Algorithms Analysis Practice Test (Sample)

Study Guide

BY EXAMZIFY

Everything you need from our exam experts!

Sample study guide. For the full version with hundreds of questions, visit:
https://algorithmsanalysis.examzify.com

Copyright © 2026 by Examzify - A Kaluba Technologies Inc. product.

ALL RIGHTS RESERVED.

No part of this book may be reproduced or transferred in any form or by any means, graphic, electronic, or mechanical, including photocopying, recording, web distribution, taping, or by any information storage retrieval system, without the written permission of the author.

Notice: Examzify makes every reasonable effort to obtain accurate, complete, and timely information about this product from reliable sources.

SAMPLE

Table of Contents

SAMPLE

Introduction

Preparing for a certification exam can feel overwhelming, but with the right tools, it becomes an opportunity to build confidence, sharpen your skills, and move one step closer to your goals. At Examzify, we believe that effective exam preparation isn't just about memorization, it's about understanding the material, identifying knowledge gaps, and building the test-taking strategies that lead to success.

This guide was designed to help you do exactly that.

Whether you're preparing for a licensing exam, professional certification, or entry-level qualification, this book offers structured practice to reinforce key concepts. You'll find a wide range of multiple-choice questions, each followed by clear explanations to help you understand not just the right answer, but why it's correct.

The content in this guide is based on real-world exam objectives and aligned with the types of questions and topics commonly found on official tests. It's ideal for learners who want to:

- **Practice answering questions under realistic conditions,**
- **Improve accuracy and speed,**
- **Review explanations to strengthen weak areas, and**
- **Approach the exam with greater confidence.**

We recommend using this book not as a stand-alone study tool, but alongside other resources like flashcards, textbooks, or hands-on training. For best results, we recommend working through each question, reflecting on the explanation provided, and revisiting the topics that challenge you most.

Remember: successful test preparation isn't about getting every question right the first time, it's about learning from your mistakes and improving over time. Stay focused, trust the process, and know that every page you turn brings you closer to success.

Let's begin.

SAMPLE

How to Use This Guide

This guide is designed to help you study more effectively and approach your exam with confidence. Whether you're reviewing for the first time or doing a final refresh, here’s how to get the most out of your Examzify study guide:

1. Start with a Diagnostic Review

Skim through the questions to get a sense of what you know and what you need to focus on. Your goal is to identify knowledge gaps early.

2. Study in Short, Focused Sessions

Break your study time into manageable blocks (e.g. 30 – 45 minutes). Review a handful of questions, reflect on the explanations.

3. Learn from the Explanations

After answering a question, always read the explanation, even if you got it right. It reinforces key points, corrects misunderstandings, and teaches subtle distinctions between similar answers.

4. Track Your Progress

Use bookmarks or notes (if reading digitally) to mark difficult questions. Revisit these regularly and track improvements over time.

5. Simulate the Real Exam

Once you're comfortable, try taking a full set of questions without pausing. Set a timer and simulate test-day conditions to build confidence and time management skills.

6. Repeat and Review

Don’t just study once, repetition builds retention. Re-attempt questions after a few days and revisit explanations to reinforce learning. Pair this guide with other Examzify tools like flashcards, and digital practice tests to strengthen your preparation across formats.

There’s no single right way to study, but consistent, thoughtful effort always wins. Use this guide flexibly, adapt the tips above to fit your pace and learning style. You've got this!

SAMPLE

Questions

SAMPLE

1. **Given the function T(n) = T(n/2) + 1, what does this imply about its complexity?**
   - A. Linear time complexity
   - B. Constant time complexity
   - C. Logarithmic time complexity
   - D. Quadratic time complexity

2. **Breadth first search primarily does what in graph traversal?**
   - A. Scans each incident node along with its children
   - B. Scans all incident edges before moving to other node
   - C. Is the same as backtracking
   - D. Scans all the nodes in random order

3. **In linear programming, must both constraints and optimization criteria be linear functions?**
   - A. True
   - B. False
   - C. Always
   - D. Only in special cases

4. **What is the space complexity of QuickSort in its average case?**
   - A. O(n)
   - B. O(log n)
   - C. O(n log n)
   - D. O(1)

