loop will sum the rest of bits of the longer string to the carry if exists then push the sum%10 +’0’ (turn to string) back to result. Then there is line that make the carry equal to sum/10(0 or 1). Then after the loop finish, we will check if the carry equal to zero or no if no we will push it back to the result(carry+’0’).

Else

Everything will be the same except the first loop will start from the second string size minus 1. And the second loop will sum the rest of bits of the first string to the carry if exists.

Then there is a built in function that takes the result string and reverse its beginning and ending because when we push back, it pushes it reversely so we use this function to handle this case. Finally, we return the result.

The complexity of the function

* This function include two for loop if any condition true , but not nested loop each one of them takes O(n)
* Increments, decrements, conditions, assignments and declarations all of these lines take o(1).

-there is a built-in function: reverse which take the result and go over it and reverse its beginning and ending

**Time complexity:** O(N)

These are the loops:-

* The first one is:

for (int i = l1 - 1, j = l2 - 1; i >= 0; i--, j--) {

int sum = (s1[i] - '0') + (s2[j] - '0') + carry;

result.push\_back(sum % 10 + '0');

carry = sum / 10;

}

-This loop add strings bit by bit and push back to the result

* **Time complexity:** :#iterationXo(body)

**-** This loop takes complexity O(n); which is calculated by number of iterations “which is n” \* the body of the loop “which is O(1)”.

O(N)\*O(1)=**O(N).**

* The second loop is :

for (int i = l2 - l1 - 1; i >= 0; i--) {

int sum = (s2[i] - '0') + carry;

result.push\_back(sum % 10 + '0');

carry = sum / 10;

}

This loop will sum the rest of bits of the longer string to the carry if exists then push the sum%10 +’0’(turn to string)back to result.

* **Time complexity:** :#iterationXo(body)

**-** This loop takes complexity O(n); which is calculated by number of iterations “which is approximately n” \* the body of the loop “which is O(1)”.

O(N)\*O(1)=**O(N).**

* **The total complexity of the Addition function is O(n).**

The function’s code and analyses:

string biginteger::addition(string s1, string s2)

{

string result = {}; O(1)

char carry = 0; O(1)

int l1 = s1.length(); O(1)

int l2 = s2.length(); O(1)

if (s1.length() <= s2.length()) O(1)

{

for (int i = l1 - 1, j = l2 - 1; i >= 0; i--, j--) { **number of iteration “n” \* body of the loop ”O(1)” = O(n)**

int sum = (s1[i] - '0') + (s2[j] - '0') + carry; O(1)

result.push\_back(sum % 10 + '0'); O(1)

carry = sum / 10; O(1)

}

for (int i = l2 - l1 - 1; i >= 0; i--) { **number of iteration “n” \* body of the loop ”O(1)” = O(n)**

int sum = (s2[i] - '0') + carry; O(1)

result.push\_back(sum % 10 + '0'); O(1)

carry = sum / 10; O(1)

}

if (carry!=0) O(1) {

result.push\_back(carry + '0'); O(1)

}

}

else {

for (int i = l2 - 1, j = l1 - 1; i >= 0; i--, j--) { **number of iteration “n” \* body of the loop ”O(1)” = O(n)**

int sum = (s1[j] - '0') + (s2[i] - '0') + carry; O(1)

result.push\_back(sum % 10 + '0'); O(1)

carry = sum / 10; O(1)

}

for (int i = l1 - l2 - 1; i >= 0; i--) { **number of iteration “n” \* body of the loop ”O(1)” = O(n)**

int sum = (s1[i] - '0') + carry; O(1)

result.push\_back(sum % 10 + '0'); O(1)

carry = sum / 10; O(1)

}

if (carry!=0) O(1) {

result.push\_back(carry + '0'); O(1)

}

}

reverse(result.begin(), result.end()); O(N)

return result; O(1)

}

**Subtraction function:**

Description:

* First we created two strings to store the two big integer , after that we checked on the two strings to know which of them is larger by : first we checked on their length & if they have the same length we start to loop on the two strings & the string which have the digit larger than the digit on the other string we consider his string is the larger big integer , if the 2nd string is larger than the 1st we assign the mark to be 1 else we assign mark to be 2 .
* Second we check on the mark to know which of the 2 strings is the bigger after that , the first loop start from the last digit of the two strings & stop when reach to the length of the smaller string & start to subtract the two digit starting by the large string & the borrow if found , if the subtraction is less than 0 so it mean that we borrow so we increment sub by 10 & increment borrow by 1 then we push the result of the sub in S3 & subtract the borrow from the next integer, after the loop end we start to check on the remaining part of the large string & if borrow of the first loop we subtract it from remaining value & if sub is less than 0 update borrow to 1 & increment sub by 10 or update borrow to 0 if sub is grater than 0 & push the remaining value in S3

