National University of Computer & Emerging Sciences, Karachi
School of Computing
Mid-2 Examination, Fall 2023
07th November 2023, 10:00 am – 12:00 pm

Course Code: CS-2009	**Course Name: Design and Analysis of Algorithm**
Instructors: Dr. Muhammad Atif Tahir, Dr. Fahad Sherwani, Dr. Farrukh Salim, Dr. Nasir Uddin, Mr. Faisal Ahmed, Mr. Sandesh Kumar, Ms. Anaum Hamid, Mr. Syed Faisal Ali, and Mr. Minhal Raza	
Student Roll No:	**Section:** BCS-5A, B, C, D, E, F, G, H, J, K, BSE-5A, B, BAI-5A, B, BCY-5A, B

Instructions:

- Return the question paper
- Read each question completely before answering it. There are **6 questions** on **11 pages**
- In case of any ambiguity, you may make assumptions. But your assumption should not contradict any statement in the question paper

Time: 120 Minutes **Max Marks:** 17.5

Question#1 [2.5 Points, CLO:3]

Compute the prefix (failure) function π for the pattern 'aaaaha'. The algorithm to compute prefix function for the Knuth-Morris-Pratt (KMP) algorithm is given below.

Show the steps for all the iterations, that contribute to any change in the value of 'k', in order to finally find each value of π [q].

```
COMPUTE-PREFIX-FUNCTION(P, m)
1  let π[1 : m] be a new array
2  π[1] = 0
3  k = 0
4  for q = 2 to m
5       while k > 0 and P[k + 1] ≠ P[q]
6            k = π[k]
7       if P[k + 1] == P[q]
8            k = k + 1
9       π[q] = k
10 return π
```

Q1

Marking Scheme: Correct 2.5 Points with accurate fail values. Partially Correct with good attempt: 1.0-1.75; Partial attempt: 0.5 - 0.75;

Solution:

Initially:

π [1] = 0, i.e.

q	1	2	3	4	5	6
P[q]	a	a	a	a	h	a
π [q]	0					

k = 0, q = 2

then P[1] == P[2], k = 1

q	1	2	3	4	5	6
P[q]	a	a	a	a	h	a
π [q]	0	1				

k=1, q = 3

then P[2] == P[3], k = 2

q	1	2	3	4	5	6
P[q]	a	a	a	a	h	a
π [q]	0	1	2			

k = 2, q =4

then P[3] = P[4], k =3

q	1	2	3	4	5	6
P[q]	a	a	a	a	h	a
π [q]	0	1	2	3		

k =3, q = 5

then k>0 and P[4] != P [5], So, k = π [3] = 2

then k>0 and P[3] != P [5], So, k = π [2] = 1

then k>0 and P[2] != P [5], So, k = π [1] = 0

q	1	2	3	4	5	6
P[q]	a	a	a	a	h	a
π [q]	0	1	2	3	0	

k =0, q = 6

then P[1] == P[6], k = 1

q	1	2	3	4	5	6
P[q]	a	a	a	a	h	a
π [q]	0	1	2	3	0	1

Question#2 [4.5 Points; CLO:4]

While computing product of chain of matrices Ahmed is curious to modify the dynamic programming algorithm that he learned in class to compute the worst possible way to find it. He has to compute the product of $A1A2A3A4 \ldots . An$. Can you help him to the modify the chain of matrix multiplication technique, that you learned in algorithm class, to achieve his goal. If he decides to store the sub problems optimal solutions in matrix M[n, n]. Then

a) Write a recursive relation for $M[i,j]$, $where\ 1 \leq i \leq j \leq$ n
b) Use your proposed technique to find the worst multiplication parenthesizing way for multiplying $A1_{5x10}\ A2_{10x3}\ A3_{3x15}\ A4_{15x20}$

Assume that the dimensions of matrix Ai is $d_{i-1}\ X\ di$, i.e., $A1 = A_{d0\ x\ d1}$.