5. **True/False: A tree is a type of graph that contains cycles.**
   - A. True
   - B. False

SAMPLE

6. **What is the big-O complexity of the expression represented by the second line on the left?**
   - **A. O(n)**
   - **B. O(log n)**
   - **C. O(2^n)**
   - **D. O(n^2)**

7. **What does it imply when an algorithm has a space complexity of O(1)?**
   - **A. The algorithm requires minimal space regardless of input size**
   - **B. The space requirement grows linearly with input size**
   - **C. The algorithm cannot execute without additional memory**
   - **D. The algorithm is not efficient**

8. **Which of the following typically characterizes divide-and-conquer strategies?**
   - **A. They exclusively use iteration**
   - **B. They break a problem into independent smaller problems**
   - **C. They solely rely on brute force methods**
   - **D. They cannot be used for sorting**

9. **What is the total number of fundamental instructions executed by the routine if n = 4?**
   - **A. 8**
   - **B. 10**
   - **C. 12**
   - **D. 4 + 2n**

10. **How can you classify a problem as NP-complete?**
    - **A. If it has no solution**
    - **B. If it can be solved in polynomial time**
    - **C. If it is in NP and every problem in NP can be reduced to it in polynomial time**
    - **D. If it can be solved using brute force**

SAMPLE

Answers

SAMPLE

1. **C**
2. **B**
3. **B**
4. **B**
5. **B**
6. **C**
7. **A**
8. **B**
9. **D**
10. **C**

SAMPLE

**Sample study guide, visit https://algorithmsanalysis.examzify.com
for the full version with hundreds of practice questions**

Explanations

SAMPLE

1. Given the function T(n) = T(n/2) + 1, what does this imply about its complexity?

A. Linear time complexity

B. Constant time complexity

<u>C. Logarithmic time complexity</u>

D. Quadratic time complexity

The function T(n) = T(n/2) + 1 describes a recurrence relation where the problem size reduces by half with each recursive call, and there is a constant amount of additional work done (which is the +1). This pattern is indicative of logarithmic growth. To analyze the complexity, we can apply the Master Theorem or simply observe the behavior of the recurrence. Starting with T(n), you can see: 1. At the initial step, we have T(n). 2. The next call is T(n/2), which adds a constant 1. 3. The following step would be T(n/4), adding another constant 1. 4. This process continues until n is reduced to 1. If we continue this pattern, each level of the recurrence contributes a constant amount of work, and we effectively halve the problem size at each step. The number of steps taken to reduce n to 1 will be $\log_2(n)$. Each of these steps contributes a constant amount of additional work. This results in the overall time complexity being proportional to the number of steps taken in the recurrence, which is log(n). Therefore, the function T(n) grows logarithmically with respect to n, leading to a logarith

2. Breadth first search primarily does what in graph traversal?

A. Scans each incident node along with its children

<u>B. Scans all incident edges before moving to other node</u>

C. Is the same as backtracking

D. Scans all the nodes in random order

In the context of graph traversal, breadth-first search (BFS) explores the graph level by level, meaning that it examines each node and all of its children (or direct neighbors) systematically before moving on to the next level of nodes. When executing BFS, the algorithm utilizes a queue to keep track of nodes that are pending examination. This means that each node is processed in the order they are discovered, and all nodes connected to the current node (its incident edges) are explored before moving on to nodes that are at the next level of the graph. The focus of BFS is to ensure that all nodes at one depth are visited before any nodes at the subsequent depth, making it a very structured traversal method. This differs from random exploration or depth-focused searches like backtracking, which do not prioritize visiting nodes in a level-wise manner. Thus, the defining characteristic of BFS is its thorough examination of all incident edges of a node before transitioning to new nodes, ensuring that the entire breadth of the current layer is scanned. The other options suggest different approaches or characteristics not inherent to BFS, underscoring the uniqueness of its methodology.

