CSL351: Assignment 4

SAHIL
2016UCS0008

Ans-4(a) The strassen's matrix multiplication algorithm is as follows:

step 1 Divide the input matrices A and B and output matrix C into $(n/2 \times n/2)$ sub-matrices.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

step 2 Create 10 matrices $S_1, S_2, \ldots, S_{10}$, each of which is a $n/2 \times n/2$ matrix and is the sum or difference of two matrices created in step 1.

$S_1 = B_{12} - B_{22}$
$S_2 = A_{11} + A_{12}$
$S_3 = A_{21} + A_{22}$
$S_4 = B_{21} - B_{11}$
$S_5 = A_{11} + A_{22}$
$S_6 = B_{11} + B_{22}$
$S_7 = A_{12} - A_{22}$
$S_8 = B_{21} + B_{22}$
$S_9 = A_{11} - A_{21}$
$S_{10} = B_{11} + B_{12}$

step-3 Using the submatrices created in step 1 and the 10 matrices created in step 2, recursively compute 7 matrix products $P_1, P_2, \ldots, P_7$

$P_1 = A_{11} S_1$
$P_2 = S_2 B_{22}$
$P_3 = S_3 B_{11}$
$P_4 = A_{22} S_4$
$P_5 = S_5 S_6$
$P_6 = S_7 S_8$
$P_7 = S_9 S_{10}$

step-4 Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding and subtracting various combinations of the P_i matrices.

$C_{11} = P_5 + P_4 - P_2 + P_6$

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

$C_{22} = P_5 + P_1 - P_3 - P_7$

The time complexity of this algorithm can be written as

$T(n) = 7T(n/2) + \theta(n^2)$

since we are computing 7 matrix products of size n/2 and all other additions, subtractions take $\theta(n^2)$ time.

on solving, we get $\boxed{T(n) = \theta(n^{\log_2 7})}$

(b) To understand how to modify the strassen's algorithm when $n \neq 2^k$, $k \in \mathbb{Z}$

Let us consider an example where $n = 3$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 & 5 \\ 3 & 4 & 6 \\ 1 & 2 & 3 \end{bmatrix}$$

In the first step, we need to divide A & B into 4 n/2 x n/2 matrices but $\frac{n}{2} = \frac{3}{2} = 1.5$ is not an integer.

So, let us make matrices A and B of size $(n+\delta) \times (n+\delta)$ where adding δ makes $(n+\delta)$ the next power of 2.

Here, $\delta = 1$ since $3+1 = 4 = 2^2$

we can add 0's in the extra cells generated

$$\Rightarrow A' = \left[\begin{array}{cc|cc} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ \hline 1 & 2 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right], \quad B' = \left[\begin{array}{cc|cc} 1 & 2 & 5 & 0 \\ 3 & 4 & 6 & 0 \\ \hline 1 & 2 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

Now, let us try to multiply them using Strassen's algorithm.

$$A'_{11} = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}, \quad A'_{12} = \begin{pmatrix} 3 & 0 \\ 6 & 0 \end{pmatrix},$$

$$A'_{21} = \begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix}, \quad A'_{22} = \begin{pmatrix} 3 & 0 \\ 0 & 0 \end{pmatrix}$$

$$B'_{11} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B'_{12} = \begin{pmatrix} 5 & 0 \\ 6 & 0 \end{pmatrix}$$

$$B'_{21} = \begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix}, \quad B'_{22} = \begin{pmatrix} 3 & 0 \\ 0 & 0 \end{pmatrix}$$

$A'B' = C'$

Let us calculate the products P_1 to P_7.

