CS6402 DESIGN AND ANALYSIS OF ALGORITM

2mark Questions with Answer

UNIT-I

INTRODUCTION

1.Write the Characteristics of an algorithm.

Input: Zero / more quantities are externally supplied**.**

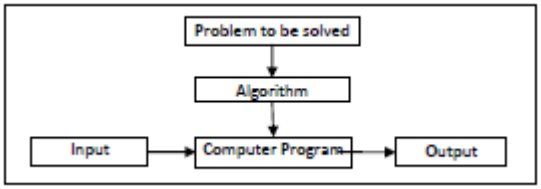
Output: At least one quantity is produced.

Definiteness: Each instruction is clear and unambiguous.

Finiteness: If the instructions of an algorithm is traced then for all cases the algorithm must terminates after a finite number of steps.

Efficiency: Every instruction must be very basic and runs in short time.

2.Give the diagram representation of Notion of algorithm.



3. What is the formula used in Euclid's algorithm for finding the greatest common divisor of two numbers?

Euclid's algorithm is based on repeatedly applying the equality gcd (m,n) = gcd(n, m mod n) until m mod n is equal to 0, since gcd(m,0)=m.

Example: gcd*(*60, 24*)* = gcd*(*24, 12*)* = gcd*(*12, 0*)* = 12.

4. Write Euclid's algorithm for computing gcd(m, n).

ALGORITHM *Euclid_gcd(m, n)*

//Computes gcd*(m, n)* by Euclid's algorithm

//Input: Two nonnegative, not-both-zero integers *m* and *n*

//Output: Greatest common divisor of *m* and *n*

while *n* ≠ 0 **do**

r ←*m* mod *n*

m←*n*

n←*r*

return *m*

Example: gcd*(*60, 24*)* = gcd*(*24, 12*)* = gcd*(*12, 0*)* = 12.

5. **What are the fundamental steps involved in algorithmic problem solving?**

The fundamental steps are

1. Understanding the problem
2. Decision making
    a. Ascertain the capabilities of computational device
    b. Choose between exact and approximate problem solving
    c. Decide on appropriate data structures

d. Algorithm design techniques

3. Methods for specifying the algorithm
    a. Natural language
    b. Pseudocode
    c. Flowchart
4. Proving an algorithms correctness
5. Analyzing an algorithm
    a. Time efficiency
    b. Space efficiency
    6. Coding an algorithm

6. What is an algorithm design technique? ®

- An algorithm design technique is a **general approach to solving problems algorithmically** that is applicable to a variety of problems from different areas of computing.
- Algorithms+ Data Structures = Programs
- Though Algorithms and Data Structures are independent, but they are combined together to develop program. Hence the choice of proper data structure is required before designing the algorithm.
- Algorithmic strategy / technique / paradigm are a general approach by which many problems can be solved algorithmically. E.g., Brute Force, Divide and Conquer, Dynamic Programming, Greedy Technique, Brach & bound and so on.

7. What is pseudo code?

A pseudo code is a mixture of a natural language and programming language constructs to specify an algorithm. A pseudo code is more precise than a natural language and its usage often yields more concise algorithm descriptions.

ALGORITHM Sum(a,b)
//Problem Description: This algorithm performs addition of two numbers
//Input: Two integers a and b
//Output: Addition of two integers
c←a+b
return c

8. What are the types of algorithm efficiencies?

The two types of algorithm efficiencies are

- Time efficiency: indicates how fast the algorithm runs
- Space efficiency: indicates how much extra memory the algorithm needs.

9. What is Space complexity?

Space complexity is a measure of the amount of working storage an algorithm needs. That means how much memory, in the worst case, is needed at any point in the algorithm. As with time complexity, we're mostly concerned with how the space needs grow, in big-Oh terms, as the size N of the input problem grows.

10. What is time complexity?

The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input. The time complexity of an algorithm is commonly expressed using big O notation, which excludes coefficients and lower order terms.

11. How will you Prove an Algorithm's Correctness?

Once an algorithm has been specified then its ***correctness*** must be proved.