Q2:

a) 1 Points for correct recursive relation; and 0.5-0.75 for a partial attempt

b) 2.5 Points for finding the maximum cost of multiplication and the worst parenthesis. 1 – 2 for just finding the cost of multiplication and putting values in the table. 0.5 – 0.75 for partial attempt

Solution a:

$$M[i,j] = 0, for\ i = j$$

$$M[i,j] = max\{M[i,k] + M[k+1,j] + d_{i-1}d_k d_j\}$$

Solution b:

	A_1	A_2	A_3	A_4
A_1	0	150	1200 $[A_1\ A_2]$	4450 $[A_1\ A_2]$
A_2		0	450	3450 $[A_3\ A_4]$
A_3			0	900
A_4				0

Multiplication required for $A_1A_2 = 5X10X3 = 150$

Multiplication required for $A_2A_3 = 10X3X15 = 450$

Multiplication required for $A_3A_4 = 3X15X20 = 900$

Multiplication required for $(A_1)(A_2A_3) = 0 + 450 + 5X10X15 = 1200$

Multiplication required for $(A_1A_2)(A_3) = 150 + 0 + 5X3X15 = 375$

Multiplication required for $(A_2)(A_3A_4) = 0 + 900 + 10X3X20 = 1500$

Multiplication required for $(A_2A_3)(A_4) = 450 + 0 + 10X15X20 = 3450$

Multiplication required for $(A_1)(A_2A_3A_4) = 0 + 3450 + 5X10X20 = 4450$

Multiplication required for $(A_1A_2)(A_3A_4) = 150 + 900 + 5X3X20 = 2900$

Multiplication required for $(A_1A_2A_3)(A_4) = 1200 + 0 + 5X15X20 = 2700$

Hence the worst multiplication parenthesizing would be $(A_1)((A_2A_3)A_4)$

Question#3 [3.5 Points, CLO:4]

A farmer wants to purchase land for agriculture purposes. On the map he has plotted the points which are P1(5,8), P2(6,1), P3(7,3), P4(5,4), P5(1,2), P6(4,5). He surveyed the land and found some benefits and some drawbacks. His observations are as follows:

P1(5,8) rocky land,
P2(6,1) pond,
P3(7,3) fertilized land,
P4(5,4) dry land,
P5(1,2) well,
P6(4,5) fertilized land.

A). Find the boundary points from the above-mentioned points through calculations.
B). Is it possible to avoid the lands which are not useful for fertilization?

Hint: The following formula may be used for calculations

$$\theta = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

Angles in degrees = Angles in radians x (180/pi)

$$(y_3 - y_2) * (x_2 - x_1) - (y_2 - y_1) * (x_3 - x_2)$$

Q3:

3 points for calculating the boundary points using formulas and by drawing the graph. 1.5 – 2 points for calculating the boundary points by drawing the graph only. 0.5 – 0.75 for some partial attempt.

0.5 points for identifying the boundary point and 0.25 for partial attempt.

Solution:

Using Graham Scan algorithm calculate the Convex hull for the points P1(5,8),P2(6,1),P3(7,3),P4(5,4),P5(1,2),P6(4,5)

The Graham Scan algorithm is a method for finding the convex hull of a set of points in the plane. Here's a step-by-step process to apply the Graham Scan algorithm to find the convex hull for the given points:

Define the points and sort them by polar angle with respect to the lowest point (P5) in counterclockwise order.
Initialize an empty stack to store the convex hull points.
Iterate through the sorted points and use the stack to keep track of the convex hull.
Check the orientation of each consecutive triplet of points to determine whether they make a left turn or a right turn.
If a right turn is encountered, remove the second-to-last point from the stack.
At the end of the process, the stack will contain the points of the convex hull in counterclockwise order.
Let's apply this algorithm to the given points P1(5,8), P2(6,1), P3(7,3), P4(5,4), P5(1,2), and P6(4,5). First, we need to sort the points based on their polar angles with respect to the lowest point (P5). After sorting, we will apply the Graham Scan algorithm to find the convex hull.