SAMPLE

3. In linear programming, must both constraints and optimization criteria be linear functions?

A. True

B. False

C. Always

D. Only in special cases

In linear programming, the defining characteristic is that both the constraints and the objective function are linear functions. Therefore, the statement that both constraints and optimization criteria must be linear is essential to the definition of linear programming. If either the constraints or the objective function were non-linear, it would not fit the classification of linear programming, but rather another type of mathematical optimization problem. Considering the provided context, it is evident that while linear constraints and an objective function are the norm in linear programming, there can be cases where either element is not strictly linear. For instance, some optimization problems may involve non-linear functions, thereby transitioning them into the realm of non-linear programming. Hence, the assertion that it’s not a requirement for both components to be linear in all scenarios aligns with the understanding of linear programming's flexible nature, allowing certain variations in problem formulation. Thus, stating that it is false to maintain that both constraints and optimization criteria must be linear is appropriate for reflecting the broader possibilities outside the stringent structure of linear programming.

4. What is the space complexity of QuickSort in its average case?

A. O(n)

B. O(log n)

C. O(n log n)

D. O(1)

In the average case for QuickSort, the space complexity is determined primarily by the recursion stack during the sorting process. QuickSort works by selecting a "pivot" element and partitioning the array into two halves, which are then sorted independently. In the average case, the pivot usually divides the array into two roughly equal parts. This results in a logarithmic depth of recursion, as each recursive call handles part of the array. Specifically, for an array of size n, you could expect about log(n) levels of recursive calls (assuming balanced partitions). At each level of recursion, space is used on the stack for managing function calls. Since the maximum depth of the recursion stack corresponds to the logarithm of the size of the input data, the average space complexity for QuickSort is O(log n). This understanding is based on the method of partitioning and the nature of recursive function calls in QuickSort, which permits efficient use of stack space while sorting. The other options reflect either incorrect assessments of recursion depth or total space usage through other interpretations, which do not apply to the average case behavior of QuickSort.

SAMPLE

5. True/False: A tree is a type of graph that contains cycles.

A. True

B. False

A tree is indeed a type of graph, and by definition, it is a connected acyclic graph. This means that a tree does not contain any cycles. Every two vertices in a tree are connected by exactly one simple path, ensuring that no loops or cycles exist within the structure. Additionally, trees have a hierarchical structure with a single root node, and they are characterized by having \(n\) vertices and \(n-1\) edges, where \(n\) represents the number of nodes in the tree. In contrast, the presence of cycles would contradict the basic properties of a tree, making the assertion that a tree contains cycles false. This fundamental characteristic distinguishes trees from other types of graphs, such as directed or undirected graphs, which may contain cycles. Understanding this distinction is essential in graph theory and algorithms that utilize tree structures.

6. What is the big-O complexity of the expression represented by the second line on the left?

A. O(n)

B. O(log n)

C. O(2^n)

D. O(n^2)

To determine the big-O complexity of an expression, it is essential to analyze how the time (or space) required for the execution of that expression grows with respect to the input size 'n'. Specifically, when assessing an algorithm's complexity, we focus on the term that grows the fastest as 'n' increases, which often identifies the upper bounds on the growth rate. If the expression in question signifies an exponential growth, particularly in the form of 2 raised to the power of 'n', it indicates that the time or space complexity is indeed O(2^n). Such complexities arise in algorithms that solve problems by evaluating all possible combinations of inputs, particularly seen in recursive algorithms addressing combinatorial issues, like generating power sets or solving the Traveling Salesman Problem via brute force. In general, exponential time complexities are substantially higher and often unmanageable for large inputs, as the number of operations doubles with each additional element. This exceptional growth is what sets O(2^n) apart from polynomial or logarithmic complexities, which grow at a significantly slower rate. Understanding the context—if the expression arises from a recursive function where each call spawns two further calls (like the Fibonacci sequence implementation without memoization)—it supports the notion that the growth of

SAMPLE

7. What does it imply when an algorithm has a space complexity of O(1)?

- A. The algorithm requires minimal space regardless of input size
- B. The space requirement grows linearly with input size
- C. The algorithm cannot execute without additional memory
- D. The algorithm is not efficient

When an algorithm has a space complexity of O(1), it indicates that the algorithm requires a constant amount of memory space regardless of the size of the input. This means that no matter how large or small the input data is, the space consumed by the algorithm remains the same. This property is particularly advantageous because it suggests that the algorithm is efficient in terms of its memory usage, making it suitable for environments with constrained memory resources. For example, if an algorithm processes an array but only uses a fixed number of variables to keep track of indices or sums during its computation, its space complexity would be O(1). This ensures that the algorithm's performance is not hindered by the size of the input data, allowing for predictable and efficient memory usage. In contrast, the other options would imply variable or inefficient space utilization which is not the case with O(1) complexity.

