Question 1 solution

Students are encouraged to write the algorithm in pseudocode form using this code.

```
#include<iostream>
#include<string>
using namespace std;

int main()
{
	int matrix1[9][9] = { 'A','B','C','B','A','B','C','T','Y',
			      'F','G','H','J','K','L','H','G','V',
			      'D','V','T','B','Y','N','G','B','G',
			      'F','D','G','A','B','C','V','D','E',
			      'V','J','O','F','G','H','H','F','W',
			      'S','R','V','D','V','T','W','F','G',
			      'K','C','D','E','C','C','A','B','C',
			      'X','Q','R','T','U','O','F','G','H',
			      'T','Y','I','O','B','F','D','V','T' };

	int pattern[3][3] = {'A','B','C','F','G','H','D','V','T'};

	int patternsum = 0;
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			patternsum += pattern[i][j];
		}
	}
	int num1 = 11;
	int num2 = 13;
	int mod1=patternsum % num1;
	int mod2=patternsum % num2;
	int col1=0,col2=0,col3 = 0;
	int hash1;
	int hash2;
	int temp = 0;
	int spurious = 0;
	int valid = 0;
	bool flag = false;

	for (int i = 0; i < 7; i++)
	{
		for (int j = 0; j < 7; j++)
		{
			if (j == 0)
			{
				col1 = col2 = col3 = 0;
				for (int k = i; k < i+3; k++)
				{
					col1 += matrix1[k][0];
					col2 += matrix1[k][1];
					col3 += matrix1[k][2];
				}
				hash1 = (col1 + col2 + col3) % num1;
				hash2 = (col1 + col2 + col3) % num2;

			}
			else
			{

				temp = 0;
```

```
                        col1 = col2;
                        col2 = col3;
                        col3 = matrix1[i][j + 2] + matrix1[i + 1][j + 2] +
matrix1[i + 2][j + 2];
                        hash1 = (col1 + col2 + col3) % num1;
                        hash2 = (col1 + col2 + col3) % num2;
                }
                if (hash1 == mod1 && hash2 == mod2)
                {
                        flag = false;
                        int x = 0;
                        int counter = 0;
                        for (int m = i; m < i + 3; m++)
                        {
                                int y = 0;
                                for (int n = j; n < j + 3; n++)
                                {

                                        if (matrix1[m][n] == pattern[x][y])
                                        {
                                                counter++;
                                        }
                                        else
                                        {
                                                cout << "spurious hit at: " << i
<< " " << j << endl;

                                                spurious++;
                                                flag = true;
                                                break;
                                        }
                                        y++;
                                }
                                x++;
                                if (flag == true)
                                {
                                        break;
                                }
                        }
                        if (counter == 9)
                        {
                                cout << "Pattern found at: " << i << " " << j
<< endl;

                                valid++;
                        }

                }
                else {}

        }
    }

    cout<< "Valid: " << valid << endl;
    cout << "Spurious: " << spurious << endl;
}
```

Question 2: Heapsort

a) Starting with the procedure MAX-HEAPIFY, write pseudocode for the procedure MIN-HEAPIFY (A, i, n), which performs the corresponding manipulation on a min-heap. How does the running time of MIN-HEAPIFY compare with that of MAX-HEAPIFY?

MIN-HEAPIFY(A, i, n)

```
L = LEFT(i)
R = RIGHT(i)
if L ≤ A.heap-size and A[L] < A[i]:
   smallest = L
else smallest = 1
if R ≤ A.heap-size and A[R] < A[smallest]:
   smallest = R
if smallest ≠ i:
   exchange A[i] with A[smallest]
  MIN-HEAPIFY(A, smallest, n)
```

There is no difference in running time between MAX-HEAPIFY and MIN-HEAPIFY.

b) Considering the function of Build-Max-Heap, write pseudocode for the procedure BUILD-MIN-HEAP.