$$P_1 = A_{11}(B_{12} - B_{22})$$

$$= \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 8 & 0 \end{pmatrix}$$

$$P_2 = (A_{11} + A_{12}) B_{22}$$

$$= \begin{pmatrix} 4 & 2 \\ 10 & 5 \end{pmatrix} \begin{pmatrix} 3 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 12 & 0 \\ 30 & 0 \end{pmatrix}$$

$$P_3 = (A_{21} + A_{22}) B_{11}$$

$$= \begin{pmatrix} 4 & 2 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 10 & 16 \\ 0 & 0 \end{pmatrix}$$

$$P_4 = A_{22}(B_{21} - B_{11})$$

$$= \begin{pmatrix} 3 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ -3 & -4 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$P_5 = (A_{11} + A_{22})(B_{11} + B_{22})$

$= \begin{pmatrix} 4 & 2 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 4 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 22 & 16 \\ 31 & 28 \end{pmatrix}$

$P_6 = (A_{12} - A_{22})(B_{21} + B_{22})$

$= \begin{pmatrix} 0 & 0 \\ 6 & 0 \end{pmatrix} \begin{pmatrix} 4 & 2 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 24 & 12 \end{pmatrix}$

$P_7 = (A_{11} - A_{21})(B_{11} + B_{12})$

$= \begin{pmatrix} 0 & 0 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 6 & 2 \\ 2 & 4 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 39 & 28 \end{pmatrix}$

Thus, $C_{11} = P_5 + P_4 - P_2 + P_6$

$= \begin{pmatrix} 22+0-12+0 & 16+0-0+0 \\ 31+0-30+24 & 28+0-0+12 \end{pmatrix}$

$= \begin{pmatrix} 10 & 16 \\ 25 & 40 \end{pmatrix}$

$C_{12} = P_1 + P_2 = \begin{pmatrix} 14 & 0 \\ 38 & 0 \end{pmatrix}$

$C_{21} = P_3 + P_4 = \begin{pmatrix} 10 & 16 \\ 0 & 0 \end{pmatrix}$

$C_{22} = P_5 + P_1 - P_3 - P_7$

$= \begin{pmatrix} 22+2-10-0 & 16+0-16-0 \\ 31+8-0-39 & 28+0-0-28 \end{pmatrix}$

$= \begin{pmatrix} 14 & 0 \\ 0 & 0 \end{pmatrix}$

Thus, $A'B' = \left[\begin{array}{ccc|c} 10 & 16 & 14 & 0 \\ 25 & 40 & 38 & 0 \\ 10 & 16 & 14 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$

Clearly we can see that, the result $AB = C_{3\times3}$ is embedded and at locations where we inserted 0, we get 0 in the result.

Pseudocode for modified Strassen's algorithm

```
StrassenAlgo ( A, B, n):
    if (n is equal to 1) return A*B
    if (n is not a power of 2) then
        n' = ⌊log2(n)⌋ + 1
        n' = pow(2, n')
    else n' = n
    - Create empty matrices of size n'/2 x n'/2
      A11, A12, A21, A22, B11, B12, B21, B22, C11, C12,
      C21, C22, S1, S2, --, S10, P1, P2, ---, P7
    - Fill matrices Aij, Bij with values from A, B
      when index i and j exceed size of A, B fill with
      it is already filled with 0 at that position
    - Assign values of S1, S2, --, S10 from expressions
      given earlier
    - P1 ← strassenAlgo(A11, S1, (n')/2)
    - P2 ← strassenAlgo(S2, B22, n'/2)
    - P3 ← strassenAlgo(S3, B11, n'/2)
    - P4 ← strassenAlgo(A22, S4, n'/2)
    - P5 ← strassenAlgo(S5, S6, n'/2)
    - P6 ← strassenAlgo(S7, S8, n'/2)
    - P7 ← strassenAlgo(S9, S10, n'/2)
    - // Compute final product values C11, C12, C21, C22
    - C11 ← P5+P4-P2+P6;  C12 ← P1+P2;
    - C21 ← P3+P4;  C22 ← P5+P1-P3-P7;
    - Combine Cij into matrix C
    - Return Submatrix C(n x n) as answer.
```

Ans-5 **Stock Pricing Problem**

Consider an example of stock prices over n = 5 days

5, 4, (1), 2, (10)

We need to find on which day to buy the stock and the day on which to sell it, so as to maximize the profit.

