GROUP ASSIGNMENT-1

GROUP-NO: 1

CS22B2012, CS22B2013, CS22B2014, CS22B2020, CS22B2023

1) Given an Integer, the objective is to find all the prime factors. Present an algorithm and its step count analysis analysis. Also asymptotic analysis using O, Ω, Θ notation

Algorithm

1) Intialize an empty vector : This vector is used to store all the prime factors.
2) while the integer is divisible by 2, repeatedly divide it by 2. at each step add 2 to vector
3) Check for odd divisors: start checking the divisibility from 3 & proceed with odd numbers for ~~check~~ each odd number check if integer is divisible by odd number
   ⇒ if yes add the current odd number to list of Prime factors and update the integer by dividing it by the current odd
   ⇒ if No then move on to next Odd number.
4) After dividing by all possible prime factors if the remaining is greater than 2, it is a prime factor itself. Add this remaining integer
5) return the vector.

Step count analysis:-

divide by 2:- This step takes O(logn) steps in worst case, where n is the given integer. As dividing by 2 reduces the no. of digits in binary representation by 1

Checking odd divisors:- Checking divisibility for odd numbers from 3 to sqrt(n) takes O(sqrt(n)). [sqrt = square root]
This is because we are iterating numbers upto Square root of n

Checking remaing Integer:- This step takes O(1) time

Total step Count = O(logn) + O(sqrt(n)) + O(1)

Asymtotic Analysis

⇒ O notation:-
For larger values of n sqrt(n) dominates Log(n) hence, the time camplexity is O(sqrt(n))

⇒ Ω notation:- The best case is when input is small Prime number (ex:- 2) then the Algorithm takes Const time Hence Ω(1)

⇒ Θ notation:- The tightest bound on the algorithm's time complexity is Θ(sqrt(n)) since it lies b/w lower bound of Ω(1) & upper bound of O(sqrt(n))

(2) a) $2n^3 + 40n - 415$

$\rightarrow O(n^3)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow \Omega(n^3)$ $\exists c>0, \forall n > 4, n_0 > 0$

$\rightarrow \theta(n^3)$ $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$

$\rightarrow o(n^4)$ $\forall c>0, \exists n_0 > 0, \forall n \geq n_0$

$\rightarrow \omega(n^2)$ $\forall c>0, n_0 > 4 \ \forall n \geq n_0$

b) $2^n + n^2 . 1.5^n + 100^n$

$\rightarrow O(100^n)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow \Omega(100^n)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow \theta(100^n)$ $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$

$\rightarrow o(n^2 . 1.5^n)$ $\forall c>0, \exists n_0 > 0, \forall n \geq n_0$

$\rightarrow \omega(2^n)$ $\forall c>0, \exists n_0 > 0 \ \forall n \geq n_0$

c) $n^k + 2^n + 4^n$

$\rightarrow O(4^n)$ $\exists c>0, \forall n \geq n_0, n_0 > 0$

$\rightarrow \Omega(4^n)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow \theta(4^n)$ $f(n) = O g(n) \wedge f(n) = \Omega g(n)$

$\rightarrow o(n^n)$ $\forall c>0, \exists n_0 > 0, \forall n \geq n_0$

$\rightarrow \omega(n^k)$ $\forall c>0, \exists n_0 > 0, \forall n \geq n_0$

d) $\dfrac{n^k + c^n}{}$

$\rightarrow O(n^k)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow \Omega(c^n)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow \omega(n)$ $\forall c, \exists n_0 > 0, \forall n \geq n_0$

$\rightarrow o(n^n)$ $\forall c, \exists n_0 > 0, \forall n \geq n_0$

e) $n^2 + \frac{1}{n^2}$

$\rightarrow O(n^2)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow \Omega(n^2)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow \Theta(n^2)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow o(n^3)$ $\forall c, \exists n_0 > 0, \forall n \geq n_0$

$\rightarrow \omega(n)$ $\forall c, \exists n_0 > 0, \forall n > n_0$

f) $5 + 1/n$

$\rightarrow O(1)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow \Omega(1)$ $\exists c>0, \forall n > n_0, n_0 > 0$

$\rightarrow \Theta(1)$ $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$

$\rightarrow \omega(1)$ $\forall c, \exists n_0 > 0, \forall n > n_0$

$\rightarrow o(1)$ $\forall c, \exists n_0 > 0, \forall n > n_0$

g) 50

$\rightarrow O(1)$ $\exists\ c > 0,\ \forall n > n_0,\ n_0 > 0$

$\rightarrow \Omega(1)$ $\exists\ c > 0,\ \forall n > 4,\ n_0 > 0$

$\rightarrow \Theta(1)$ $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$

$\rightarrow o(1)$ $\forall c > 0,\ \exists\ n_0 > 0,\ \forall n \geq n_0$

$\rightarrow \omega(1)$ $\forall\ c > 0,\ n_0 > 0\ \ \forall n \geq n_0$

Q3.

