MGM's College of Engineering and Technology Kamothe, Navi Mumbai
Department of Computer Engineering
Assignment-1

Course Code: CSC402 Course Name: Analysis of Algorithms Class: SE/IV AY: 2021-2022 Date of Issue: 11/02/2022

Q. No	Question
Q1. Fill in the blanks	
a)	O(g(n)) = { f(n)l there exists positive constants c and n_0 such that $0 \le f(n) \le c\ g(n)$ for all $n \ge n_0$ }
b)	θ(g(n) = {f(n): there exist positive constants c_1 , c_2 , and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$ }
c)	**f(n)** = {f(n): there exist positive constants c and n_0 such that $0 \le c\ g(n) \le f(n)$ for all $n \ge n_0$ }
Q2. Choose Correct Options	

a)	The time complexity of the following code is:
	int x, y, z = 0;
	for (x = n / 2; x <= n; x++)
	{
	for (y = 2; y <= n; y = y * 2)
	{
	z = z + n / 2;
	}
	}
	a) O(n log n)
	b) $O(n^2)$
	c) O(log n)
	d) O(n)

b)	Which of the following class of problems need not be in NP class? a) Both NP-hard and NP-complete b) Only NP-Hard c) Only NP-complete d) None of the above

c)	Which of the following asymptotic notation provides only an asymptotic upper bound on a function? a) Θ-notation b) O-notation c) Ω-notation d) None of the above
d)	The best case running time of the binary search is a) O(log n) b) O(n) c) O(1) d) O(n log n)
e)	If the recurrence relation is $T(n) = 7T(n/3) + n^2$ then its solution is a) $\Theta(n^2)$ b) $\Theta(\log_2 n)$ c) $\Theta(n)$ d) $\Theta(n\log_2 n)$
Q3. State whether the following statements are true or false (Give Reasons)	
a)	Time complexity of an algorithm is the amount of time required for it to execute. True
b)	Best case time complexity is the maximum amount of time an algorithm takes to execute a specific set of inputs. False
c)	The Base case is also known as an escape clause which is used to terminate the algorithm. True
Q4. Name the following or define or design the following	
a) Ans:	Define NP Class The NP in NP class stands for Non-deterministic Polynomial Time. It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.
b) Ans:	Define NP hard An NP-hard problem is at least as hard as the hardest problem in NP and it is the class of the problems such that every problem in NP reduces to NP- hard.
c) Ans:	Define NP complete A problem that is NP and NP-hard is NP- complete.
Q5. Answer the following questions in brief (20 to 30 words)	

a) Explain P, NP, NP-Hard and NP-Complete complexity class.

Ans:

P Class

• The P in the P class stands for Polynomial Time. It is the collection of decision problems(problems with a "yes" or "no" answer) that can be solved by a deterministic machine in polynomial time.

• Features: 1. The solution to P problems is easy to find. 2. P is often a class of computational problems that are solvable and tractable. Tractable means that the problems can be solved in theory as well as in practice. But the problems that can be solved in theory but not in practice are known as intractable.

• This class contains many natural problems like:

– Calculating the greatest common divisor.

– Finding a maximum matching.

– Decision versions of linear programming.

NP Class

• The NP in NP class stands for Non-deterministic Polynomial Time. It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.

• Features:

1. The solutions of the NP class are hard to find since they are being solved by a non-deterministic machine but the solutions are easy to verify.

2. Problems of NP can be verified by a Turing machine in polynomial time.

• Example:

• Let us consider an example to better understand the NP class. Suppose there is a company having a total of 1000 employees having unique employee IDs. Assume that there are 200 rooms available for them. A selection of 200 employees must be paired together, but the CEO of the company has the data of some employees who can't work in the same room due to some personal reasons.

This is an example of an NP problem. Since it is easy to check if the given choice of 200 employees proposed by a coworker is satisfactory or not i.e. no pair taken from the coworker list appears on the list given by the CEO. But generating such a list from scratch seems to be so hard as to be completely impractical.

• It indicates that if someone can provide us with the solution to the problem, we can find the correct and incorrect pair in polynomial time. Thus for the NP class problem, the answer is possible, which can be calculated in polynomial time.