8. Which of the following typically characterizes divide-and-conquer strategies?

- A. They exclusively use iteration
- B. They break a problem into independent smaller problems
- C. They solely rely on brute force methods
- D. They cannot be used for sorting

The correct choice accurately characterizes divide-and-conquer strategies by focusing on how these methods approach problem-solving. Divide-and-conquer involves breaking a larger problem into smaller, independent subproblems that can be solved individually. This is a fundamental goal in many algorithms that employ this strategy, as it allows for a more manageable way to tackle complex problems. For instance, in sorting algorithms like Merge Sort and Quick Sort, the initial problem of sorting a list is divided into smaller segments that are sorted independently. Once each segment is sorted, these smaller solutions are combined to form the overall solution to the original problem. This characteristic enables the efficiency and effectiveness seen in various divide-and-conquer algorithms, often improving performance through parallelism and reducing the time complexity in comparison to solving the problem directly without such a decomposition. In contrast, the other options either describe strategies that do not apply to divide-and-conquer or misunderstand its fundamental nature. Options that suggest exclusive reliance on iteration, brute force methods, or imply limitations on the types of problems that can be solved through this strategy do not align with the essential principles of divide-and-conquer.

SAMPLE

9. What is the total number of fundamental instructions executed by the routine if n = 4?

A. 8

B. 10

C. 12

D. 4 + 2n

The total number of fundamental instructions executed by the routine when \(n = 4 \) can be derived from analyzing the structure of the routine itself. When an algorithm or code snippet is expressed as \(4 + 2n \), it indicates the number of operations depends linearly on the size \(n \). For \(n = 4 \), substituting this value into the expression yields: \[4 + 2(4) = 4 + 8 = 12 \] Thus, when \(n = 4 \), the total number of fundamental instructions executed is 12, confirming that the provided expression accurately captures the pattern of execution within the routine based on varying values of \(n \). This type of expression often arises in algorithms that entail a constant number of operations in addition to a linear scaling with \(n \) — for instance, loops that execute a fixed number of times for each element processed or a combination of sequential and iterative tasks. This clarity in understanding how the formula describes the relationship between the input size and operations is key to analyzing algorithm efficiency.

10. How can you classify a problem as NP-complete?

A. If it has no solution

B. If it can be solved in polynomial time

C. If it is in NP and every problem in NP can be reduced to it in polynomial time

D. If it can be solved using brute force

To classify a problem as NP-complete, it is essential to determine two specific criteria. First, the problem must be in the class NP, which means that any proposed solution can be verified in polynomial time. Second, the problem must satisfy the condition that every problem in NP can be reduced to it in polynomial time. This reduction effectively shows that if we can find a polynomial-time solution for the NP-complete problem, then we can use that solution to solve all problems in NP within polynomial time as well. This classification is crucial because it helps in understanding the complexity of various computational problems. If a new problem is shown to be NP-complete, it can be interpreted as being as hard as the hardest problems in NP, indicating that no polynomial-time solution is known for it, and finding one would be a significant breakthrough in computer science. The other options do not accurately represent the criteria for NP-completeness. For instance, a problem having no solution does not help classify it as NP-complete; in fact, many NP-complete problems have solutions that can be verified. Likewise, a problem that can be solved in polynomial time is classified as being in P, not NP-complete. Finally, any problem solvable by brute force does not

SAMPLE

Next Steps

Congratulations on reaching the final section of this guide. You've taken a meaningful step toward passing your certification exam and advancing your career.

As you continue preparing, remember that consistent practice, review, and self-reflection are key to success. Make time to revisit difficult topics, simulate exam conditions, and track your progress along the way.

If you need help, have suggestions, or want to share feedback, we'd love to hear from you. Reach out to our team at hello@examzify.com.

Or visit your dedicated course page for more study tools and resources:

https://algorithmsanalysis.examzify.com

We wish you the very best on your exam journey. You've got this!

SAMPLE