BUILD-MIN-HEAP(A)

```
   n = length[A]
    for i ← ⌊n/2⌋ downto 1
         do MIN-HEAPIFY(A, i, n)
```

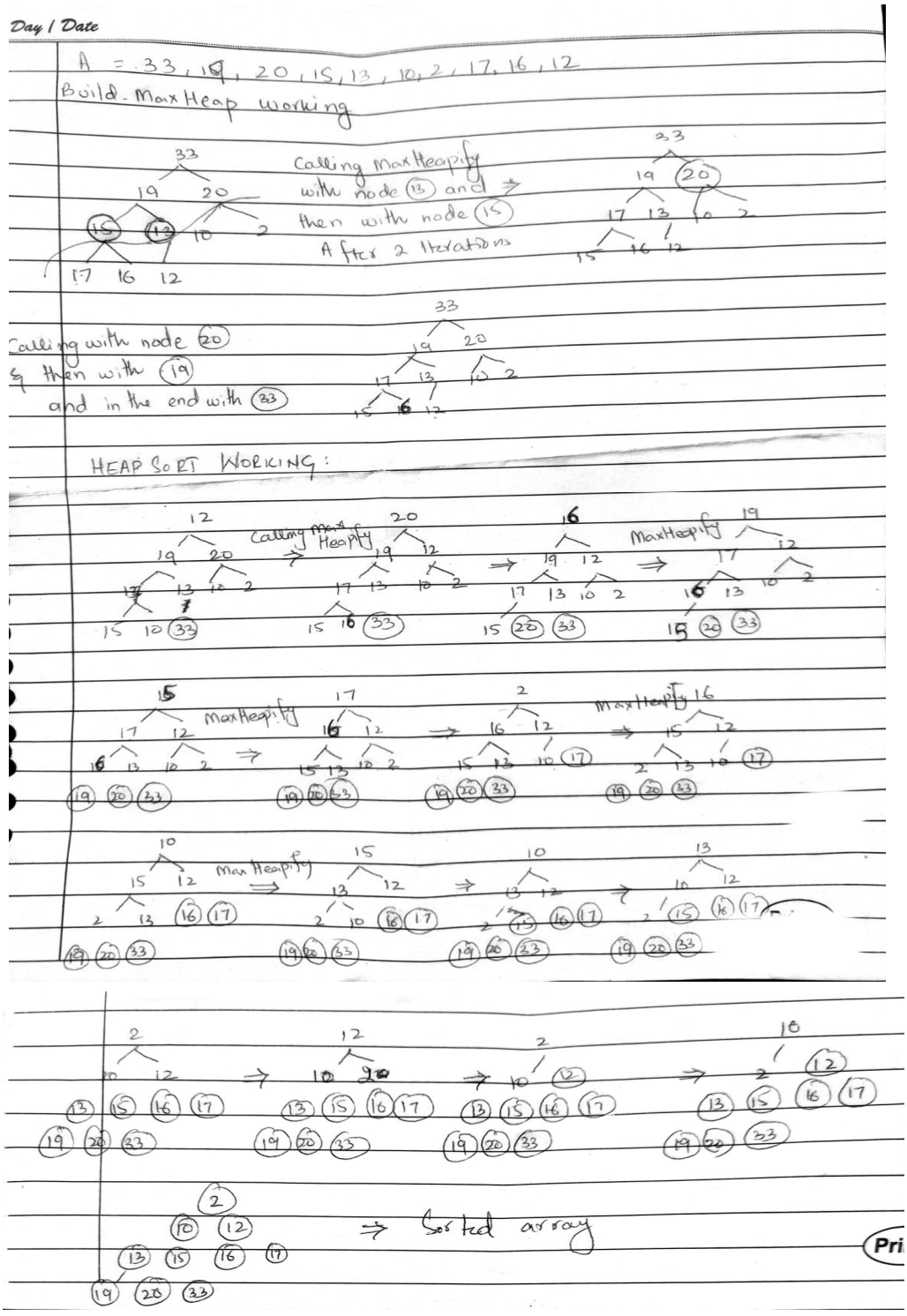
c) Comment on whether or not will there be any change in the procedure HEAP-SORT keeping in mind the modifications done in part (a) and 9(b). Give valid arguments.

HEAPSORT(A)

```
BUILD-MIN-HEAP(A)
 for i ← length[A] downto 2
      do exchange A[1] ↔A[i]
 MIN-HEAPIFY(A, 1, i - 1)
```

The sort will now produce data in descending order.

d) Illustrate the operation of BUILD-MAX-HEAP and then HEAP-SORT on the array A= <33, 19, 20, 15, 13, 4, 2, 17, 16, 12> with root at index 1. Show complete steps with elements of array after each operation.