• This class contains many problems that one would like to be able to solve effectively:

– Boolean Satisfiability Problem (SAT).

– Hamiltonian Path Problem.

– Graph coloring.

NP-hard class

• An NP-hard problem is at least as hard as the hardest problem in NP and it is the class of the problems such that every problem in NP reduces to NP-hard.

• Features:

1. All NP-hard problems are not in NP.

2. It takes a long time to check them. This means if a solution for an NP-hard problem is given then it takes a long time to check whether it is right or not.

3. A problem A is in NP-hard if, for every problem L in NP, there exists a polynomial-time reduction from L to A.

• Some of the examples of problems in Np-hard are:

– Halting problem.

– Qualified Boolean formulas.

– No Hamiltonian cycle.

	NP-complete A problem that is NP and NP-hard is NP-complete.
b) Ans:	Explain Master method with examples for all three cases. Master theorem for Decreasing function General form for decreasing functions will be • T(n) = aT(n-b) + f(n), where a and b are the constants. a>0, b>0 and f(n)=O(n^k) where k≥0. a = Number of subproblems and b = The cost of dividing and merging the subproblems. To solve this type of recurrence relation there are 3 cases : • Case : 1 – If a = 1 then T(n) = O(n^(k+1)) or simply T(n) = O(n*f(n)). • Case : 2 – If a>1 then T(n) = O(a^(n/b) * f(n)). • Case : 3 – If a<1 then T(n) = O(n^k) or simply T(n) = O(f(n)). Master theorem for Dividing functions • The general form for the dividing functions : • T(n) = aT(n/b) + f(n), where a and b are constants. a≥1, b>1 and f(n) can be expressed as O(n^k * (logn)^p). a = Number of subproblems and b = The cost of dividing and merging the subproblems. • To find the time complexity for these kinds of functions again there are 3 cases. Here we have to find two things 1. log a base b and 2. k • For simplicity, let's take log a base b as x for all below cases. • Case : 1 – If k < x then T(n) = O(n^x). • Case : 2 – If k = x then again some cases here comes p in the picture : 1. If p > -1 then T(n) = O(n^k * (logn)^(p+1)) that is simply O(f(n)*logn) 2. If p = -1 then T(n) = O(n^k * log(logn)) 3. if p < -1 then T(n) = O(n^k) • **Case : 3** – If k > x then T(n) = O(f(n)). --- 1. Consider the recurrence T (n) = 4 T (n/2) + n 2. Consider the recurrenceT (n) = 2 T (n/2) + n log n 3. consider the recurrenceT (n) = T (n/3) + n 4. Consider the recurrence T (n) = 9 T (n/3) + n ^2.5
c) Ans:	Explain Best Case, Average Case and Worst Case. • Best case Running Time: Searching key element present at first index. • Worst case Running Time: Searching key element present at last index. • Average case Running Time: All possible case time / number of cases
Q6. Answer the following questions in brief (50 to 70 words)	

a)	Apply a selection sort method to sort the following numbers. Show each step clearly. 28, 78, 45, 8, 32, 56. Derive the complexity of selection sort.
Ans:	Enter number of elements to be sorted: 6 Enter numbers to be sorted: 28 78 45 8 32 56 The given list 28 78 45 8 32 56 Iteration 1: 28 56 45 8 32 78 Iteration 2: 28 32 45 8 56 78 Iteration 3: 28 32 8 45 56 78 Iteration 4: 28 8 32 45 56 78 Iteration 5: 8 28 32 45 56 78 The final sorted list 8 28 32 45 56 78 Complexity of selection sort O(n^2)
b)	Apply a selection sort method to sort the following numbers. Show each step clearly. 8, 7, 4, 5, 2, 1. Derive the complexity of Insertion sort.
Ans:	Enter number of elements to be sorted: 6 Enter numbers to be sorted: 8 7 4 5 2 1 The given list 8 7 4 5 2 1 Iteration 1: 1 7 4 5 8 Iteration 2: 1 5 4 7 8 Iteration 3: 1 4 5 7 8 The final sorted list 1 4 5 7 8 Complexity of insertion sort O(n^2)

c) Solve the following Recurrences using Recursion-Tree Method. T(n) =2 T(n/2) + n

