GUJARAT TECHNOLOGICAL UNIVERSITY

BE-5 SEMESTER – OLD PAPER – S22 TO W25 – QUESTION BANK ANSWER

Subject Name & Code:

ANALYSIS AND DESIGN OF ALGORITHMS (3150703)

Unit 2: Analysis of Algorithms & Asymptotic Notations

Repeated Question 1 – Define Big-O, Big-Ω, and Big-Θ notations with graphs. (4 marks)
Appeared in: S25 (Q1b, 04 marks), S24 (Q1b, 04 marks), S23 (Q1a, 03 marks)
Highest marks: 4
Ans:

Notation	Name	Definition (Mathematical)	Graph Interpretation
O(g(n))	Big-Oh (Upper bound)	∃ c>0, n_0 such that $0 \le f(n) \le c \cdot g(n)$ ∀ $n \ge n_0$	f(n) lies **on or below** c·g(n) after n_0
Ω(g(n))	Big-Omega (Lower bound)	∃ c>0, n_0 such that $0 \le c \cdot g(n) \le f(n)$ ∀ $n \ge n_0$	f(n) lies **on or above** c·g(n) after n_0
Θ(g(n))	Theta (Tight bound)	∃ c_1, c_2>0, n_0 such that $c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$ ∀ $n \ge n_0$	f(n) is **sandwiched** between two multiples of g(n)

Graph description (text for drawing):
[DG PROMPT]
Title: Asymptotic Notations (Big-O, Ω, Θ)
Description:

- X-axis: n (input size), Y-axis: f(n) (running time).
- Draw a smooth increasing curve labeled **f(n)**.
- Draw another curve labeled **c·g(n)** starting above f(n) after intersection point n_0 – this shows Big-O.
- Draw a third curve labeled **c·g(n)** but starting below f(n) after n_0 – this shows Big-Ω.
- For Θ, draw two curves **c_1·g(n)** and **c_2·g(n)** such that f(n) stays between them after n_0.
- Label the point **n_0** on X-axis where bounds become valid.

Real-world application: Big-O guarantees worst-case response time (e.g., database query). Ω gives best-case. Θ gives exact asymptotic class (e.g., merge sort = Θ(n log n)).

Other Important Question 1 – Find Big-O notation of given functions. (3 marks)
Appeared in: S25 (Q1a, 03 marks)
Given:

$$f(n) = 3n^6 + 2n \cdot \lg n + 6n$$
$$h(n) = 3n^2 + 6n \cdot \lg n + 2n^{2.5}$$

Ans:
For f(n):

- Dominant term: $3n^6$ (highest power)
- $2n\lg\ n$ and $6n$ are lower order
- Therefore, $f(n) = O(n^6)$

Verification: For large n, $3n^6 \geq 2n\lg\ n + 6n$. Choose c = 4, $n_0 = 1 \rightarrow 0 \leq f(n) \leq 4n^6$.
For h(n):

- Compare $2n^{2.5}$ vs $3n^2$: $n^{2.5} = n^2 \cdot n^{0.5}$, so $n^{2.5}$ grows faster than n^2.
- $n\lg\ n$ is lower than $n^{2.5}$.
- Dominant term: $2n^{2.5}$
- Therefore, $h(n) = O(n^{2.5})$ or more precisely $O(n^{2.5})$.
- Since $n^{2.5}$ is between n^2 and n^3, we can also write $O(n^{2.5})$.

Final Answer:

$$f(n) = O(n^6)$$
$$h(n) = O(n^{2.5})$$

Q2 – Find Big-O / Ω notation for given functions. (3 marks)
⚠️ **Incomplete question:** No functions provided.
Example format (if functions were given):
Function: f(n) = $5n^3 + 2n^2 + 10$

- Big-O: $O(n^3)$ (upper bound)
- Big-Omega: $\Omega(n^3)$ (lower bound)
- Theta: $\Theta(n^3)$ (tight bound)

Function: f(n) = n log n + 100n

- Big-O: O(n log n)
- Big-Omega: Ω(n log n)
- Theta: Θ(n log n)

Please supply the specific functions for a complete answer.