Question 3: String Matching

a) Suppose that all characters in the pattern P are different. Show how to accelerate NAIVE-STRING-MATCHER to run in O(n) time on an n-character text T.

```
Accelerated_naive_algo(T,P,n,m)
BEGIN
        s ← 0
        WHILE s ≤ n-m DO
                FOR (j : 1 to m) DO
                        IF P[j] ≠ T[s+1] THEN
                                break
                        END IF

                        IF j = m THEN
                                PRNT "Pattern occurs at shift : ", s-m
                        END IF

                        s ← s + 1
                END FOR
        END WHILE
END
```

b) You are supposed to find out how many spurious hits does the Rabin-Karp matcher encounter in the text T = 3141592653589793 when looking for the pattern P =26 working with modulo q calculated by using the given formula.

i = 11

q = (Your registration number) mod i

if q = 0 repeat calculation after incrementing i by 1 and keep repeating until a legal value is found.

For example

i) if registration number is i23-2159 then q = 2159 mod 11 = 3.
ii) If q = registration number mod 11 = 0 then do
q = registration number mod 12 if again q = 0 then do
q = registration number mod 13 until a valid value is found

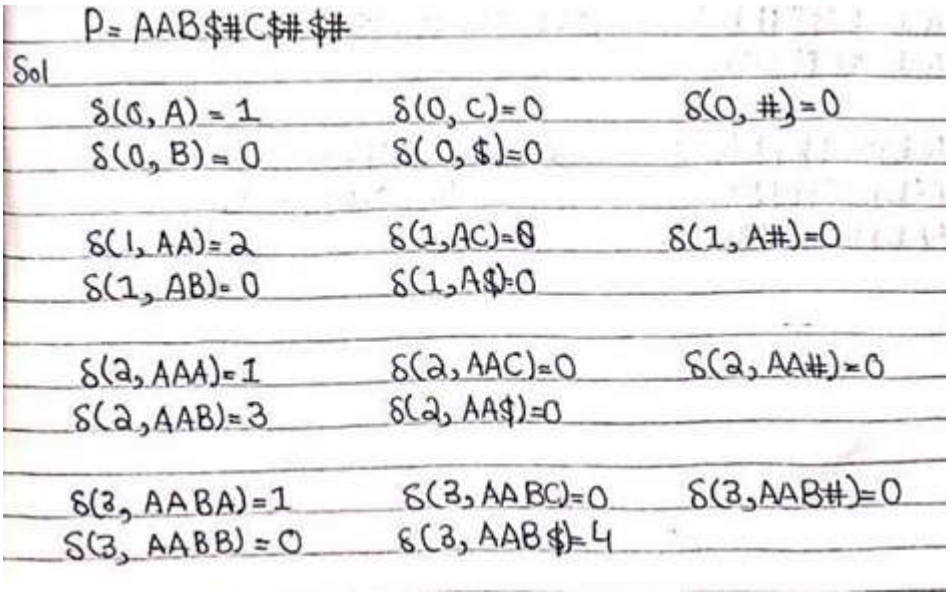
Given, P=26
T = 3141592653589793
Here, we have taken q=11
P mod Q = 26 mod 11=4
Now find the exact match of P mod Q
Now if S=0, 31 mod 11 = 9≠ 4

If S=1, 14mod 11=3 ≠ 4
If S=2, 41mod11 ≠ 4
S=3, 15 mod 11 = 4 (Spurious Hit)
S=4, 59mod 11 = 4 (Spurious Hit)
S=5, 92mod 11 = 4 (Spurious Hit)
S=6, 26mod 11 = 4 (Valid Match)
. . .65mod 11 ≠ 4
. . .53mod 11 ≠ 4
. . .35mod 11 ≠ 4
. . .58mod 11 ≠ 4
. . .89mod 11 ≠ 4
. . .97mod 11 ≠ 4
. . .79mod 11 ≠ 4
. . .93mod 11 ≠ 4
Therefore, pattern P occurs at shift = 6
Hence, the number of times spurious hits the Rabin-Karp matcher encounter in the text T, when looking for the pattern P = 26 is 3.

