Date:

Assignment 1

① Asymptotic Notation is used to describe the running time of an algorithm - how much time an algorithm takes with a input n. There are three different notations: big O, big Theta $\Theta(O)$, and big Omega (Ω)

→ Big-O Notation :- it describe the worst-case running time of a program. we count the Big O of an algorithm by counting how many iteration an algorithm will take in the worst-case scenario with an input of N.

→ Big-Ω Notation :- it describe the best running time of a program. we compute the big-Ω counting how many iterations an algorithm will take in the best-case scenario based on an input of N.

Big Theta (Θ) Notation:- theata notation encloses the function from above and below. Since it represents the upper and the lower ~~bound~~ bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.

2)

Time complexity of

for (i=1 to n) ⟶ n

{

i = i * 2 ;

}

if

i = 1, 2, 4, 8, ... n

n = K-1

$n = 1.2^{K-1}$

$n = \frac{2^K}{2}$

$2n = 2^K$

$\log_2 2n = \log_2 2^K$

$K = 1 + \log_2 n$

Time complexity = $O(\log_2 n)$

3)

$T(n) = 3(T(n-1))$

$T(n) = 3T(n-1)$, $n > 0$

$= 1$, otherwise

$T(0) = 1$

$T(n) = 3T(n-1) \quad -(i)$

put $n = n-1$

$T(n-1) = 3T(n-1-1)$

$T(n-1) = 3T(n-2) \quad -(ii)$

put (ii) in (i)

$T(n) = 3 \cdot 3T(n-2) \quad -(iii)$

put $n = n-2$

$T(n-2) = 3T(n-2-1)$

$T(n-2) = 3T(n-3)$

put in (iii)

$T(n) = 3 \cdot 3 \cdot 3T(n-3)$

$= 3^k T(n-k)$

put $n-k = 1$

$k = n-1$

$T(n) = 3^{n-1} T(n-n-1)$

$= \frac{3^n}{3} \cdot T(1)$

$= \frac{1}{3} \cdot 3^n$

$T(n) = O(3^n)$

4. $T(n) = \begin{cases} 2T(n-1) - 1 & ; n > 0 \\ 1 & ; \text{otherwise} \end{cases}$

$T(0) = 1$

$T(n) = 2T(n-1) - 1$ — (i)

put $n = n-1$ in (i)

$T(n-1) = 2T(n-1-1) - 1$

$T(n-1) = 2T(n-2) - 1$ — (ii)

put in eq. (i)

$T(n) = 2 \cdot 2T(n-2) - 1 - 1$ — (iii)

put $n = n-2$

$T(n-2) \cdot 2T(n-2-1) - 1$

$T(n-2) = 2T(n-3) - 1$

put in eq (iii)

$$T(n) = 2.2.2T(n-3) - 1 - 1 - 1$$

$$T(n) = 2^k T(n-k) - k$$

put $n-k = 1$

$$k = n-1$$

$$T(n) = 2^{n-1} T(n-n+1) - n - 1$$

$$= \frac{1}{2} \cdot 2^n T(1) - n + 1$$

$$= O(2^n)$$

6 void function(int n)
{
int i, count = 0;
for(i=1; i*i<=n; i++)
{
count++;
}

if n=5, i=1,2
if n=10, i=1,2,3
if n=20, i=1,2,3,4

so loop runs $\sqrt{n}$ for all cases
$i = 1, 2, 3, 4, \ldots, \sqrt{n}$
$O(\sqrt{n})$

5 int i=1, s=1;
while (s<=n)
{
i++;
s=s+i;
printf("#");
}

$i = 2, 3, 4, 5, 6, \ldots k$
$s = 1 + 2 + 3 + 4 + \ldots n$

$T = a + (n-1)d$
$= 1 + (n-1)$
$T \rightarrow O(n)$