classmate

Date

Page

CS203 : Assignment

2.

(a) $f(n) = n - 100$

$g(n) = n - 200$

Clearly, both $f(n)$ and $g(n)$ are $O(n)$.

$\therefore \boxed{f(n) = \Theta(g(n))}$

(b) $f(n) = 100n + \log n$

$g(n) = n + (\log n)^2$

Since, $\log n$ is clearly asymptotically smaller than n.

$\therefore f(n) = O(n)$

Also, $(\log n)^2$ is asymptotically smaller than n.

$\{$ check for $n = 1024$, $\log n = 10$, $(\log n)^2 = 100 \}$

$\therefore g(n) = \Theta(n)$

$\boxed{\therefore f = \Theta(g(n))}$

(c) $f(n) = n^{1.5}$
$g(n) = n \log^2 n$

If we divide both functions by $\cancel{\ }$ $\sqrt{n}$

i.e. $\frac{f(n)}{\sqrt{n}} = \sqrt{n}$

$\frac{g(n)}{\sqrt{n}} = \cancel{\ } \log^2 n$

Since, $\log^2 n \geq \sqrt{n}$ asymptotically.

$\therefore \boxed{f(n) = \Omega(g(n))}$

(d) $f(n) = \log(2n)$
$g(n) = \log(3n)$

$f(n) = \log(2) + \log(n)$
$g(n) = \log(3) + \log(n)$

$\therefore f(n) = g(n) = O(\log n)$

$\boxed{\therefore f(n) = \Theta(g(n))}$

(e) $f(n) = n!$, $g(n) = 2^n$

Since, both $n!$ and 2^n are non-decreasing functions, and for $n = 4$,
$f(n) > g(n)$