Q3 – Explain asymptotic notations with inequalities and graphs. (4 marks)
Ans:
Asymptotic notations describe growth rates of functions as input size n → ∞. They ignore constants and lower-order terms.
Notations, inequalities, and graph characteristics:

Notation	Inequality (∃ c, n_0)	Graph meaning	Example
O(g(n))	$0 \leq f(n) \leq c \cdot g(n)$ ∀ n ≥ n_0	f(n) stays **below** or on c·g(n)	2n+3 = O(n)
Ω(g(n))	$0 \leq c \cdot g(n) \leq f(n)$ ∀ n ≥ n_0	f(n) stays **above** or on c·g(n)	$n^2 = \Omega(n)$
Θ(g(n))	$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$	f(n) is **sandwiched** between two multiples	$5n^2 = \Theta(n^2)$
o(g(n))	∀ c>0, $0 \leq f(n) < c \cdot g(n)$	f(n) becomes arbitrarily small relative to g(n)	n = o(n log n)

Notation	Inequality (∃ c, n_0)	Graph meaning	Example
ω(g(n))	∀ c>0, 0 ≤ c·g(n) < f(n)	f(n) grows strictly faster than any multiple	n log n = ω(n)

Graph description (for visualization):

- X-axis: n (input size), Y-axis: running time.
- For **Big-O**: f(n) lies on or below c·g(n) after n_0.
- For **Big-Omega**: f(n) lies on or above c·g(n) after n_0.
- For **Theta**: f(n) lies between c_1·g(n) and c_2·g(n).

Real-world application: Big-O is used to guarantee worst-case performance (e.g., sorting algorithms in library functions).

Q4 – Prove that log(√n) = O(log n). (3 marks)
Ans:
We need to show ∃ constants c > 0 and n_0 such that
0 ≤ log(√n) ≤ c·log n for all n ≥ n_0.
Proof:

1. Simplify left side:
   log(√n) = log(n^(1/2)) = (1/2) · log n
2. Choose c = 1/2 (or any c ≥ 1/2).
   Then: (1/2) log n ≤ (1/2) log n (equality holds)
3. More formally: For c = 1, n_0 = 1:
   log(√n) = 0.5 log n ≤ 1·log n for all n ≥ 1.
4. Therefore, by definition, **log(√n) = O(log n)**.

Alternative: Any c ≥ 0.5 works. The statement is also Θ(log n) because log(√n) = 0.5 log n is just a constant factor.

Q6 – Write an algorithm with time complexity O(1) that includes a loop. Confirm using tabular method. (3 marks)
Appeared in: S25 (Q3b, 03 marks)
Ans:
An O(1) algorithm with a loop is possible only if the **number of loop iterations is constant** (does not depend on input size n).
Algorithm (Constant-time loop):

```
Algorithm PrintFirstFive(arr):
  Input: An array arr (size ≥ 5, but n can be larger)
  Output: Prints first 5 elements

  for i = 1 to 5 do
    print(arr[i])
  end for
```

Time complexity analysis – Tabular method:

Iteration (i)	Operation	Cost	Count
1 to 5	print	1	5
Loop control	increment i, check condition	1 each	5 each

Iteration (i)	Operation	Cost	Count
Initialization	i = 1	1	1

Total cost:

- Print operations: 5 × 1 = 5
- Loop overhead (increment + check): 5 × 2 = 10
- Initialization: 1
  Total = 16 operations (constant).

Since total operations **do not depend on input size n**, the time complexity is **O(1)**.
Key point: A loop that runs a fixed number of times (e.g., 5, 100, 1000) is O(1), even if the input is huge. Loop like for i = 1 to n would be O(n).
Another example – Sum of first 100 numbers:
text

```
Algorithm SumFirstHundred():
   sum = 0
   for i = 1 to 100 do
      sum = sum + i
   end for
   return sum
```

Runs exactly 100 iterations → O(1).
Real-world application: Processing a fixed-size header of a network packet (e.g., first 20 bytes) regardless of total packet length.

