# Theoretical Assignment 1

14282 14679

## 1. Maximum Sum Submatrix

**A** ← n x n Input Matrix
**S** ← n x n Matrix whereS [ i ][ j ] stores sum of submatrix ( i x j ) fromA [ 0 ][ 0 ] to A [ i ][ j ] { 0 <= i <= n-1; 0<=j<=n-1 }

```
Calculate_S ( A )
        S [ 0 ][ 0 ] ← 0
        for i =1 to n-1 do
                S [ i ][ 0 ] ← S [ i-1 ][ 0 ] + A [ i ][ 0 ]          //first column cumulative sum --> O(n)
                S [ 0 ][ i ] ← S [ 0 ][ i-1 ] + A [ 0 ][ i-1 ]        //first row cumulative sum --> O(n)
        end for
        for i =1 to n-1 do                                            // sum of the remaining matrix --> O(n^2)
                for j=1 to n-1 do
                        S [ i ][ j ] ← S [ i-1 ][ j ] + S [ i ][ j-1 ] – S [ i-1 ][ j-1 ] + A [ i ][ j ]
                }
        end for
End
```

***Time Complexity*** **→** ***2 * O(n) + O($n^2$) = O($n^2$)***

---

```
Sum_row ( i, j , k)                                   // sum of the kth row from A[k][i] to A[k][j] --> O(1)
        Return S[k][j] - S[k][i] –S[k-1][j] + S[k-1][i]
End
```

**Time Complexity → O(1)**

---

```
Traverse( A , S )
        max_sum ← S [ 0 ][ 0 ] ; max_x1← 0 ; max_y1←0 ; max_x2←0 ; max_y2←0
        for i = 0 to n-1                              // i, j varies width of submatrices --> O(n^2)
                for j = i to n-1
                        local_sum←Sum_row ( i , j , 0 )       // initializing local_sum to sum of first row
                        local_y1← 0                           // top row of submatrix
                        for k = 1 to n-1                      // traversing through all rows in submatrix --> O(n)
                                if ( local_sum > max_sum) then //updating max Sum Submatrix corner co-ordinates
                                        max_sum ←local_sum
                                        max_x1 ← i ; max_x2 ← j ; max_y1 ← local_y1 ; max_y2 ← k
                                end if
                                if (local_sum >= 0 ) then     //updating local sum
                                        local_sum ← local_sum + Sum_row ( i , j , k )
                                else
                                        local_sum←Sum_row(i,j,k)
                                        local_y1 ← k-1        //updating boundary
                                end if
                        end for

                end for
        end for
End
```

**Time Complexity → O($n^3$)**

**Total Time Complexity → O($n^2$) + n * O(1) + O($n^3$ ) → O($n^3$ )**

## 2. Range- Minima Problem

### Algorithm

We divide the array of size "n" in blocks of size "log(n)" each. We design a data structure such that we can calculate mini ma of any number of continuous blocks in O(1) time.

Using same intuition , we divide the blocks of size "log(n)" into sub-blocks of size "log(log(n))" such that we can calculate minima of any number of continuous "log(log(n))" sub-blocks in O(1) time.

Then, we can scan through the remaining elements of order log(log(n)) to get the minima O(log(log(n))) time**.**

### Data Structures

**A**← input array of size**n**

**Power-of-2** ← "n" sized array where Power-of-2 [ i ] stores the greatest number of the form $2^k$ such that $2^k <= i$

**Space → O(n)**

---

**Prev_log** ← **"n"** sized array wherePrev_log[ i ] stores the greatest integer k such that $2^k <= i$

**Space → O(n)**

---

**K_log** ← "n" sized array where K_log [ i ] stores the greatest integer k such that k*log(n) <= i <= (k+1)*log(n)

**Space → O(n)**

---

**B** ← {(n / log(n) ) x 2*log (n )} matrix structure where B [ i ] [ ±j ] stores minima of subarray from **A[ i * log(n) ]** to **A[ i * (log(n)) + *sign( j )*** $\mathbf{*2^{|j|}}$ **]**