- An algorithm must yields a required result for every legitimate input in a finite amount of time. For example, the correctness of Euclid's algorithm for computing the greatest common divisor stems from the correctness of the equality gcd(m, n) = gcd(n, m mod n).
- A common technique for proving correctness is to use mathematical induction because an algorithm's iterations provide a natural sequence of steps needed for such proofs.
- The notion of correctness for approximation algorithms is less straightforward than it is for exact algorithms. The error produced by the algorithm should not exceed a predefined limit.
- Better than existing algorithm by comparison method.

12. Mention some of the important problem types?

Some of the important problem types are as follows

1. Sorting
2. Searching
3. String processing
4. Graph problems
5. Combinatorial problems
6. Geometric problem
7. Numerical problems

13. What are the steps involved in the analysis framework?

The various steps for the analysis framework are as follows

1. Measuring the input's size
2. Units for measuring running time
3. Orders of growth
4. Worst case, best case and average case efficiencies

14. Write about fundamentals of the analysis of algorithm efficiency.

The efficiency of an algorithm can be in terms of time and space.The algorithm efficiency can be analyzed by the following ways.

a. Analysis Framework.
b. Asymptotic Notations and its properties.
c. Mathematical analysis for Recursive algorithms.
d. Mathematical analysis for Non-recursive algorithms.

15. What is the basic operation of an algorithm and how is it identified?

The most important operation of the algorithm is called the basic operation of the algorithm, the operation that contributes the most to the total running time. It can be identified easily because it is usually the most time-consuming operation in the algorithms innermost loop. The most important operation (+, -, *, /) of the algorithm, called the basic operation. One possible approach is to count the number of times each of the algorithm's operations is executed. This approach is excessively difficult.

16. What is the running time of a program implementing the algorithm?

- The running time T(n) is given by the following formula C(n)

- Cop (n) is the time of execution of an algorithm's basic operation on a particular computer and C(n) is the number of times this operation needs to be executed for the particular algorithm.

17. What are exponential growth functions?

The functions 2n and n! are exponential growth functions, because these two functions grow so fast that their values become astronomically large even for smaller values of n.

18. What is worst-case efficiency?

The worst-case efficiency of an algorithm is its efficiency for the worst-case input of size n, which is an input or inputs of size n for which the algorithm runs the longest among all possible inputs of that size.

19. What is best-case efficiency?

The best-case efficiency of an algorithm is its efficiency for the best-case input of size n, which is an input or inputs for which the algorithm runs the fastest among all possible inputs of that size.

20. What is average case efficiency?

The average case efficiency of an algorithm is its efficiency for an average case input of size n. It provides information about an algorithm behavior on a “typical” or “random” input.

21. What is amortized efficiency?

In some situations, a single operation can be expensive, but the total time for the entire sequence of n such operations is always significantly better that the worst-case efficiency of that single operation multiplied by n. this is called amortized efficiency.

22. Define asymptotic notation. / List some asymptotic notations ®

O-notation

A function t(n) is said to be in O(g(n)), denoted by t(n) ϵ O(g(n)), if t(n) is bounded above by some constant multiple of g(n) for all large n, i.e., if there exists some positive constant c and some nonnegative integer n0 such that T(n) **<=** cg(n) for all n **>=** n0

Ω notation

A function t(n) is said to be in Ω (g(n)), denoted by t(n) ϵ Ω (g(n)), if t(n) is bounded below by some constant multiple of g(n) for all large n, i.e., if there exists some positive constant c and some nonnegative integer n0 such that T(n) **>=** cg(n) for all n **>=** n0

ɵ -notation

A function t(n) is said to be in ɵ (g(n)), denoted by t(n) ϵ ɵ (g(n)), if t(n) is bounded both above & below by some constant multiple of g(n) for all large n, i.e., if there exists some positive constants c1 & c2 and some nonnegative integer n0 such that c2g(n) **<=** t(n) <= c1g(n) for all n **>=** n0