Other Important Question 7 – Find Ω notation of function. (3 marks)
Appeared in: S24 (Q1a, 03 marks)
Given: $f(n) = 2n^2 + n \cdot \lg\ n + 6n$
Ans:

- Ω notation gives a **lower bound**.
- Dominant term: $2n^2$
- For large n, $2n^2$ grows faster than $n\lg\ n + 6n$.
- We can choose c = 1, $n_0 = 1$: $f(n) \geq 1 \cdot n^2$ for sufficiently large n? Actually $2n^2 \geq n^2$ always.
- More precisely, $f(n) \geq n^2$ for n ≥ 1.
- Therefore, $f(n) = \Omega(n^2)$.
- Actually, since the highest order term is $2n^2$, the tight lower bound is also n^2, so $f(n) = \Theta(n^2)$ as well.

Final Answer: $f(n) = \Omega(n^2)$

Other Important Question 8 – Prove that $(n + 6)^2 = \Theta(n^3)$. (3 marks)
Appeared in: S25 (Q2a, 03 marks)
⚠ **Correction:** The statement is **incorrect**. $(n + 6)^2 = \Theta(n^2)$, not $\Theta(n^3)$.
Proof that it is NOT Θ(n³):

1. Expand: $(n + 6)^2 = n^2 + 12n + 36$
2. For Θ(n³), we would need constants $c_1, c_2 > 0$ such that: $c_1 n^3 \leq n^2 + 12n + 36 \leq c_2 n^3$ for all large n.
3. Check upper bound: $n^2 + 12n + 36 \leq c_2 n^3$ → Divide by n^3: $1/n + 12/n^2 + 36/n^3 \leq c_2$. As n → ∞, LHS → 0, so any $c_2 > 0$ works for large n – **upper bound is**

possible (since n^3 grows faster, it's an upper bound, not tight).

4. Check lower bound: $c_1 n^3 \leq n^2 + 12n + 36 \rightarrow$ Divide by n^3: $c_1 \leq 1/n + 12/n^2 + 36/n^3$. As $n \rightarrow \infty$, RHS $\rightarrow 0$. So for any fixed $c_1 > 0$, the inequality fails for sufficiently large n because RHS becomes smaller than c_1.
5. Hence, **no positive c_1 exists**. Therefore, $(n+6)^2$ is **not $\Omega(n^3)$**, so it cannot be $\Theta(n^3)$.

Correct statement: $(n+6)^2 = \Theta(n^2)$.

Answer for exam: The statement is **false**. Prove by contradiction or limit method: $\lim_{n\to\infty} \frac{(n+6)^2}{n^3} = 0$, so the function is $o(n^3)$, not $\Theta(n^3)$.

Question 9 (Arranged growth rates) – With given list. (3 marks)

Appeared in: S25 (Q3a, 03 marks)

Given list:

$$O(n\log n),\ O(n^{2n}),\ \Omega(\log n),\ O(n^{0.5}),\ O(n!),\ \Omega(2^n),\ O(n^{0.5}\log n)$$

Step 1 – Extract actual functions (ignoring O/Ω bounds, since they indicate the growth rate of the function itself):

- $n\log n$
- n^{2n} (super-exponential)
- $\log n$
- $n^{0.5} = \sqrt{n}$
- $n!$ (factorial)
- 2^n (exponential)
- $n^{0.5}\log n = \sqrt{n}\log n$

Step 2 – Order from lowest (slowest growth) to highest (fastest growth):

1. $\log n$ (logarithmic)
2. $\sqrt{n} = n^{0.5}$ (root-n)
3. $\sqrt{n}\log n = n^{0.5}\log n$ (slightly faster than root-n)
4. $n\log n$ (linearithmic)
5. 2^n (exponential)
6. $n!$ (factorial)
7. $n^{2n} = (n^2)^n$ (grows faster than n! because exponent depends on n)

Final ordered list (lowest to highest asymptotic order):

$\Omega(\log n) \rightarrow O(n^{0.5}) \rightarrow O(n^{0.5}\log n) \rightarrow O(n\log n) \rightarrow \Omega(2^n) \rightarrow O(n!) \rightarrow O(n^{2n})$

Note: The notations Ω and O here are simply attached to each function; they do not change the order because the asymptotic class of the function itself determines its position.

****