Question: 1 : Solution

Time complexity Analysis of Prisms Algo:-

① If Adjacency list is used to represent the graph then we can visit all the vertices in $O(V+E)$ time.

② For storing of vertices & get minimum weight edge we use min heap as a priority queue.

③ min heap operation like extracting minimum element will takes $O(\log V)$ time.

So overall time complexity:-

$= O(E+V) \cdot O(\log V)$

$= O((E+V) \log(V))$

$= O(E \log V)$ ∵ for a graph $V = O(E)$

Space complexity :- $O(V+E)$

$O(V)$ for visited Array

$O(E)$ for mean heap

Question : 2 : Solution

Time Complexity Analysis of Kruskal's Algo :-

① Sorting of edgeas takes $O(E\lg E)$ time.

② After sorting, we iterate through all the edges and apply the union-find algorithms.

③ The time complexity of union-find depends upon how it implemented.
After taking path compression and optimiszed by rank the complexity of union-find will never greater than $O(\log V)$

So overall time complexity :-

$= O(E \lg E) + O(E \lg V)$ $\left\{ \because \text{the max value of } E = O(V^2) \right.$

$= O(E \lg E)$ or $O(E \lg V)$

Question: 5: solution

Application of DFS:-

1. Detecting cycle in a graph.
2. Path finding
3. Topological Sorting
4. Finding strongly connected components of a graph.
5. ~~Idst~~ Solving puzzles with only one Solution. e.g. mazes problems.

NAME :- SHISHU Assignment 6 ①

Reg : 2020CA089 String

Question 1: Solution

Naive Algorithms :- Each character of the pattern is compared to a substring of the text which is the length of the pattern, until there is a mismatch or a matched.

Algo:

```
void SearchPat(string text, string pat) {

    n = text.size()
    m = pat.size()
    for(int i=0; i<=n-m; i++) {

        for (int j=0; j<m; j++) {

            if (text[i+j] != pat[j])
                break;

            if (i == m-1)
                print pattern match at index i.

        }

    }

}
```

Analysis :-

Since we are not performing any preprocessing therefore preprocessing time will be 0.

The best case occurs when the first character of the pattern not match

Then Best case time complexity is length of the text => O(n).

In the "worst case" all the character of text and pattern are same or last character of pattern different.

$$\boxed{T(n) = (n-m+1) \times m}$$

e.g. text(n) = A A A A A A A
pat (m) = A A A B } worst case example

Rabin - Karp :- Rabin Karp algorithms matches the hash value of the pattern with the hash value of current substring of text and if the hash values match. Then only it starts individual character.

Analysis :-

n = text.size()
m = pat.size()

First we calculate the hash value of the given pat. Hence the preprocessing time will be O(m)

During iteration from text we calculate the current hash value of substring of lenght (m) in O(1) and if hash value match with pattern hash value then we match character by character (similary as Naiv algo).

Hence in the worst case in all the time hash value equal to the pattern's hash value

$T(n) = O((n-m+1)*m)$ — all the substring hash value with pat's hash value. So we match character by character.

e.g. Text(n) = AAAAAAA
pat = AAA

Knuth-Morris-Pratt (KMP):- The KMP algorithms uses degenerating property (Pattern having some sub-patterns appearing more than once in the pattern) of the pattern and improves the worst case complexity to O(n).

The basic idea behind KMP's algorithm is: whenever we detect a mismatch (after some matches), we take advantage of this information to avoid matching the characters that we know will anyway match.

Analysis:-

KMP algorithms preprocesses pat[] and construct an auxiliary lps[] (longest substring) of size (m) which is used to skip characters while matching. Hence here we have required $\theta(m)$ time to pre calculate m lps array.

For pattern matching we iterate only forward in text and use lps for skipping backward. So the time complexity for matching will be $\theta(n)$.

Here:-

Algorithms	Preprocessing time	Matching time
Naive	0	$O((n-m+1)*m)$
Rabin-karp	$\theta(m)$	$O((n-m+1)*m)$
finite automation	$\theta(m\|\Sigma\|)$	$\theta(n)$
KMP	$\theta(m)$	$\theta(n)$

Question 2:- Solution

Whenever we get a non-matching character i.e. text[i] ≠ pat[j], then we do j=0, i.e. pat[0] match with text[i]. This works because the pattern character are all different, which mean that whenever we have a partial match there can be no other match overlapping with text.

So in this way running time complexity will be O(n).

Question 3:- Solution :

T = 3141 5926 5358 9793 (text)
q = 11 (mod)
P = 26 (Pattern)

hash value for the pattern P = P mod q

⇒ 26 % 11
= 4

Now we find the exact match of P mod q (4) in the given text of length 2.

T = $\boxed{31}$ 4 1 5 9 .....
↳ 31 and 11 = 9 ≠ 4

3 $\boxed{14}$ 1 5 9 . . .
→ 14 mod 11 = 3 ≠ 4

3 1 $\boxed{41}$ 5 9 . . .
→ 41 mod 11 = 8 ≠ 4

3 1 4 $\boxed{15}$ 9 . . .
→ 15 mod 11 = 4 = 4 i.e spurious hit - 1

3 1 4 1 $\boxed{59}$ 2 6 . . .
→ 59 mod 11 = 4 = 4 i.e Spurious hit - 2

. . . . 1 5 $\boxed{92}$ 6 . .
→ 92 mod 11 = 4 = 4 i.e Spurious hit 3

. . . . 1 5 9 $\boxed{26}$ 5 3 5 8 . . .
→ 26 mod 11 = ~~4~~ 4 ≠ 4 Exact match

. . . . 9 2 $\boxed{65}$ 3 5 8 . . .
→ 65 mod 11 = 10 ≠ 4

. . . . 2 6 $\boxed{53}$ 5 8 . .
→ 53 mod 11 = 9 ≠ 4

. . . 6 5 $\boxed{35}$ 8 . . . .
→ 35 mod 11 = 2 ≠ 4

. . . 5 3 $\boxed{58}$ 9 7 9 3 . . .
→ 58 mod 11 = 3 ≠ 4

. . . . 3 5 $\boxed{89}$ 7 . .
→ 89 mod 11 = 1 ≠ 4

. . . . 5 8 $\boxed{97}$ 9 . . .
→ 97 mod 11 = 9 ≠ 4

... 89 $\boxed{79}$ 3

79 mod 11 = 2 ≠ 4

... 97 $\boxed{93}$

93 mod 11 = 5 ≠ 4

So, we found total 3 spurious hit at the 15, 59 and 92

Question 5: solution

We can see that T is a cyclic rotation of T' of and of T is substring of (T' + T')

Now this problem reduce into third pattern matching in given text.

where { text = T' T'
pattern = T

Therefore, we can solve this using kmp algorithm in linear time.

Algo:-

```
bool iscycle (T, T'){
    String text = T' + T;
    string pat = T;
    bool res = kmp(text, pat);
    return res;
}
```

these kmp will pattern 0-1 in case of mismatch / match.

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

$T = arc$

$T' = car$

text = carcar
pat = arc

c $\boxed{arc}$ ar
→ pattern is matcted

Hence, arc is cyclic rotation of string car