To apply the Graham Scan algorithm to the given set of points:

Calculate the polar angle of each point with respect to the lowest point (P5).
Sort the points based on their polar angles.
Use the sorted points to find the convex hull.
Let's perform the calculations:

Given points:
P1(5,8), P2(6,1), P3(7,3), P4(5,4), P5(1,2), P6(4,5)

Calculating polar angles with respect to P5(1,2):

P1(5,8) - polar angle = atan2(6, 6) = 45 degrees
P2(6,1) - polar angle = atan2(5, -1) = 83.66 degrees
P3(7,3) - polar angle = atan2(6, 1) = 75.96 degrees
P4(5,4) - polar angle = atan2(4, 2) = 63.43 degrees
P6(4,5) - polar angle = atan2(3, 3) = 45 degrees

Sorted points based on polar angles:
P5(1,2), P6(4,5), P4(5,4), P3(7,3), P1(5,8), P2(6,1)

Now, using the Graham Scan algorithm, we will find the convex hull of these points.

The Graham Scan algorithm is as follows:

Initialize an empty stack.
Sort the points by polar angle with respect to the lowest point.
Iterate through the sorted points:
While the current point and the last two points in the stack do not make a counterclockwise turn, pop the last point from the stack.
Push the current point onto the stack.
The stack will contain the points of the convex hull.
Applying the Graham Scan algorithm to the sorted points:

Sorted points based on polar angles:
P5(1,2), P6(4,5), P4(5,4), P3(7,3), P1(5,8), P2(6,1)

The convex hull points would be:
P5(1,2), P6(4,5), P1(5,8), P3(7,3), P2(6,1)

Therefore, P4 cannot be avoided as it is in the convex Hull.
P5 is at the corner point so we can avoid it.

Question#4 [3.5 Points, CLO:3]

Design an O(n) algorithm that sorts in alphabetical order by first grouping them by the last character of the last name, then by the second last character, and so on, until you reach the first character of the last name. Then, you can repeat the same process for the first name. So that you can easily find the names that start or end with a certain letter or sound.

Your answer should include a clear explanation of your designed algorithm and the reasoning behind your choice. Use a list of names of 5 Pakistan Cricket Team players in the format Firstname Lastname, "Babar Azam", "Haris Rauf", "Hasan Ali", "Shadab Khan", "Saud Shakeel" to check whether the result is correct from your design. Show all the steps in each iteration.

Q4:

3.5 points for designing the algorithm with explanation and dry run should be performed by showing all steps of iteration

1.5 – 3 points for performing the dry run and showing all steps of iteration

0 – 1.25 points for a partial attempt.

Solution:

Algorithm:

1. Create buckets for each character ('A' to 'Z') for both last names and first names.
2. Initialize an output array to store the sorted names.

Step 1 (Sorting by the last name):

- Starting from the last character of the last name, place each name in the respective bucket based on the character.
- Repeat the process for the second last character, and so on, until the first character of the last name.
- Gather the names from each bucket in alphabetical order and update the output array.

Step 2 (Sorting by the first name):

- Repeat the same process as in Step 1, but this time sort based on the characters in the first name.
- Update the output array with the names from each bucket.

Step 1: Sorting by the last name:

Buckets:

- A: []
- I: []
- L: []
- M: []
- N: [Hasan Ali]
- U: [Babar Azam]
- R: [Haris Rauf]
- S: [Shadab Khan, Saud Shakeel]

Output after sorting by the last name:

- Hasan Ali
- Babar Azam
- Haris Rauf

- Shadab Khan
- Saud Shakeel

Step 2: Sorting by the first name:

Buckets:

- A: [Babar Azam]
- H: [Hasan Ali, Haris Rauf]
- S: [Shadab Khan, Saud Shakeel]

Output after sorting by the first name:

- Babar Azam
- Hasan Ali
- Haris Rauf
- Shadab Khan
- Saud Shakeel

Final sorted list of names:

- Babar Azam
- Hasan Ali
- Haris Rauf
- Shadab Khan
- Saud Shakeel

The final output is the sorted list of names: "Babar Azam," "Hasan Ali," "Haris Rauf," "Shadab Khan," and "Saud Shakeel." This custom radix sort algorithm allows you to easily find names that start or end with a certain letter or sound.

Question#5 [3.5 Points, CLO:4]

The National Highway Authority is planning to build two fast lanes between Karachi and Hyderabad. Assume, each lane is represented by a line L which consists of two points. These 2 fast lanes should not cross each other.