Ans:

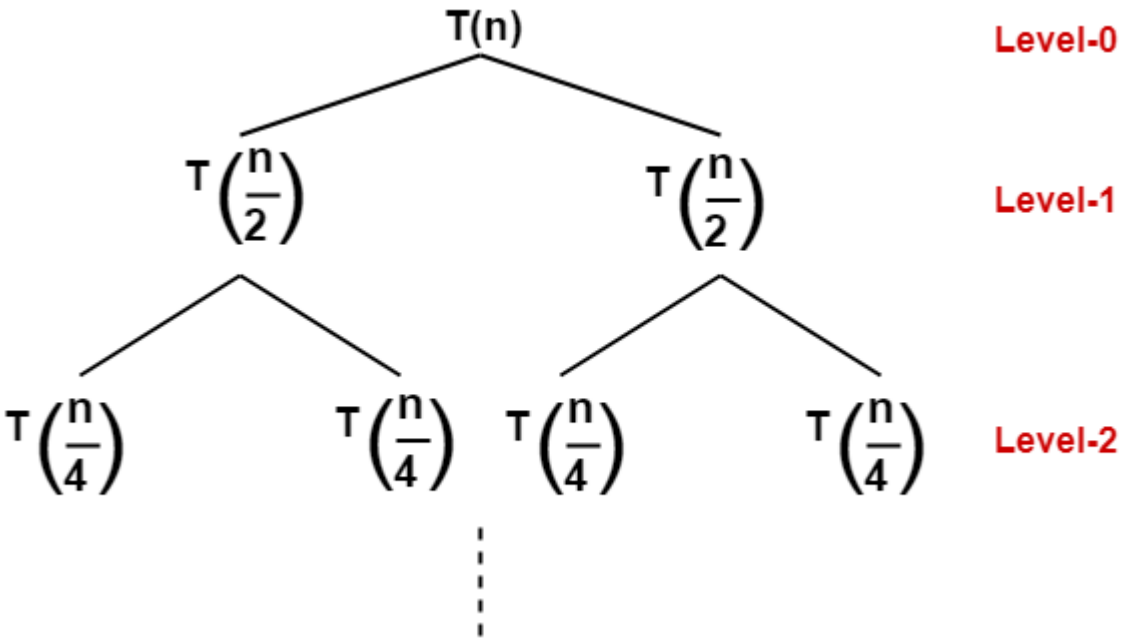
Step-01:

Draw a recursion tree based on the given recurrence relation.

The given recurrence relation shows-

- A problem of size n will get divided into 2 sub-problems of size n/2.
- Then, each sub-problem of size n/2 will get divided into 2 sub-problems of size n/4 and so on.
- At the bottom most layer, the size of sub-problems will reduce to 1.

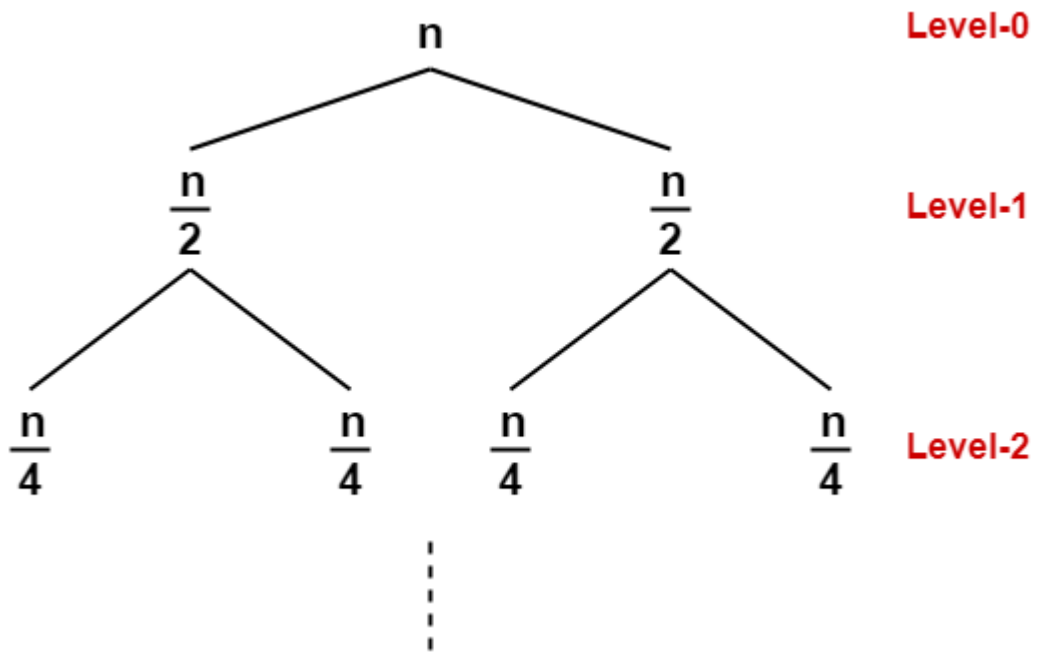
This is illustrated through following recursion tree-



