Kartik Raj - 14308
Sanjay Ramesh - 14601

Theoritical Assignment 1

Maximum Sum Submatrix

We are given the matrix A as input which is nxn.

Data Structure

1. nxnxn matrix S where S(i,j,k) denotes the maximum sum submatrix starting at kth row and ending at index (i,j).
O(n^3) space
2. nxn matrix s where s(i,j) denote the maximum sum submatrix ending at index (i,j).
O(n^2) space
3. nxnxn matrix B where B(k,i,j) denote the sum from kth row to ith row in jth column.
O(n^3) space
Space complexity of Algorithm = O(n^3)

Listing 1: Algorithm to calculate Maximum Sum Submatrix

```
Maxsum-submatrix(A)
{   "This is the initialization part for arrays used"
    for i=0 to n-1{
        for k=0 to i{
            S(i,0,k)="Summation of a[x][0] over x in range k to i"
            index(i,0,k)=0;
            //index(i,j,k) stores the starting column for the maxsum ↩
                matrix whose starting row k and ending index is (i,j)
        }   //O(n cube) time complexity
    }
    "Initialize each element of B(k,i,j) to be 0"
    for j=0 to n-1{
        for i=0 to n-1{
            for x=0 to i{
            B(0,i,j)=B(0,i,j)+a[x][j];
            }   //Initialized B(0,i,j) as per definition
        }   //O(n cube) time complexity
    }
    for j=0 to n-1{ //Filling the B(k,i,j) matrix
        for i=1 to n-1{
            for k=1 to i{
                B(k,i,j)=B(k-1,i,j)-a[k-1][j];
            }   //O(n cube) time complexity
        }
    }   //Note k<=i are the only valid entries as per definition
    "Algorithmic part"
    for i=0 to n-1{
        for j=1 to n-1{
            for k=0 to i{
                if S(i,j-1,k)>0 then { //same concept used for 1D array
                    index(i,j,k)=index(i,j-1,k);
                    //column index remains unchanged
                    S(i,j,k)=S(i,j-1,k)+B(k,i,j);
                }
                else {
                    index(i,j,k)=j; //column index updated
                    S(i,j,k)=B(k,i,j);
                }
            }
            s(i,j)="Max of S(i,j,k) over k in range 1 to i"
            k(i,j)="k which maximises s(i,j)"
        }   //O(n cube) time complexity
    }
    "Scan s to find the maximum entry"  //O(n square) time complexity
    "Return the corresponding(i,j) as ending index & (k(i,j),index(i,j,k))↩
        as starting index"
}
```

Time complexity of Algorithm=$O(n^3) + O(n^3) + O(n^3) + O(n^3) + O(n^2) = O(n^3)$

Range-Minima Problem-2nd part

We are given 1-D order n array A as input. We now divide this array into blocks of log(n) elements, and we store the minimum of each of these blocks in another array,say B. So B will be of size n/log(n).

Algorithm:

(I) This is the first part of the Algorithm. Suppose we need the minimum from ith to jth element in the original array A, suppose ith and jth element are involved in B[s] and B[t] of array B respectively. The minima of the blocks from B[s+1] to B[t-1] can be calculated using method discussed in class. It can be calculated in O(1) time. The number of elements in B is n/log(n), so the total space used will be

$$\begin{aligned} Space &= O(\frac{n}{log(n)}log(\frac{n}{log(n)})) \\ &= O(\frac{n}{log(n)}log(n)) \\ &= O(n) \end{aligned} \tag{1}$$

Now we need to (1) Find the minimum of the elements starting from A[i] to the element at the end of the block corresponding to B[s] (2) Find the minimum of the elements from the start of the block corresponding to B[t] to A[j] (3) Get the minimum of all the three minima.
In (1) and (2) we do simple comparision in the block to find the minimum element, and the maximum number of elements involved can be log(n). Hence these two takes O(log(n)) time. Note this takes no extra space.
Therefore **total time complexity** = O(log(n)), **total space complexity** = O(n)

(II) We now extend this solution to get complexity of O(log(log(n))). In this to find the minimum in (1) and (2), we apply the first part of the algorithm taking single block consisting of log(n) elements as the original array. As for n elements it takes O(log(n)) time, for log(n) elements it will take **O(log(log(n))) time complexity**.
And also as for n elements it takes O(n) space, space required for each block of log(n) elements is O(log(n)). As there are total n/log(n) blocks so the extra space involved will be O(n). Hence total **space complexity** remains **O(n)**.

Data Structure

Suppose we need to calculate the minimum from A[i] to A[j]. The following are the data structures involved.
1. Array Power-of-2[ ]
2. Array Log[ ]
3. Array B where B[i] consists the minimum of elements A[(i)log(n)] to A[(i+1)log(n)-1]
4. A 2D matrix C corresponding to B. This is similar to the matrix used in class. Each element C[i][k] consists of minimum of the elements B[i] to B[i+2^k].
5. A 3D matrix D where in D[x][i][k],we had divided block B[x] into blocks of log(log(n)) elements thus D[x][i][k] consists of minimum of the elements C[x][i] to C[x][i+2^k]. However, note space complexity of D is still O(n) as explained above in the algorithm.

Listing 2: Range-Minima Algorithm - Pseudo Code

```
int rangem(i,j,B)
{//This function returns the minimum from A[i] to A[j]. B is the 2D matrix↩
    corresponding to A.
    L = j-i;
    t = Power-of-2[L];
    k = Log[L];
    if (t=L) return B[i][k]
    else return min(B[i][k],B[j-t][k]);
}

int simple-minima(A,i,j)
{   //This function returns the minima from A[i] to A[j] by the brute ↩
```

```
        force method.
    min = A[i]
    for t=i+1 to j{
        if (A[t]<min) min = A[t]
    }
    return min
}

int main(){
    "Implement Data Structures as described above"
    "Find index s & t corresponding to i & j in B matrix"
    min=rangem(s+1,t-1,C);
    min1=range1(B,s,i,j);
    min2=range2(B,t,j);
    "Return minimum of min1,min2,and min"
}
int range1(B,s,i,j){
    "We needed to calculate minimum of A[i] to A[s] in O(log(log(n))) time↩
       "
    "We had already divided A[i] to A[s] in boxes of size log(log(n))"
    "Find index s1 corresponding to i in D[s] matrix"
    "Let E be a 2D array which is equal to D[s]"
    min=rangem(s1+1,s,E);
    min1=simple-minima(A,i,s1);
    "Return minimum of min and min1"
}
int range2(B,s,i,j){
    "We needed to calculate minimum of A[t] to A[j] in O(log(log(n))) time↩
       "
    "We had already divided A[t] to A[j] in boxes of size log(log(n))"
    "Find index t1 corresponding to j in D[s] matrix"
    "Let E be a 2D array which is equal to D[s]"
    min=rangem(t,t1-1,E);
    min2=simple-minima(A,t1,j);
    "Return minimum of min and min2"
}
```