The complexity of the function

* This function include three for loop , but not nested loop each one of them take O(n)

These are the loop s

**The first one is:**

for (int i = 0; i < l1; i++) {

if (s2[i] > s1[i]) {

mark = 1;

break;

}

else if (s1[i]>s2[i]) {

mark = 2;

break;

}

}

}

if (l2>l1 || mark == 1) {

int j = l2 - 1;

for (int i = l1 - 1; i >= 0; i--) {

sub = ((s2[j] - '0') - (s1[i] - '0') - borrow);

if (sub < 0) {

sub += 10;

borrow = 1;

}

else {

borrow = 0;

}

j--;

if (i > 0)

s3.push\_back(sub + '0');

else if (sub != 0)

s3.push\_back(sub + '0');

}

* This loop is loop to find which string is the larger if the length of the two string is equal and this loop takes complexity O(n); which is calculated by number of iteration “which is n” \* the body of the loop “which is O(1)”.
* **The second loop is :**

for (int i = l1 - 1; i >= 0; i--) {

sub = ((s2[j] - '0') - (s1[i] - '0') - borrow);

if (sub < 0) {

sub += 10;

borrow = 1;

}

else {

borrow = 0;

}

j--;

if (i > 0)

s3.push\_back(sub + '0');

else if (sub != 0)

s3.push\_back(sub + '0');

}

* This loop iterate on the two strings to subtract each digit by digit & the borrow & this loop take complexity O(n) ); which is calculated by number of iteration “which is n” \* the body of the loop “which is O(1)”.
* **The third loop is :**

for (int i = l2-l1-1; i >= 0; i--) {

int sub = ((s2[i] - '0') - borrow);

if (sub < 0)

{

sub = sub + 10;

borrow = 1;

}

else

borrow = 0;

if (i > 0)

s3.push\_back(sub + '0');

else if (sub != 0)

s3.push\_back(sub + '0');

}

}

* This loop iterate to update the remaining part of the large string if the two strings are not equal in length & if the two strings are equal we return zero & also this loop take O(n) ); which is calculated by number of iteration “which is n” \* O(the body of the loop) “which is O(1)”.
* We also use the built in function reverse to reverse the string after the subtraction & this function take O(n) as it loops allover, the string & this function working by taking the beginning of the string & the ending of the string .
* Note : each state in the function we didn`t mention it take an O(1) “ declaration , assign , increment , decrement ”.
* **The total complexity of the subtraction function is O(n).**

**The code of the function:**

string biginteger::subtrication(string s1, string s2)

{

int l1 = s1.length(); **O(1)**

int l2 = s2.length(); **O(1)**

string s3; **O(1)**

int borrow = 0; **O(1)**

int sub; **O(1)**

int mark = 0; **O(1)**

if (l1 == l2) { **O(1)**

for (int i = 0; i < l1; i++) { **number of iteration “n” \* body of the loop ”O(1)” = O(n)**

if (s2[i] > s1[i]) { **O(1)**

mark = 1; **O(1)**

break; **O(1)**

}

else if (s1[i]>s2[i]) { **O(1)**

mark = 2; **O(1)**

break; **O(1)**

}

}

}

if (l2>l1 || mark == 1) { **O(1)**

int j = l2 - 1; **O(1)**

for (int i = l1 - 1; i >= 0; i--) **number of iteration “n” \* body of the loop ”O(1)”=O(n)**

{

sub = ((s2[j] - '0') - (s1[i] - '0') - borrow); **O(1)**

if (sub < 0) { **O(1)**

sub += 10; **O(1)**

borrow = 1; **O(1)**

}

else { **O(1)**

borrow = 0; **O(1)**

}

j--; **O(1)**

if (i > 0) **O(1)**

s3.push\_back(sub + '0'); **O(1)**

else if (sub != 0) **O(1)**

s3.push\_back(sub + '0'); **O(1)**

}

for (int i = l2 - l1 - 1; i >= 0; i--) **number of iteration “n” \* body of the loop ”O(1)”=O(n)**

{

int sub = ((s2[i] - '0') - borrow); **O(1)**

if (sub < 0) **O(1)**

{

sub = sub + 10; **O(1)**

borrow = 1; **O(1)**

}

else

borrow = 0; **O(1)**

if (i > 0) **O(1)**

s3.push\_back(sub + '0'); **O(1)**

else if (sub != 0) **O(1)**

s3.push\_back(sub + '0'); **O(1)**

}

}

else {

int j = l1 - 1; **O(1)**

for (int i = l2 - 1; i >= 0; i--) **number of iteration “n” \* body of the loop ”O(1)” = O(n)**