Algorithm for Recursive Addition

```
Algo addRecursive(a, b)
{
   if (b==0)   return a;

   if (b>0)
   {  return addRecursive(a+1, b-1); }

   if (b<0)
   {     return addRecursive(a-1, b+1); }
}
```

Addition of A+B is same as adding 1 to A 'B' times.

Regardless of the sign of A we can only either add +B (or) add -'B' to it depending on input.

For $b>0$

$$T(a,b) = T(a+1, b-1) + 1$$

$$T(a+1,b-1) = T(a+2, b-2) + 1$$

$$T(a+2,b-2) = T(a+3, b-3) + 1$$

$\vdots$

$$T(a+b-1, 1) = T(a+b, 0) + 1$$

b-1 steps

$$T(a,b) = \underline{T(a+b, 0)} + b-1$$

only one comparision takes place

$$\therefore T(a,b) = 1+b-1 = b$$

For $b<0$

$$T(a,b) = T(a-1, b+1) + 1$$

$$T(a-1,b+1) = T(a-2, b+2) + 1$$

$$T(a-2,b+2) = T(a-3, b+3) + 1$$

$\vdots$

Steps

$\cancel{T(a-b+0) = T(a}$

$$T(a-b-1, 1) = \underline{T(a-b, 0)} + 1$$

only one comparision takes place here

$\cancel{T(a-b-1}$

$$T(a,b) = T(a-b, 0) + b - 1$$

$$T(a,b) = T(a-b, 0) + |b| - 1$$

$$T(a,b) = 1 + |b| - 1 = |b|$$

$$\therefore T(a,b) = O(\max(|b|,1))$$

Algo for multiply recursively

```
multiplyRecursive (a, b)
{
    if (a==0 || b==0)
    { return 0; }

    if (b==1)
    { return a; }

    return a+ multiplyRecursive (a, b-1);
}

Algo multiply (int a, int b)
{
    int sign = 1;
    if ((a<0 && b>0) || (a>0 && b<0)) { sign=-1; }

   // if (a<0 && b<0) {sign== 1;}

    a=abs(a);
    b=abs(b);

    return sign * multiplyRecursive(a, b);
}
```

Mulipication is nothing but repeated addition.
We repeatedly add a to itself 'b-1' times ($\because a + a(b-1) = a \cdot b$)

Time Complexity of Algo Multiply is $T(n) = f(n)$, $f(n) \leq C(1)$
$\Rightarrow T(n) = O(1)$

Until b= abs(b) then recursive call time complexity has to be added

Recursive division:


```
divisionRecursive ( a, b)
{
  static int count = 0;
  if (a>=b)
  {
    count++;
    return divisionRecursive (a-b, b);
  }
  else if ( b==0) { cout << "ZeroDivisionError" <<endl; }
  else { Return count ; }
}

Algo Division
{
    int sign = = 1;
    if ((a<0 && b>0) || (a>0 && b<0)) { sign = -1 ; }
  // if (a<0 && b<0) { sign = 1; }
  a = abs(a);
  b = abs(b);
  Return sign * divisionRecursive (a, b);
}
```

Division is nothing but repeated subtraction of b from a. The number of times we are able to subtract b from a will be our quotient.

Time Complexity of Algo Division is $T(n) = f(n)$, $f(n) \leq C(1)$

$\Rightarrow T(n) = O(1)$

Until b = abs(b) then recursive call time Complexity has to be added.

