3-12-20	Advanced Algorithms TEST-2	GURU NANMA 1BM18CS031 Guru Nanma

PART-C

4a) Horspool's Algorithm for string matching:

```
ShiftTable (P[0..m-1])
//Fills the shift table used by Horspool's and Boyer-Moore
//algorithms
//Input: Pattern P[0..n-1] and an alphabet of possible characters
//Output: Table[0..size-1] indexed by the alphabet's character
    //and filled with shift sizes computed by formula

    for i ← 0 to size-1
            do Table[i] ← m
    for j ← 0 to m-2
            do Table[P[j]] ← m-1-j
    return Table

HorspoolMatching (P[0..m-1], T[0..n-1])
//Implements Horspool's algorithm for string matching
//Input : Pattern P[0..m-1] and text T[0..n-1]
//Output : The index of the left end of the first matching
 // substring or -1 if there are no matches
    ShiftTable (P[0..m-1])   //generate Table of shifts
    i ← m-1                  //position of pattern's right end
    while i ≤ n-1 do
        k ← 0
```

GURU NANMA
1BM18CS031

```
while k ≤ m-1 and P[m-1-k] = T[i-k] do
        k ← k+1
    if k=m
        return i-m+1
    else i ← i + Table[T[i]]

return -1
```

Shift Table :⇒

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6

W	X	Y	Z
6	6	6	6

⇓

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
~~3~~ 0	6	6	6	6	6	6	1	6	6	6	6	6	6	6	5	6	4	2

T	U	V	W	X	Y	Z
6	6	6	6	6	6	6

$j = 0$ to $m-2$, Table[p[j]] ← $m-1-j$

Table[p[0]] = $m-j-1 = 6-0-1 = 5$
P = 5

Table[p[1]] = $m-j-1 = 6-1-1 = 4$
R = 4

Table[p[2]] = $m-j-1 = 6-2-1 = 3$
A = 3

Table[p[3]] = $m-j-1 = 6-3-1 = 2$
S = 2

Table[p[4]] = $m-j-1 = 6-4-1 = 1$
H = 1

Table[p[5]] = $m-j-1 = 6-5-1 = 0$
A = 0

Prashia

m-1-j ⇒ P
6-1-0
5
m-1-j ⇒ r
6-1-1
4
m-1-j ⇒ a
6-1-2 ⇒
3
m-1-j ⇒ s
6-1-3 ⇒

GURU NAMMA
1BM18CS031

String notching steps:

```
praveenprashanathpradan
prasha                      //not same
                            //so move 6 as
                            //e = 6
       prasha               //not same
                            //h = 1, so move 1
        prasha
```

String has been matched

PART-A

1) Time complexity of Naive string matching:

Pseudocode:

```
Naive_String_Matching(T, P)
    n = T.length  — ①
    m = P.length  — ②
    for s = 0 to n-m  — ③
        if P[1...m] == T[s+1...s+m]  — ④
            print "Patter occurs with shift" s  ⑤
```

This for loop from ③ to ⑤ executes n-m+1 times and in each iteration we are doing m comparisons so, the total time complexity is $O((n-m+1)m)$

- For each of the n-m+1 possible values of shift s, the implecit loop on line 4 to compare corresponding characters must execute m times to validate the shift

GURU NANMA
1BM18CS031

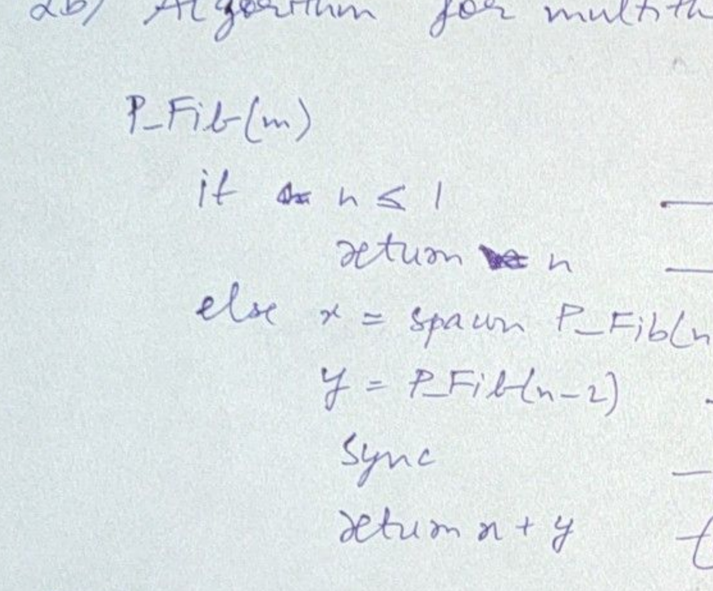
• The worst case running time is thus $O((n-m+1)m)$ which can be $O(n^2)$, if $m=[n/2]$, Because it requires no preprocessing, Naive-String-Matching running time equal its matching time