Question # 4

Create a transition table for the pattern P = AAB$#C$#$# by computing transition function for all states:

a) Create Finite Automaton using the transition table, which you have formulated above.
b) Show the trace of using your automaton for the text, T = ##$CAAB$#C$#$#B



$\delta(4, AAB\$A) = 1$ $\delta(4, AAB\$C) = 0$ $\delta(4, AAB\$\#) = 5$
$\delta(4, AAB\$B) = 0$ $\delta(4, AAB\$C) = 0$

$\delta(5, AAB\$\#A) = 1$ $\delta(5, AAB\$\#\$) = 0$
$\delta(5, AAB\$\#B) = 0$ $\delta(5, AAB\$\#\#) = 0$
$\delta(5, AAB\$\#C) = 6$

$\delta(6, AAB\$\#CA) = 1$ $\delta(6, AAB\$\#C\$) = 7$
$\delta(6, AAB\$\#CB) = 0$ $\delta(6, AAB\$\#C\#) = 0$
$\delta(6, AAB\$\#CC) = 0$

$\delta(7, AAB\$\#C\$A) = 1$ $\delta(7, AAB\$\#C\$\$) = 0$
$\delta(7, AAB\$\#C\$B) = 0$ $\delta(7, AAB\$\#C\$\#) = 8$
$\delta(7, AAB\$\#C\$C) = 0$

$\delta(8, AAB\$\#C\$\#A) = 1$ $\delta(8, AAB\$\#C\$\#\$) = 9$
$\delta(8, AAB\$\#C\$\#B) = 0$ $\delta(8, AAB\$\#C\$\#\#) = 0$
$\delta(8, AAB\$\#C\$\#C) = 0$

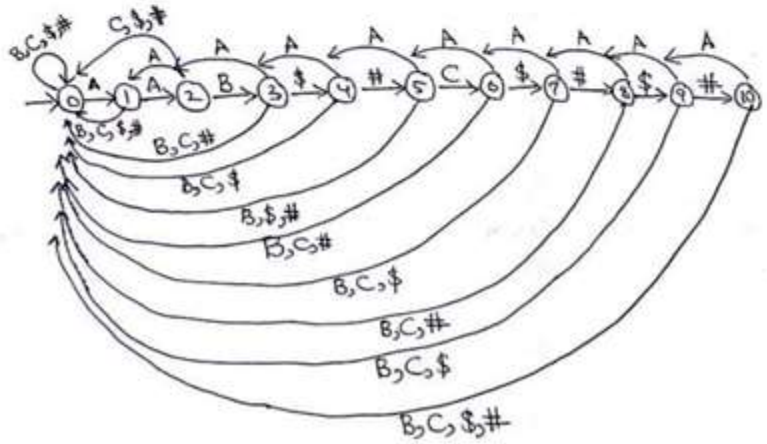
$\delta(9, AAB\$\#C\$\#\$A) = 1$ $\delta(9, AAB\$\#C\$\#\$\$) = 0$
$\delta(9, AAB\$\#C\$\#\$B) = 0$ $\delta(9, AAB\$\#C\$\#\$\#) = 10$
$\delta(9, AAB\$\#C\$\#\$C) = 0$

$\delta(10, AAB\$\#C\$\#\$\#A) = 1$ $\delta(10, AAB\$\#C\$\#\$\#\$) = 0$
$\delta(10, AAB\$\#C\$\#\$\#B) = 0$ $\delta(10, AAB\$\#C\$\#\$\#\#) = 0$
$\delta(10, AAB\$\#C\$\#\$\#C) = 0$

Stage	A	B	C	$	#
0	1	0	0	0	0
1	2	0	0	0	0
2	1	3	0	0	0
3	1	0	0	4	0
4	1	0	0	0	5
5	1	0	6	0	0
6	1	0	0	7	0
7	1	0	0	0	8
8	1	0	0	9	0
9	1	0	0	0	10
10	1	0	0	0	0

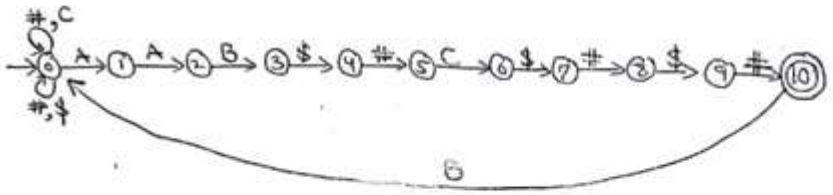
(a) P = AAB$#C$#5#

→ Finite automata of given pattern.



(b) T = ##$CAAB$#C$#S#B

Trace of the text through finite automata



The pattern is Found in the given text, because it Reach the end stage (10) in the Finite automata.

Question # 5

a) Compute the Prefix function of pattern, P = ABCACBCAB using Knuth-Morris-Pratt algorithm for String matching.

b) Dry run the Knuth-Morris-Pratt Algorithm on text, T = ABABABCACBCABCAB using the pattern P defined in part a. Show all the steps.