In the above example, it is clear that we buy on day 3 and sell on day 5. Profit = 10 − 1 = 9.

* So, a simple brute force approach is to try all (start_day, end_day) pairs where end_day > start_day and find profit in each pair. The maximum over all can be taken then.

* Also, we can model the given problem as the maximum subarray problem by considering a new array containing the difference of prices between two consecutive days.

In above eg, new array

d = {−1, −3, 1, 8}

Clearly, the subarray {1, 8} has max sum = 1 + 8 = 9

Since subarray starts from 3rd element and ends at 4th element, we buy stock on day 3 and sell on day 5.

* So, we can apply the divide and conquer approach to solve maximum subarray sum problem as done in previous assignment.

The start day will be the beginning element of max subarray sum subarray and end day will be one more than the end element of max sum subarray.

Ans-6 (a) $A = \{6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2\}$

First, we create a count array.

After iterating over all elements of array we get:

count

2	2	2	2	1	0	2
0	1	2	3	4	5	6

Index:

Now, we do $count[i] = count[i-1] + 1$

$\forall \; 1 \leq i \leq n = 6$

So, that we get the no. of elements less than or equal to i at ith index.

count

2	4	6	8	9	9	11
0	1	2	3	4	5	6

Now, we simply insert the elements of A (reverse traversal) in sorted array srt-A using the count[element] as the index to store it and then decrement count[element] by 1.

srt-A

	0	0	1	1	2	2	3	3	4	6	6
0	1	2	3	4	5	6	7	8	9	10	11

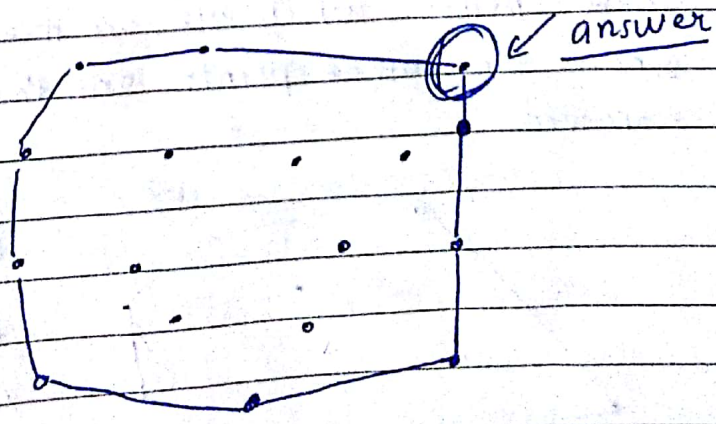
(b) Counting array is a stable sort, i.e. multiple keys with the same value are placed in the sorted array in the same order that they appear in the input array.

Proof - Consider two elements which are equal at indices i & j and in the input array where $i < j$. In sorted array the element at index i must come before element at index j, since we process the array in reverse order, and after inserting an element at index in sorted array

given by count[element], we decrement count by 1, i.e. the index also decreases.
So, element at higher index j in A, gets inserted ~~earlier~~ first in reverse traversal but at higher index of sorted array than that at index 'i'.
Hence, any 2 equal elements will be inserted in the order they appear in input, in the final sorted array.

Ans-7 (a) A simple brute force algorithm to find the no. of vowels in a text is to just iterate over all the characters in the text, and check if it is a vowel, the increment the vowel count by 1.

Ans-8 (a) To determine the largest coordinate of the convex hull of a set of n≥1 points in the plane, we consider the point with the largest x-coordinate. Since there can be multiple such points, we need to consider only the one with the largest y-coordinate among them.