PART-B

2b) Algorithm for multithread fibonacci number generation:

```
P_Fib(n)
    if n ≤ 1                      — ①
        return n                  — ②
    else x = spawn P_Fib(n-1)     — ③
        y = P_Fib(n-2)            — ④
        sync                      — ⑤
        return x+y                — ⑥
```

Considering P_Fib(4):-



• Each circle represents one strand

• Black dots: base case or part of the procedure up to the spawn of P_FIN(n-1) in line ③

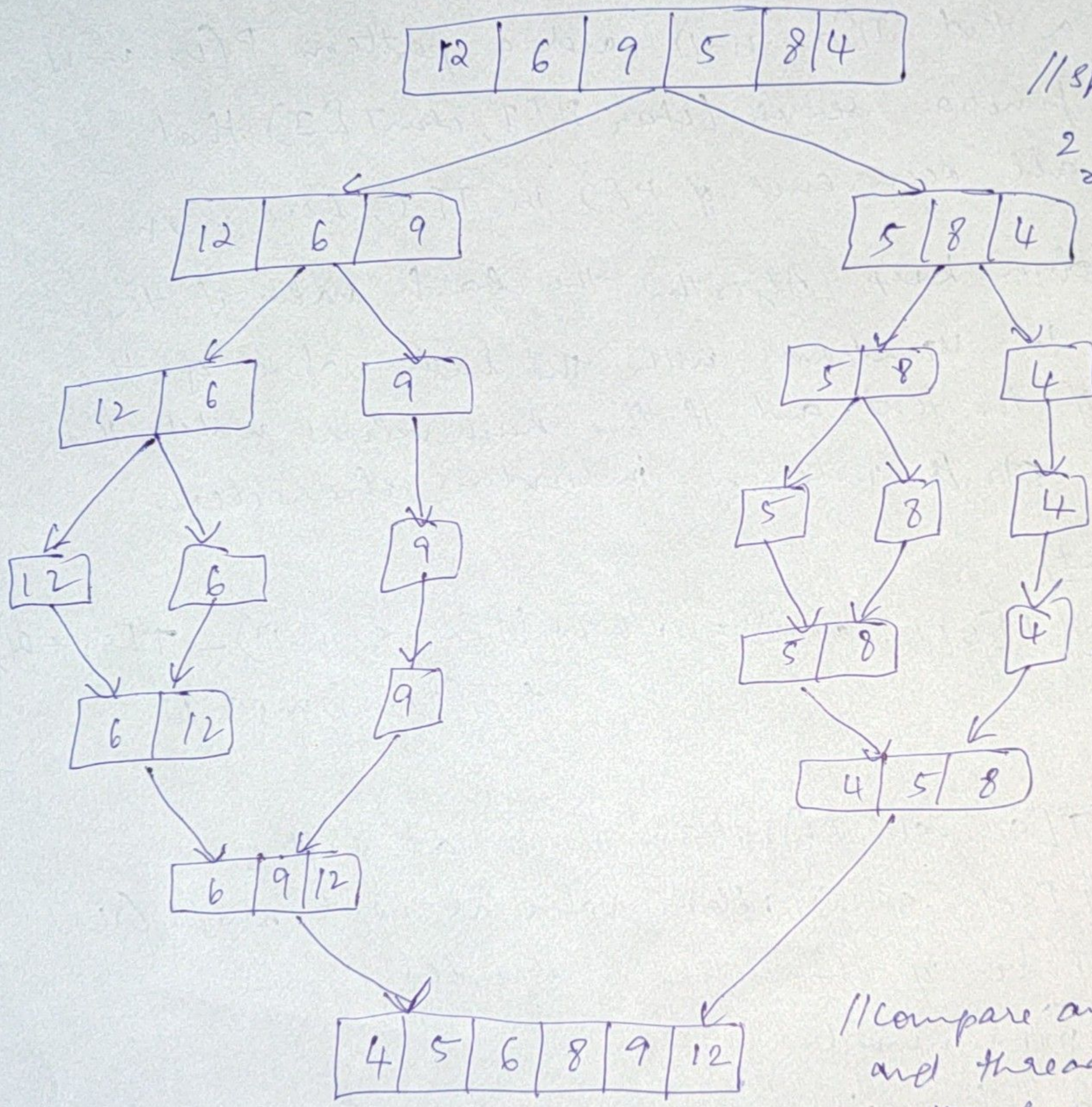
• Grey dots: regular execution i.e. the part of the procedure that calls P_FIN(n-2) in line ④ up to the sync in line ⑤

• White dots: part of the procedure after sync up to the where it returns the result.