(a) Design algorithm with O(1) complexity to accomplish that task
(b) Which two lines should be selected among the three below?
L1 = {(2,2), (2,5)}, L2 = {(1,1), (4,5)}, L3 = {(3,2), (3,4)}

Hint: The following Formula may be used to design the algorithm

$$2 \times Area(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$$

Q5:

1 points for a correct algorithm. 0.5– 0.75 points for a partially correct algorithm. 0.25 – 0.5 points for partial attempt.

2.5 points for the correct solution. 1 – 2 points for a partially correct solution. 0.5 – 0.75 for partial attempt.

Solution:

(a) 4 ccw computations to check whether 2 lines are intersecting or not

$$2 \times Area(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$$

```
public static boolean intersect(Line l1, Line l2)
{
   int test1, test2;
   test1 = Point.ccw(l1.p1, l1.p2, l2.p1)
           * Point.ccw(l1.p1, l1.p2, l2.p2);
   test2 = Point.ccw(l2.p1, l2.p2, l1.p1)
           * Point.ccw(l2.p1, l2.p2, l1.p2);
   return (test1 <= 0) && (test2 <= 0);
}
```

b) Compute intersection b/w i) L1,L2 ii) L1,L3 iii) L2,L3 and any which return 0 will be selected

e.g. intersect(L1,L2)

test1 = ccw((2,2), (2,5),(1,1)) * ccw((2,2), (2,5),(4,5)) = -3, so true and intersection

Line1 and Line3 will return False

Question#6 [2 Points; CLO:3] Perform Dry Run the following given algorithm on this array String Array = ["bed", "apple", "cat", "dog", "ball", "zoo", "banana"]

Q6:Already adjusted in Above Question(s) . Q-6 has been discarded now.

```
Algo_Test1(String[] arr, int l, int h, int p) {
        if (low < high && position < arr[0].length()) {
           int[] bounds = Algo_Test2(arr, l, h, p);
           Algo_Test1(arr, l, bounds[0] - 1, p);
           Algo_Test1(arr, bounds[1], h, p + 1);
        }
    }
   int[] Algo_Test2(String[] arr, int l, int h, int p) {
        int min = l;
        int equal = l;
        int max = h;
        int key = arr[l].charAt(p);

        while (equal <= max) {
            int curr = arr[equal].charAt(p);
            if (curr < key) {
                swap(arr, min, equal);
                min++;
                equal++;
            } else if (current == key) {
                equal++;
            } else {
                swap(arr, equal, max);
                max--;
            }
        }
        return new int[]{min, equal};
    }
```

Solution:

A 3-way radix quicksort is a variation of the quicksort algorithm that uses radix partitioning to sort strings. In this algorithm, strings are partitioned into three groups based on their radix (character at a particular position), and then the algorithm recursively sorts these groups. Let's perform a dry run of a 3-way radix quicksort on an example string array.

Choose a key character position. In this example, let's start with the rightmost character (position 2).

Perform a radix partition based on the chosen character position (position 2 in this case). Strings are grouped into three categories: less than key, equal to key, and greater than key:

Less than key: ["apple", "ball", "cat"]

Equal to key: ["bed"]

Greater than key: ["banana", "dog", "zoo"]

Recursively apply the 3-way radix quicksort to each of the three groups. Repeat steps 1 and 2 for each group.

Group 1 (Less than key):

Choose a key character position. Let's choose position 1.

Radix partition based on character at position 1:

Less than key: ["apple", "ball"]

Equal to key: []

Greater than key: ["cat"]

Since the equal group is empty, there's nothing to do. Recursively apply the 3-way radix quicksort to the "less than" and "greater than" groups. This step is not shown in detail, but the result is:\

["apple", "ball", "cat"]

Group 2 (Equal to key):

Since there's only one string in this group, it is already sorted: ["bed"]

Group 3 (Greater than key):

Choose a key character position. Let's choose position 1.

Radix partition based on character at position 1:

Less than key: []

Equal to key: []

Greater than key: ["banana", "dog", "zoo"]

Since both the less than and equal groups are empty, there's nothing to do. Recursively apply the 3-way radix quicksort to the "greater than" group. This step is not shown in detail, but the result is:

["banana", "dog", "zoo"]

Combine the sorted groups to get the final sorted array:

["apple", "ball", "bed", "cat", "banana", "dog", "zoo"]