23. Compare asymptotic notations

O-notation	**Ω notation**	**ɵ -notation**
T(n) **<=** cg(n)	T(n) **>=** cg(n)	c2g(n) **<=** t(n) <= c1g(n)
Easy to calculate.	Easy to calculate	More calculations than others.(Tight bound)
Useful for worst case analysis.	Useful for best case analysis.	Useful for average case analysis.
Bounded by upper curve.	Bounded by lower curve	Bounded by both upper curve and lower curve

Bounded by least maximum.	Bounded by most minimum	Bounded by least maximum and most minimum

24. Write worst case, avg. case and best-case analysis for binary search.

Time complexity of Binary search		
Best case	Average case	Worst case
Θ(1)	Θ(log2 n)	Θ(log2 n)

25. Write worst case, avg. case and best-case analysis for linear search

Time complexity of Linear search		
Best case	Average case	Worst case
Θ(1)	Θ(n)	Θ(n)

26. Compare sequential search and binary search

Sequential search	**Binary search**
This is the **simple** technique of searching an element	This is the **efficient** technique of searching an element
This technique does not require the list to be sorted.	This technique requires the list to be sorted. Then only this method is applicable.
Every element of the list may get compared with the key element.	Only the mid element of the list and sublist are compared with the key element
The worst-case time complexity is **O(n)**.	The worst case time complexity is **O(log n)**.

27. Define O-notation?

A function t(n) is said to be in O(g(n)), denoted by t(n) ϵ O(g(n)), if t(n) is bounded above by some constant multiple of g(n) for all large n, i.e., if there exists some positive constant c and some nonnegative integer n0 such that T(n) **<=** cg(n) for all n **>=** n0
E.g., n3-5n2-7n+8 ϵ O(n3)

28. Define Ω -notation?

A function t(n) is said to be in Ω (g(n)), denoted by t(n) ϵ Ω (g(n)), if t(n) is bounded below by some constant multiple of g(n) for all large n, i.e., if there exists some positive constant c and some nonnegative integer n0 such that T(n) **>=** cg(n) for all n **>=** n0
E.g., n3-5n2-7n+8 ϵ Ω (n2)

29. Define ɵ -notation?

A function t(n) is said to be in ɵ(g(n)), denoted by t(n) ϵ ɵ(g(n)), if t(n) is bounded both above & below by some constant multiple of g(n) for all large n, i.e., if there exists some positive constants c1 & c2 and some nonnegative integer n0 such that
c2g(n) **<=** t(n) <= c1g(n) for all n **>=** n0 E.g., n3-5n2-7n+8 ϵ ɵ (n3)

30. Mention the useful property, which can be applied to the asymptotic notations.

If t1(n) ϵ O(g1(n)) and t2(n) ϵ O(g2(n)) then t1(n)+t2(n) ϵ max {g1(n),g2(n)} this property is also true for Ω and ɵ notations. This property will be useful in analyzing algorithms that comprise of two consecutive executable parts.

31. What are the basic asymptotic efficiency classes?
List the various classes of the time efficiencies of algorithms. ®

Constant : 1 Quadratic : n2
Logarithmic : log n Cubic : n3
Linear : n Exponential : 2n
N-log-n : nlog n Factorial : n!

32. Compare the order of growth 2^n and n^2.

n	n^2	2^n
Polynomial	Quadratic	Exponential
1	1	2
2	4	4
4	16	16
8	64	$2.6 \bullet 10^2$
10	10^2	10^3
16	$2.6 \bullet 10^2$	$6.5 \bullet 10^4$
10^2	10^4	$1.3 \bullet 10^{30}$
Complexity	Low	High
Growth	Low	Very High

33. Compare the order of grow $\log_2 n$and $\sqrt{n}$

n	$\log_2 n$	$\sqrt{n}$
Polynomial	Logarithmic	Square root
1	0	1
4	2	2
10	3.3	3.3
16	4	4
10^2	6.6	10
10^3	10	31
10^4	13	10^2
10^5	17	316
10^6	20	10^3
Complexity	Low	High
Growth	Low	High

34. Show that T(n) = T(n/2) +1 is ɵ (log n)

T(n) = T(n/2) + n^0, a=1, b=2, and d=0;
By master theorem, T(n) ϵ ɵ (n^d log n)
Therefore, T(n) ϵ ɵ (n0 log n)
T(n) ϵ ɵ (log n)

35. If $f(n) = 5n^2 + 6n + 4$ then prove that f(n) is $O(n^2)$

$f(n) = 5n^2 + 6n + 4$
$< = 5n^2 + 6n^2 + 4n^2$
$= 15n^2$
f(n) ϵ O (n^2) where n>=1 and c=15

36. Write a recursive algorithm to find the number of binary digits in the binary

representation of an integer.

