**Algorithm Design**

**1.Recursive algorithm using divide and conquer strategy to calculate maximum profit and day to buy and sell in single sell**

a.If one-day price is available then buy and sell immediately = no profit at all

if len(listOfStockValues) == 0:

return 0,0,0

size = stop - start

if size == 0:

return 0, start, start

# case 2: buy on the first day, sell on the second.

if size == 1:

return listOfStockValues[stop] - listOfStockValues[start], start, stop

b.if current value is always greater then previous value then print

# No solution possible since price is decreasing continuously

otherwise:

c.Split the array in half by calculating mid value divide by 2

d.then "divide" part in Divide & Conquer: try both halfs of the array

e.find the minimum buy price in the lower half and the maximum sell price in the upper half with recursive call

f.those two points generate the maximum cross border profit and now compare our three options and find the best one

g. if profitcalculatedfromSecondpart > profitcalculatedfromfirstpart:

if profit > profitcalculatedfromSecondpart:

return profit, stock\_buy\_idx, stock\_Sell\_idx

else:

return profitcalculatedfromSecondpart, buy2, sell2

else:

if profit > profitcalculatedfromfirstpart:

return profit, stock\_buy\_idx, stock\_Sell\_idx

else:

return profitcalculatedfromfirstpart, buy1, sell1

**Asymptotic complexity Analysis:**

The algo that has been implement "Divide and Conquer" uses Binary Search and the worst case

performance of Binary search is O(Log(n)). However, per General principle of implement "Divide

and Conquer" with Recursion, it takes a problem of size n. Now dividing it into a sub problem of size

n/b, would add linear complexity of (c\*n). With Recursion implementation, applying Master

Theorem-

T(n) = a T(n/b) + cn

T(1) = 1

With a=2, b=2, => a=b

T(n) = O(nlogn)

**2.Iterative algorithm to calculate maximum profit and day to buy and sell in single sell**

a.Get the length of **stock\_prices\_list**

#min\_index represents day on user should buy

#max\_index represents day on user should sell

b.Initialize min\_index , max\_index and diff with 0

# min\_ value represents price at which user should buy

# max\_ value represents price at which user should sell

c.Initialize min\_value and max\_value with first value of **stock\_prices\_list**

#Logic to calculate maximum and minimum value

d.Loop over **stock\_prices\_list** till second last item

Get the current\_value from **stock\_prices\_list**

Get the next\_value from **stock\_prices\_list**

Initialize global\_maxima to difference of maximum and minimum value

Initialize local\_maxima to difference of next\_value and current\_value

If(current\_value<min\_value and (global\_maxima<local\_maxima or diff<local\_maxima) :

Update min\_index to current index

Update min\_value to current\_value

if max\_index<min\_index & local\_maxima>diff:

Update max\_index to current index

Update max\_value to current\_value

Update diff to local\_maxima

if currentValue>=max\_value:

Update max\_index to current index

Update max\_value to current\_value

if last value in **stock\_prices\_list** is greater than max\_value

Update max\_index to last index of **stock\_prices\_list**

Update max\_value to last value of **stock\_prices\_list**

If min\_index and max\_index are not equal then

Return min\_index,min\_value, max\_index,max\_value

Else

Return “Cannot suggest solution”

**Asymptotic complexity Analysis:**

Time complexity: O(n)