{

sub = ((s1[j] - '0') - (s2[i] - '0') - borrow); **O(1)**

if (sub < 0) { **O(1)**

sub += 10; **O(1)**

borrow = 1; **O(1)**

}

else

borrow = 0; **O(1)**

j--; **O(1)**

if (i > 0) **O(1)**

s3.push\_back(sub + '0'); **O(1)**

else if (sub != 0) **O(1)**

s3.push\_back(sub + '0'); **O(1)**

}

for (int i = l1 - l2 - 1; i >= 0; i--) **number of iteration “n” \* body of the loop ”O(1)” = O(n)**

{

int sub = ((s1[i] - '0') - borrow); **O(1)**

if (sub < 0) **O(1)**

{

sub = sub + 10; **O(1)**

borrow = 1; **O(1)**

}

else

borrow = 0; **O(1)**

if (i>0) **O(1)**

s3.push\_back(sub + '0'); **O(1)**

else if (sub != 0) **O(1)**

s3.push\_back(sub + '0'); **O(1)**

}

}

reverse(s3.begin(), s3.end()); **O(n)**

return s3; **O(1)**

}

**Multiplication function:**

To get the result of multiplication of two big integers we use a efficient algorithm Divide and conquer (Karatsuba algorithm)

It should take two big integers as input and return a third big integer that is the product of the two. Observe that if a and b are N-digit integers, the product will have at most 2N digits.{\displaystyle z\_{1}=x\_{1}y\_{0}+x\_{0}y\_{1},} {\displaystyle z\_{0}=x\_{0}y\_{0}.}

**D&C Steps:**

* **Divide:**
* split x into two halves: a & b
* split y into c & d.
* Calculate a+c and b+d
* **Conquer:** Recursively calculate 3 multiplications: ac, bd and (a+c)(b+d).
* **Combine:**
* subtract the first two multiplications from the third one.
* pad the second multiplication (bd) with N zeros and the subtraction result with N/2 zeros.
* add them together with the first multiplication (ac).

Implement function with name multiplication and it takes two parameters with data type string and returns result as string .

At the first of the function we make the size of the two integers equal to each other by putting zeroes at he left and then put our base cases that if the integer contains no integer return zero or if the integer contains one integer at this case we use normal multiplication one digit by one digit else we get the half of the strings size and divide them into two halves and put them in variables a,b,c,d as following

|  |  |  |
| --- | --- | --- |
|  | N/2 | N/2 |
| **S1 =** | **a** | **b** |
|  | N/2 | N/2 |
| **S2 =** | **c** | **d** |

And then we start to call the function multiplication recessively to get the

eq1 = multiplication(b, d)

eq2 = multiplication(addition(a, b), addition(c, d))

eq3 = multiplication(a, c)

and compute eq4 = subtraction (eq2, addition(eq1, eq3))

After finishing filling the equations we start to put zeroes in the eq4 equal to number of the half integer if even but if the number of digits is odd we put zeroes equal to half the size -1 and we put zeroes to eq3 equal to the number of the number of digits of the integer if even but if the number of digits is odd we put zeroes equal to the number of the digits – 1.

Then we compute the result =(eq1,addition (eq4,eq3)) after that we use a build in function “result.erase(0, result.find\_first\_not\_of('0'))” this function erase

Zeroes from the left side until it finds any number else at the least, we return the correct result

The complexity of un recursive code of the function

* This function include two for loop, but not nested loop each one of them takes O(n)
* Increments, decrements, conditions, and declarations all of these lines take O(1).

-there is a built-in function: substr Complexity equal O(1)

result.erase(0, result.find\_first\_not\_of('0'))

Complexity equal up to linear in size of string - pos times the number of characters to match (worst case)(O(n))

These are the loops:-

Frist loop:

for (int i = l2 - l1; i > 0; i--)

{

s1 = '0' + s1;

}

Complexity O(n)

Second loop:

for (int i = 0; i < n - n % 2; i++)

{

if (i < mid - n % 2)

{

eq4.push\_back('0');

}

eq3.push\_back('0');

}colexity O(n)

The complexity of recursive code of the function

we have three recursive codes :

eq1 = multiplication(b, d)

eq2 = multiplication(addition(a, b), addition(c, d))

eq3 = multiplication(a, c)

total comlexity

T(n)=3T(n/2)+n

By using master method

complexity

 (nlog23){\displaystyle T(n)=\Theta (n^{\log \_{2}3})\,\!}