ALGORITHM BinRec(n)
Input A positive decimal integer n
Output The number of binary digits in n's binary representation
if n=1 return 1
else return BinRec(n/2)+1

37. Write algorithm for linear search.

ALGORITHM *Search(A*[0*..n* - 1]*, K)*
//Searches for a given value in a given array by sequential search
//Input: An array *A*[0*..n* - 1] and a search key *K*
//Output: The index of the first element in *A* that matches *K* or -1 if there are no
// matching elements
i ←0
while *i* < *n* and *A*[*i*] ≠ *K* do
i ←*i* + 1
if *i* < *n* return *i*
else return -1

38. Define recursive algorithm.

A recursive algorithm is an algorithm which calls itself with "smaller (or simpler)" input values, and which obtains the result for the current input by applying simple operations to the returned value for the smaller (or simpler) input.

EXAMPLE: Compute the factorial function F(n) = n! for an arbitrary nonnegative integer n. Since *n!= 1•. . . .* • (*n* − *1*) • *n* = (*n* − *1*)*!* • *n*, for n ≥ 1 and 0!= 1 by definition, we can compute *F*(*n*) = *F*(*n* − *1*) • *n* with the following recursive algorithm.

ALGORITHM F(n)
//Computes *n*! recursively
//Input: A nonnegative integer *n*
//Output: The value of *n*!
if *n* = 0 return 1
else return *F(n* − 1*)* * *n*

39. Write the General Plan for Analyzing the Time Efficiency of Nonrecursive Algorithms

1. Decide on a *parameter* (or parameters) indicating an input's size.
2. Identify the algorithm's *basic operation* (in the innermost loop).
3. Check whether the *number of times the basic operation is executed* depends only on the size of an input. If it also depends on some additional property, the worst-case, average-case, and, if necessary, best-case *efficiencies* have to be investigated separately.
4. Set up a *sum* expressing the number of times the algorithm's basic operation is executed.
5. Using standard formulas and rules of sum manipulation either find a closed form formula for the count or at the least, establish its order of growth.

40. Write the General Plan for Analyzing the Time Efficiency of Recursive Algorithms

1. Decide on a parameter (or parameters) indicating an input's size.
2. Identify the algorithm's *basic operation*.

3. Check whether the number of times the basic operation is executed can vary on different inputs of the same size; if it can, the worst-case, average-case, and best-case efficiencies must be investigated separately.
4. *Set up a* recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed.
5. Solve the recurrence or, at least, ascertain the order of growth of its solution.

41. Differentiate between recursive Algorithms and non-recursive Algorithms. Compare and contrast between recursive Algorithms and non-recursive Algorithms.

Non-Recursive Algorithms	Recursive Algorithms
Similarities	
Decide on parameter n indicating input size	Decide on parameter n indicating input size
Identify algorithm's basic operation	Identify algorithm's basic operation
Determine worst, average, and best case for input of size n	Determine worst, average, and best case for input of size n
Differences	
A non-recursive technique is anything that doesn't use recursion.	A recursive technique is nothing but a function calls itself.
Set up summation for C(n) reflecting algorithm's loop structure	Set up a recurrence relation and initial condition(s) for C(n)-the number of times the basic operation will be executed for an input of size n
No recursion. So no recursive calls count	Count the number of recursive calls.
E.g. Insertion sort, Matrix multiplication, maximum element in array.	E.g. Fibonacci, Factorial, Merge sort, Quick sort. Tower of Hanoi.

42. What is called Substitution Method? Write its types.

Replacing one term with another equal term is called substitution. Forward substitution and backward substitution are two types of substitution. It is very useful in solving recursive algorithms.