GURU NANMA
1BM18CS031

PART-B

2c)



//split to 2

2 threads run at a time

// we divide it further

//compare and ~~merge~~ merge and threads are synced

//compare and ~~se~~ merge and threads are ~~sty~~ synched

GURU NANMA
1BM18CS031

PART – C

3a) Robin_Karp Algorithm

Given a text T[0 … n−1] and a pattern P[0 … m−1], write a function search (char P[], char T[]), that prints all occurence of P[] in T[], here n>m

In Robin karp Algorithm, the hash value of the substring is matched with the hash value of the substring of the text and if the hash values match. then only it starts matching individual characters.

<u>Important</u>

$$hash(T[s+1 \dots s+m]) = d(hash(T[s \dots s+m-1]) - T[s]*h) + T[s+m]) \bmod q$$

here

$hash(T[s \dots s+m+1])$: Hash value at shift s

$hash(T[s+1 \dots s+m])$: Hash value at next shift (s+1)

d : Number of characters in Alphabet

q : prime number

$k = d \wedge (m-1)$

<u>Algorithm</u>:

```
Robin_karp_Match (T, P, d, q)
        n = length(T);
        m = length(P);
        h = $d^{m-1}$ mod q;
        p = 0;  $t_0$ = 0;
```

GURU NANMA
1BM18CS031

```
for i=1 through m do
        p = (d * q ~~*~~ + P[i]) mod q;
        to = (d * to + T[i]) mod q;
end for;
for s=0 through (n-m) do
        if (p = = ts) then
            if (P[1 ... m] = = T[s+1 .... s+m]) then
                print the shift value as s;
        if (S < n-m) then
            $t_{s+1}$ (d (t_s - h * T[s+1] + T[s+m+1]) mod q;
        end ~~for~~ if;
    end for;
End Algorithm
```

For pattern = "baab"

Text = "ababaabbab"   n = 10

1 2 3 4 5 6 7 8 9 10

Σ = {a,b} when   consider Value of a = 0

* · of b = 2

Considering q = 13

Pattern = baab = 2+1+1+2 = 6

m = 3

Pattern at index 4

P = (d * p + pat[i] mod q)

t = (d * t + pat[i] mod q)

Guru Nanna
1BM18CS031

$i = 0$

$p = 0$
$t = 0$

$p = (0+1) \bmod 11 = 1$

$t = (0+0) \bmod 11 = 0$

$p = 1, \ t = 0$

$i = 1$

$p = (26 + 0) \bmod 11 = 4$
$t = (0+1) \bmod 11 = 1$

$p = 4, \ t = 1$

$i = 2$

$p = (26 * 4 + 0) \bmod 11 = 5$
$t = (26 + 0) \bmod 11 = 4$

$t = 4$
$p = 5, \ t = 4$

$i = 3$

$p = (d * p + pat[i]) \bmod q$
$= (26 * 5 + 1) \bmod 11 = 9$
$t = (26 * 4 + 1) \bmod 11 = 6$

$p = 6 \quad t = 6$

$i = 2$

$T =$ "aba$\boxed{\text{baab}}$bab"

$t = 6$
$p = 6$

String matched

GURU NANMA
1BM18CS031

a b a b a a b b a b

$\overset{a}{b}\ \ \overset{b}{a}\ \ \overset{a}{a}\ \ \overset{b}{b}$ ×

$\overset{b}{b}\ \ \overset{a}{a}\ \ \overset{b}{a}\ \ \overset{a}{b}$ ×

$\overset{a}{b}\ \ \overset{b}{a}\ \ \overset{a}{a}\ \ \overset{a}{b}$ ×

$\overset{b}{b}\ \ \overset{a}{a}\ \ \overset{a}{a}\ \ \overset{b}{b}$ ✓

String Matched.

PART – B

2a) For multithreaded matrix multiplication:
time complexity

(1) partition — $O(1)$
(2) 8 requirements of $n/2 \times n/2$
(3) Adding — $O(n^2)$

Speed Up
T_1/T_p

∴ Total time complexity

$$T_1(n) = 8 \times T_1(n/2) + \Theta(n^2) = \Theta(n^3)$$

$T_p(n) \approx T_1/p$

$T_p(n) = O(n^3/p$
$= O(n^3/p)$

$P << (T_1 / T_\infty)$
$T_\infty = O((\log n)^2)$

$P = O(n^3)/O((\log n)^2)$

$T_p(n) = O(n^3)/p$

$T_p(n) = O(n^3)/P$ (Substitute p from 1)

$T_p(n) = O[(\log n)^2]$ → Speed up $= T_1/T_P = \frac{O(n^3)}{O((\log n)^2)}$ //

(9)