$\therefore \quad \boxed{f(n) = \Omega(g(n))}$

3. (a) $T(n) = 4T(n/2) + n^2 \log n$

Using Master's Theorem,

$a = 4$

$b = 2$

$k = 2$

$p = 1$

$\log_2 4 = 2$

$\because \log_b a = k$ and $p > -1$

$\therefore \boxed{T(n) = O(n^2 \log^2 n)}$

(b) $T(n) = 8T(n/2) + n \log n$

Using Master's Theorem

$a = 8, b = 2, \log_b a = 3$

$k = 1, p = 1$

$\because \log_b a > k,$

$\boxed{\therefore T(n) = O(n^3)}$

(c) $T(n) = 2T(\sqrt{n}) + n$

let $n = 2^k$

$T(2^k) = 2T(2^{k/2}) + 2^k$

let, $S(k) = T(2^k)$

$S(k) = 2S(k/2) + 2^k$

$S(k) = 2(2S(k/4) + 2^{k/2}) + 2^k$

$\vdots$

$S(k) = 2^m S(k/2^m)\, 2^{k/2^m} + 2^{m-1} \cdot 2^{k/m-1} + \dots + 2^m$

Approximating above expression,

$S(k) \le 2^m S(k/2^m)\, 2^k + 2^{m-1} 2^k + 2^{m-2} 2^k + \dots + 2^k$

$S(k) \le 2^m S(1)\, 2^k + 2^k(2^m - 1)$

Since, $T(1) = 1$, so $S(1) = T(4) = 4$

$S(k) \le 4k\,2^k + 2^k(k-1)$

$S(k) \le O(k\,2^k)$

$T(2^k) = O(k\,2^k)$

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

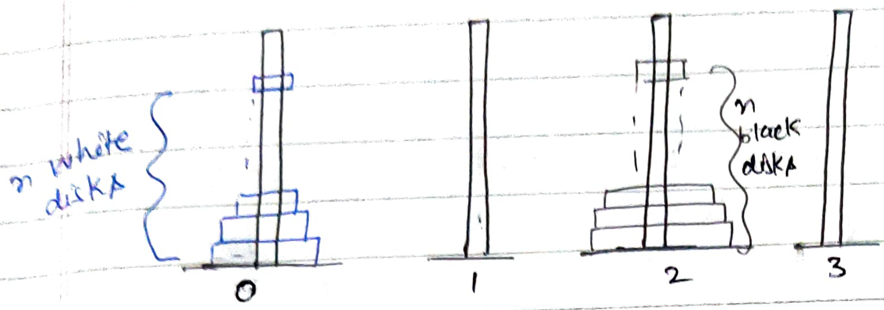
4. The above problem is m colouring problem. We can solve this problem using backtracking. The algorithm for the same ~~as~~ is as follows:

```
graph coloring (G, m, color[], V) {
        if all vertices colored, return true;
        for all possible colours {
              if colour is valid {
                   if (graph colouring (G, m, color, V+1 == true)) {
                         return true;
                   }
              colour [V] = 0;
        }
        return false;
}
```

The time complexity for above algorithm is $O(m^V)$ and space complexity is $O(V)$.

∴ Since, only two colours are given, therefore time complexity is $O(2^V)$ and space complexity is $O(V)$.

5 (a).



Since, we have to interchange stack at '0' and stack at '2' using pole 1 and 3 as temporary holding towers. It can be done in the following manner:

(i) Move white disk stack to pole 1 from pole 0.
(ii) Then, move black disk stack to pole 0 from pole 2.
(iii) Then, move ~~the~~ white disk stack to pole 2 from pole 1.

Since, the standard tower of hanoi problem requires $(2^n - 1)$ moves for n disks. and here we are just using this algorithm thrice.
Therefore, the least number of moves required will be

$$\boxed{3(2^n - 1)}$$

The standard algorithm for TOH problem is:

```
TOH(n, A, B, C){

    if (n==1)  L <- <A,B>;
    else {
        L1 <- TOH(n-1, A, C, B);
        L2 <- TOH(1, A, B, C);
        L3 <- TOH(n-1, C, B, A);
        L = L1 + L2 + L3;
    }
    return (L);
}
```

The above algorithm is used to find the moves to move n disks from pole A to pole B via pole C. following the rules of TOH problem.

This algorithm can be used for our problem in the following way:

```
interchange Stacks (n) {

    TOH(n, 0, 1, 3);
    TOH(n, 2, 0, 3);
    TOH(n, 1, 2, 3);

}
```

5(b) Now, we have 2 disks of each size. Therefore, 2n disks on pole 0 and we need to move them on pole 2 using pole 1.

Think of the standard algorithm used to solve TOH problem. We ~~need to~~ only need to modify a small thing that is explained below.
Everytime, we make a move, we just need to duplicate that move so that the disk which has same size always stick together. Therefore, for each move in standard algorithm, we need two (~~but~~ not one) move to solve our problem.

∴ Least Number of moves required to solve this problem will be

$$\boxed{2(2^n - 1)}$$

The following algorithm can be used to solve this problem:

```
TOH (2n, 0, 2, 1) {
    if (n==2) {
        L ← <0,2> + <0,2>
    } else {
        L1 ← TOH(2n-2, 0, 1, 2);
        L2 ← TOH(2, 0, 2, 1);
        L3 ← TOH(2n-2, 1, 2, 0);
        L ← L1+L2+L3;
    }
```

}
return L;
}

5. Basically, we can consider disks of same size as a single disk that requires 2 moves to shift from one pole to other.

6. We can use binary search to find the maximum element. The algorithm is given below;

→ The maximum element is the only element whose next is smaller than it. If there is no next smaller element, then there is no rotation. Check this condition for middle element by comparing it with elements at mid-1 and mid+1.

→ If maximum element is not at middle, then it lies either in left or right half
- If middle element is greater than the last element, then the maximum element ~~is greater~~ lies in left half.
- Else it lies in right half.

the pseudocode for the above approach is given below;

```
maxElement (arr[], low, high){

    if (high == low){
        return arr[low];
    }

    int mid = low + (high-low)/2;

    if (mid=0 && arr[mid] > arr[mid+1]){
        return arr[mid];
    }

    if (arr[low] > arr[mid]){
        return findMax (arr, low, mid-1);
    } else {
        return findMax (arr, mid+1, high);
    }

}
```