Recursive Multiplication

$b \le 0$

$\cancel{T(a,b) = 1}$

$$\underline{b>0}$$

$$T(a,b) = T(a, b-1) + 2$$

$$T(a,b-1) = T(a, b-2) + 2$$

$$T(a, b-2) = T(a, b-3) + 2$$

$$\vdots$$

$$T(a, b-(b-2)) = T(a,1) + 2$$

$\Downarrow$ one comp takes place here

$$T(a,b) = T(a,1) + 2(b-1)$$
$$= 1 + 2b - 2$$
$$= 2b - 1$$

$$\Rightarrow \quad T(a,b) = \Theta(\max(2|b| \mp 1, 1))$$
$$= \Theta(\max(|b|, 1))$$

Recursive Division

$$T(a,b) = T(a-b, b) + 2$$

$$T(a-b, b) = T(a-2b, b) + 2$$

$$\vdots$$

$$T(a-kb, b) = T(a-(k+1)b, b) + 2$$

$\Downarrow$ Assuming that $a - kb < b$ termination condition

$\Downarrow$ one cmp takes place

$\cancel{T(a-kb, b) = T(}$

$$T(a,b) = T(a-(k+1)b, b) + (k+1)2$$

$$T(a,b) = 2k+3 \Rightarrow \Theta(2k+3)$$
$$= \Theta(a/b)$$

4 Q) identify $f(n)$ & $g(n)$ such that $f(n) \neq O(g(n))$ &
$f(n) \neq \Omega(g(n))$

let us take $f(n) = n^{1+\sin(n)}$ & $g(n) = n^2$

$f(n) = n^{1+\sin(n)}$ as the $\sin(n)$ oscillates b/w -1 to 1

as n increases. it never settles in to a specific

growth rate making it hard to find c & n_0 which

are positive to define $f(n) = O(g(n))$ or $f(n) = \Omega(g(n))$

$f(n) \neq O(g(n))$ as there's no constant c & ~~ko~~ n_0 such that

$|n^{1+\sin(n)}| \leq c^* n^2$ for all $n \geq n_0$.

$f(n) \neq \Omega(g(n))$ as there's no constant c & n_0 such that

$|n^{1+\sin(n)}| \geq c^* n^2$ for all $n \geq n_0$.

Q5)

```
n, r, a[100]
Scanf(n)   // Number in decimal
Scanf(r)   // Base
i=0
while(n!=0)
{   int d= n%r;      // To get the remainder if
    a[i++]=d;        //   the decimal number is
    n=n/r;           //   Devided by the base
}
Output=0
for(int j=0; j<i; j++)
{
  output= output+ a[j] * pow(10, j)
}
print( output)
```

Time Complexity Analysis

Best case: When n=0 while loop won't be entered and since i=0 forloop won't be entered. So $\Omega(1)$

Worst case: when n>0 while loop is entered and at each iteration inside while loop 3 statements are done at each step. we can see that every time n=n/r the value of n decreases by r times so the while loop runs for $3\lceil \log_r n \rceil$ and for loop also runs for $\lceil \log_r n \rceil$ due to $i = \lceil \log_r n \rceil$ at the end of while loop

So worst case is $O(\lceil \log_r n \rceil)$

Average case is also same as worst case so

$\Theta(\lceil \log_r n \rceil)$

6Q) Arrange the following in non-decreasing order of its asymptotic growth. $n^3, 4^{\log_2 n}, (1.5)^n, 2^{n\log_2 n}, n^n, n^{100}$.

Ans) $4^{\log_2 n}$ can also be written as $n^{\log_2 4} = \underline{\underline{n^2}}$

$2^{n\log_2 n}$ can also be written as $2^{\log_2 n^n} = n^n$

So, the order (non-decreasing): $n^n, 2^{n\log_2 n}$, ~~[illegible]~~, $(1.5)^n, n^{100}$, $n^3, 4^{\log_2 n}$

7Q) A program has three-modules: time complexities are $O(n^2), \Omega(n^2), \Theta(n^2)$. What is the overall time complexity. Present O, Ω, Θ if exists.

Ans) Program: P_1:

$M_1 \rightarrow O(n^2)$ at most n^2

$M_2 \rightarrow \Omega(n^2)$ at least n^2 $\Rightarrow$ this module would be deciding / playing major role in computing the time complexity of the program.

$M_2 \longrightarrow \Theta(n^2)$ avg n^2

$\Rightarrow M_2 \rightarrow$ Lower bound is given; so upper bound can be anything. $\Rightarrow$ time-complexity of program: $\Omega(n^2)$

) Arrange the following in increasing order: $n^{O(1)}, n^{\Omega(1)}, n^{\Theta(1)}$

) $\Rightarrow$ for $2^n = \Omega(1)$, $n^{10000} = \Omega(1)$ etc...

$\Omega(1)$ is Lower bound & upper bound can be anything;

$\Rightarrow \boxed{n^{O(1)}, n^{\Theta(1)}, n^{\Omega(1)}}$