The given recurrence relation shows-

- The cost of dividing a problem of size n into its 2 sub-problems and then combining its solution is n.
- The cost of dividing a problem of size n/2 into its 2 sub-problems and then combining its solution is n/2 and so on.

This is illustrated through following recursion tree where each node represents the cost of the corresponding sub-problem-



Step-02:

Determine cost of each level-

- Cost of level-0 = n
- Cost of level-1 = n/2 + n/2 = n
- Cost of level-2 = n/4 + n/4 + n/4 + n/4 = n and so on.

Step-03:

Determine total number of levels in the recursion tree-

- Size of sub-problem at level-0 = $n/2^0$
- Size of sub-problem at level-1 = $n/2^1$
- Size of sub-problem at level-2 = $n/2^2$

Continuing in similar manner, we have-

Size of sub-problem at level-i = $n/2^i$

Suppose at level-x (last level), size of the sub-problem becomes 1. Then-

$$n / 2^x = 1$$

$$2^x = n$$

Taking log on both sides, we get-

$$x\log 2 = \log n$$

$$x = \log_2 n$$

$\therefore$ Total number of levels in the recursion tree = $\log_2 n + 1$

Step-04:

Determine number of nodes in the last level-

- Level-0 has 2^0 nodes i.e. 1 node
- Level-1 has 2^1 nodes i.e. 2 nodes
- Level-2 has 2^2 nodes i.e. 4 nodes

Continuing in similar manner, we have-

Level-$\log_2 n$ has $2^{\log_2 n}$ nodes i.e. n nodes

Step-05:

Determine cost of last level-

Cost of last level = n x T(1) = θ(n)

Step-06:

Add costs of all the levels of the recursion tree and simplify the expression so obtained in terms of asymptotic notation-

$$T(n) = \underbrace{\{ n + n + n + ... \}}_{\text{For } \log_2 n \text{ levels}} + \theta(n)$$

= n x $\log_2 n$ + θ (n)

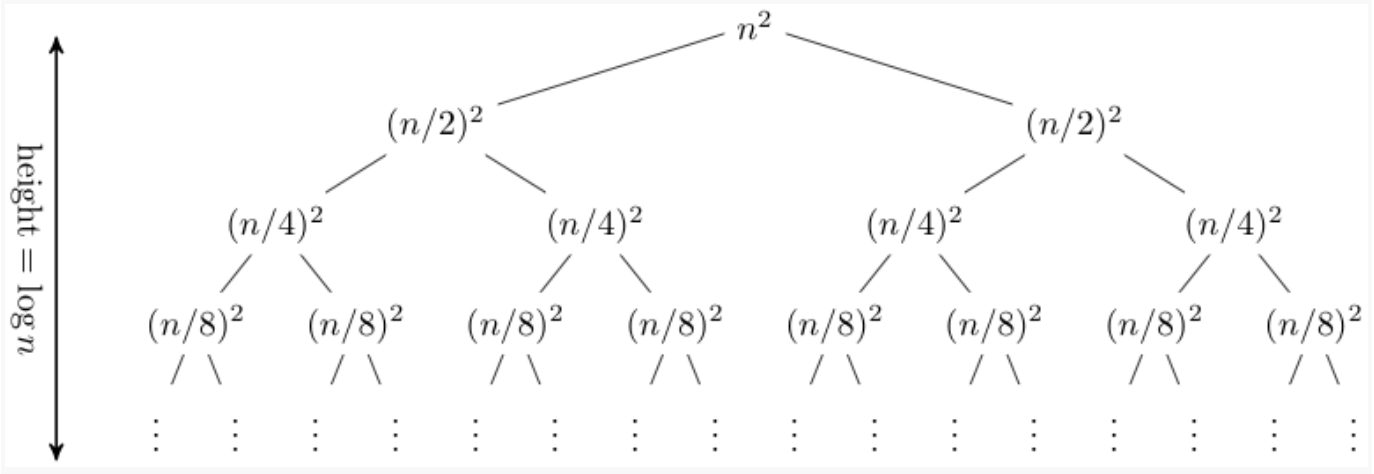
= n$\log_2 n$ + θ (n)

