**BOOTH’S ALGORITHM**

**DOCUMENTATION**

**Booth's multiplication algorithm** is a [multiplication algorithm](https://en.wikipedia.org/wiki/Multiplication_algorithm" \o "Multiplication algorithm) that multiplies two signed [binary](https://en.wikipedia.org/wiki/Base_2" \o "Base 2) numbers in [two's complement notation](https://en.wikipedia.org/wiki/Two's_complement" \o "Two's complement) efficiently. I have taken no. Of bits(**nof**), and two numbers(n1,n2) to be multiplied in decimal format. Since a string is immutable therefore at times I have used string buffer. I have taken the binary form of the two numbers taken as input in **s1,s2**(taking two’s complement in case of negative no.). As in case of normal multiplication we need to repeatedly add 0 but in case of booth’s algorithm we do this by shifting and thus require less number of additions and subtractions.

Some important declared variables before we move on to the main algo:

**check:** contains the no of bits.

**multiplier:** binary form of the number being multiplied by.

**op[]:** A one-d array where the first index is the last bit of the multiplier while the second index of the array is basically one bit register and it takes the value of the the first index during a right shift(or can be said as the the second last bit of the multiplier). This array acts as double check.

**multiplicand:** two’s complement of the number being multiplied.

**ac:** A register initialized with “0”s depending upon no. Of bits.

Now beginning with our main loop which performs the multiplication.

The while loop keeps track of the “check” till it is not zero.

At each iteration we keep updating the String “operate” which has the two bits of the “op” array. “Operate” has the following values possible:00,01,10,11.

These values indicate the process to be performed:

**“00”/”11”:** We perform an arithmetic right shift to “ac” which basically replicates the sign bit(also the msb) as it is after performing a right shift operation. In the next step there is right shift of the multiplier where the last bit of “ac” becomes the first bit of the multiplier. The op array is changed accordingly. Finally value of “check” is reduced by 1.

**“01”:** Update the value of “ac” register by adding “s1”(carrying out a binary addition) excluding any overflow. This is followed by arithmetic right shift as performed in case of “00”/”11”.

**“10”:** Update the value of “ac” register by adding “multiplicand”(which is the two’s complement of s1) excluding any overflow. This is followed by arithmetic right shift as performed in case of “00”/”11”.

After moving out of the loop we reach to our final answer. The output is

represented in binary and its decimal equivalent is also written in the next line.

**Assumption:** It is important that while giving the no. of bits as input we give extra bit.

This becomes important as we are working on signed numbers. The extra bit will be

helpful to check the given number is positive or negative.Although we are giving input

in decimal format. For eg. If I want to multiply 8 and -7. I need to give the no. Of bits as

5 because 8 can be represented in 4 bits but an extra bit will check its sign avoiding

any Overflow while taking or adding two’s complement.