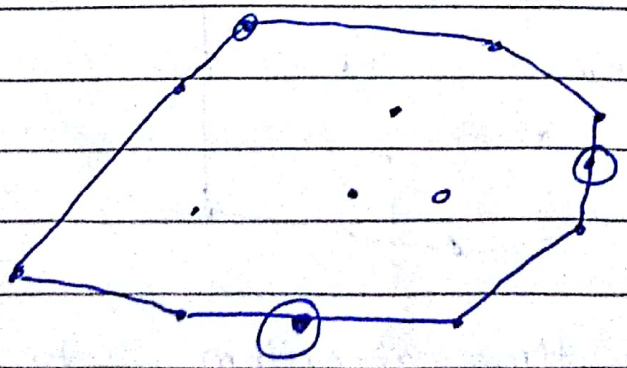
So, a linear time algorithm is to just iterate over all the points and keep checking if its x-coordinate is larger, if its then update the answer. Also, if the

x-coordinate is equal to the current largest value, then compare y-coordinat with current largest y-coordinate, if its then update it.

Algorithm -

```
largest_x = -∞
largest_y = -∞
for all the given points
    if (point.x > largest_x)
        largest_x = point.x
        largest_y = point.y
    else if (point.x == largest_x)
        if (point.y > largest_y)
            largest_y = point.y
// now (largest_x, largest_y) is the answer.
```

(b) To handle more than two points on the same straight line, in the brute-force algorithm for the convex hull, we can only consider those points which are on the extreme of the line, the other points on the line can be removed.



We can remove the points marked in circle in this example.

DATE / /

Ans-1 We can use depth first search for solving this. From each 1 entry, we can make a recursive call to all 8 neighbours. In each of the dfs calls, we keep track of visited 1's and update the answer with maximum value.

Pseudocode-

```
largestRegion ( Matrix M, Rows r, Columns c ) =
    visited [r][c] ← false for each entry
    result = -∞
    for i ← 0 to r-1
        for j ← 0 to c-1
            if (M[i][j]==1 and not visited[i][j])
              then  count ← 1
                    DFS (M, i, j, R, C, visited, count)
                    result = max (result, count)
    return result

DFS ( Matrix M, i, j, Rows r, Col c, Matrix visited, count)
    visited [i][j] ← true
    for all neighbours of (i,j)
        // nextx, nexty denote the neighbour
        if (nextx >= 0 and nexty >= 0
            and next < r and nexty < c )
        then if (not visited [nextx][nexty]
                 and M[nextx][nexty]==1)
             then
             DFS (M, nextx, nexty, R, C, visited, count+1)
             count ← count + 1
```

Ans- 2 **Algorithm for Reversing words in a sentence**

We can do this in the following manner, In the example provided "This is a Career Monk String", we can first reverse the individual words, ~~and then~~ to get

"sihT si a reraC knoM gniriS"

and now if we reverse the whole string, we get the required result

"String Monk Career a is This".

So, the algorithm is -

Step-1 Reverse the individual words

Step-2 Reverse the whole string

We can simply find individual words by identifying a space, mark the begin and end of word and reverse as we iterate the string. Since ~~reverse~~ reverse takes $O(n)$ time where n = length of string to be reversed

Total complexity,

$$T(n) = \underbrace{\left(\sum_{i=1}^{noof\_words} O(\text{length of word})\right)}_{\text{Step 1}} + \underbrace{O(n)}_{\text{Step 2}}$$

$$= O(n) + O(n)$$

Thus, $\boxed{T(n) = O(n)}$

Ans-3 Algorithm to find if the pattern is present in the 2D array,

It is given that pattern can be in any order (all 8 neighbours can be considered) but we can't use the same character twice.

* So, we can use exhaustive search technique to start from that character which matches with the first character of the pattern, then we can repeat the search recursively to its neighbours but considering pattern from next element. Also, we mark a character in 2D array visited so that we don't consider it again. If after all recursive calls the whole character pattern matches, we return true.

* We repeat the search from all locations where the first character matches till we find it or no more locations are left, reinitializing visted array to false.