**Space → O(n)**

---

**C** ← { n/ log(n) x log(n) / log(log(n)) x 2*log(log(n)) } matrix structure where C [ i ][ j ] [ ±k ] stores minima of subarray from **A[ i*log(n) + j*log(log(n))** ] to **A[ i*log(n) + j*log(log(n)) + sign(k)*** $\mathbf{2^{|k|}}$ **]** , k varies such that

$( i*log(n) + j*log(log(n)) + 2^k ) < (i+1)* log(n)$ **if** $k >0$

$( i*log(n) + j*log(log(n)) - 2^{-k} ) > (i)* log(n)$ **if** $k <0$

**Space → O(n)**

---

**Prev_loglog** ← { n/log(n) x log(n)} matrix wherePrev_loglog[ i ][ j ] stores the value of k where

$( i*log(n) + k*log(log(n)) ) <= j < ( i*log(n) + (k+1)*log(log(n)) )$

**Space → O(n)**

---

**Total Space → O(n)**

### Pseudo-Code

a,b ← query input

```
Minima(start, end)
    if (a = start) then
        start_block ← end / (log(n) ) – 1 ;

        left_block ← Prev_loglog[ start_block ][ start ]
        right_block ← Prev_loglog [start_block][ end ]

        diff ← (right_block – ( left_block + 1))* log(log(n)) ;
        p ← Power-of-2( diff )

        if ( diff =  p )then                    //  minima_mid --> O(1)
            minima_mid ← C [ start_block] [ left_block ] [ log(p) ]
        else
            minima_mid ←min( C [ start_block] [ left_block ] [ log(p) ], C[start_block] [ right_block ] [ -log(p) ] )
        end if

        minima_left ←{minimum of elements from A[start] to A [start_block*log(n) + (left_block +1)* log(log(n))  ] }

        minima_right ← {minimum of elements from A [start_block*log(n) + (right_block)* log(log(n)) ] to
A[ (start_block+1) * log(n) ] }                    // Time complexity for both minima_left andminima_right--> O(log(log(n)))

        Return min(minima_left ,  minima_mid , minima_right )

    end if

    if (b = end) then
        start_block ←start / log(n)
        left_block ← Prev_loglog[ start_block ] [ end ]

        diff ← left_block * log(log(n))
        p ← Power-of-2(diff)

        if(diff = p)then
            minima_mid ← C[ start_block ] [ 0 ] [ log(p) ]
        else
            minima_mid ← min ( C [ start_block] [0] [ log(p) ] , C[start_block] [ left_block] [-log(p)] )
                            //  minima_mid --> O(1)
        end if

        minima_right ←{minimum of elements from A[start_block*log(n) + left_block* log(log(n)) ] to A[ end ] }
                            // Time complexity for both minima_left andminima_right--> O(log(log(n)))

        Return min(minima_mid, minima_right)

    end if
End
```

**Time Complexity→ O( log(log(n)) )**

---

```
Range_minima(a,b)
    left_block ← K_log(a)
    right_block ← K_log(b)

    diff← (right_block- left_block -1)*log(n)
    p ← Power-of-2(diff)
    q ← Prev_log(diff)
```

```
        If ( diff = p ) then                                      //  minima_mid --> O(1)
                minima_mid ← B[ left_block+1 ][ q ]
        else
                minima_mid ← min( B[ left_block+ 1 ][ q ] , B[ right_block ] [-q ] )
        end if
        minima_left ← Minima( a  ,  (left_block+1)*log(n)  )      //minima_left -->  O( log(log(n)) )
        minima_right ← Minima( right_block*log(n) , b )           //minima_right --> O( log(log(n)) )

        Return min( minima_left , minima_mid , minima_right )
End
```

**Time Complexity → O(1) + 2*O( log(log(n)) ) → O( log(log((n)) )**