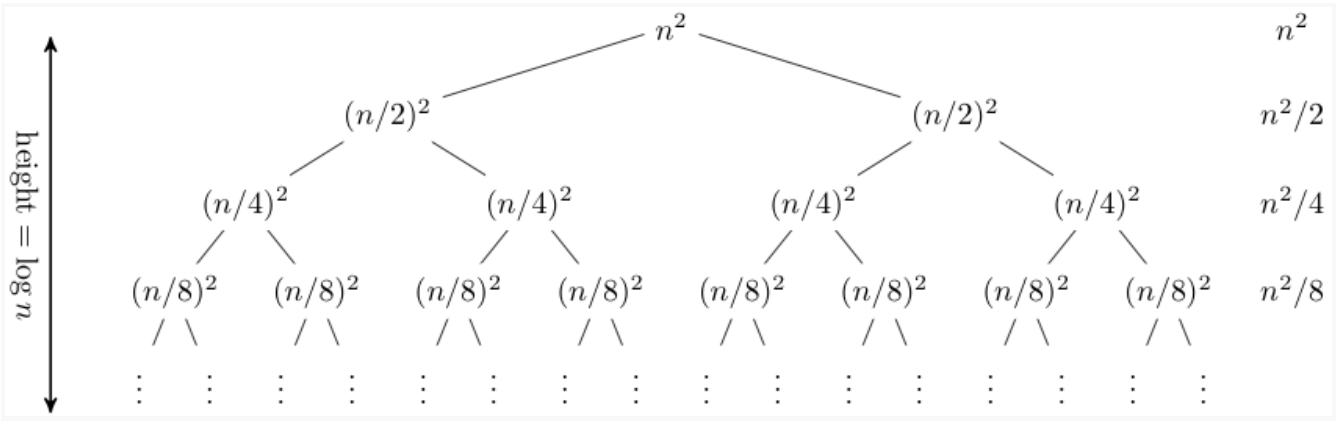
= **θ (n$\log_2 n$)**

Q7. Think and Answer

a) Consider the recurrence T(n) = 4T (n/2) + log n. What is the expression for the runtime T (n) if recurrence can be solved using Recursion Tree Method?

Ans: For this recurrence, there are again $a=4$ subproblems, each dividing the input by $b=2$, but now the work done on each call is $f(n)=n^3$. Again $n^{\log_b a}$ is n^2, and $f(n)$ is thus $\Omega(n^{2+\varepsilon})$ for $\varepsilon=1$. Moreover, $4(n/2)^3 \le kn^3$ for $k=1/2$, so Case 3 applies. Thus $T(n)$ is $\Theta(n^3)$.





b) Ans:

Consider the recurrence T(n) = 4T (n/2) + log n. What is the expression for the runtime T (n) if recurrence can be solved using the Substitution Method?

T(n) = 2T(n-1) + 4

Q8. My Ideas

a) Ans:

Basic Issues Related to Algorithms

- How to design algorithms
- How to express algorithms
- Proving correctness of algorithms
- Efficiency
  - Theoretical analysis
  - Empirical analysis
- Optimality

b) Ans:

How good is the algorithm?

- It is easy to understand.
- Algorithm is a step-wise representation of a solution to a given problem.

	• In Algorithms the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.