DESIGN AND ANALYSIS OF ALGORITHMS

DAA – Questions & Detailed Answers

19 Theoretical & Problem-Based Questions

Prepared for B.Tech / AKTU Examination Preparation

Prepared by Dr. Aadarsh Malviya
ECS Department

INDEX

No.	Question / Topic
17	State the height bound of a B-tree of minimum degree t with n keys. Taking t = 3, insert the keys 12, 24, 36, 48, 60, 72, 84, 96, 18 and 30 into an initially empty B-tree. Also state the properties of a B-tree for minimum degree t.
18	Apply the merge sort algorithm and illustrate it by sorting the sequence {38, 27, 43, 3, 9, 82, 10, 15}, showing each divide and merge step. Write the recurrence relation for its running time and solve it using the recursion-tree method to obtain the worst-case time complexity.
19	Describe the standard Trie data structure and construct a Trie by inserting the strings BAT, BATH, BALL, BASE, BAND, BE, BEAT, BEE, BAG and DOG, into an initially empty Trie. State the total number of nodes in the final Trie.

PART II – QUESTIONS AND ANSWERS

Each question is followed by a concise, examination-oriented answer. Diagrams are represented in text where appropriate.

Question 1

What is an Algorithm? Explain the characteristics of a good algorithm.

Answer:

An algorithm is a finite, ordered sequence of unambiguous steps used to solve a problem or produce a required result.

Characteristics of a good algorithm:
1. Input: It accepts zero or more clearly defined inputs.
2. Output: It produces at least one well-defined result.
3. Definiteness: Every step must be precise and unambiguous.
4. Finiteness: It must terminate after a finite number of steps.
5. Effectiveness: Each operation must be basic enough to be performed in finite time.
6. Correctness: It should produce the required result for valid inputs.
7. Efficiency: It should use reasonable time and memory resources.

Question 2

What is the difference between Time Complexity and Space Complexity? Explain with suitable examples.

Answer:

Time complexity measures how the running time of an algorithm grows with input size n. Space complexity measures how much memory the algorithm requires as n grows.

For example, a linear search examines at most n elements, so its worst-case time complexity is O(n). If it uses only a fixed number of extra variables, its auxiliary space complexity is O(1).

Thus, time complexity concerns computational operations, whereas space complexity concerns memory usage. Both are commonly expressed using asymptotic notation.

Question 3

What are Asymptotic Notations? Explain Big-O, Big-Ω (Omega), and Big-Θ (Theta) notations.

Answer:

Asymptotic notation describes the growth of an algorithm's resource usage as n becomes large, ignoring constants and lower-order terms.

Big-O, O(g(n)): gives an asymptotic upper bound. It describes a function that does not grow faster than a constant multiple of g(n), for sufficiently large n.
Big-Ω, Ω(g(n)): gives an asymptotic lower bound.
Big-Θ, Θ(g(n)): gives a tight asymptotic bound when both O(g(n)) and Ω(g(n)) hold.

Example: $f(n)=3n^2+5n+2$ is $O(n^2)$, $\Omega(n^2)$, and therefore $\Theta(n^2)$.

Question 4

Explain the Divide and Conquer technique. Give suitable examples of algorithms based on this technique.

Answer:

Divide and Conquer solves a problem by dividing it into smaller subproblems, solving those subproblems recursively, and combining their results.

Three steps:
1. Divide: Split the problem into smaller instances.
2. Conquer: Solve the smaller instances, usually recursively.
3. Combine: Merge their solutions into the solution of the original problem.

Examples include Merge Sort, Quick Sort, and Binary Search. Merge Sort has recurrence $T(n)=2T(n/2)+\Theta(n)$, giving $\Theta(n \log n)$ time.

Question 5

What is the difference between Greedy Method and Dynamic Programming? Explain with suitable examples.

Answer:

The Greedy method builds a solution step by step by choosing the locally best option at each stage. It is appropriate when the problem has the greedy-choice property and optimal substructure. Examples include Kruskal's algorithm, Prim's algorithm and Dijkstra's algorithm for non-negative edge weights.

Dynamic Programming (DP) solves overlapping subproblems and stores their results to avoid repeated computation. It is useful when the problem has optimal substructure and overlapping subproblems. Examples include 0/1 Knapsack, Matrix Chain Multiplication and Longest Common Subsequence.

The main difference is that greedy algorithms commit to local choices, whereas DP systematically evaluates and stores subproblem solutions.

Question 6

Define time complexity and space complexity of an algorithm.

Answer:

Time complexity is the asymptotic measure of the number of basic operations performed by an algorithm as a function of input size n.

Space complexity is the asymptotic measure of the memory required by an algorithm as a function of n. It may include input storage and auxiliary memory; in algorithm analysis, auxiliary space is often reported separately.

Example: An iterative sum of n numbers takes $\Theta(n)$ time and $\Theta(1)$ auxiliary space.

Question 7

Describe little-oh (o) and little-omega (ω) notations for a function f(n), giving one example of each.

Answer:

Little-o: $f(n)=o(g(n))$ means f grows strictly slower than g. Formally, $\lim(n\to\infty)\ f(n)/g(n)=0$, when the limit exists.

Example: $n = o(n^2)$.

Little-omega: $f(n)=\omega(g(n))$ means f grows strictly faster than g. Formally, $\lim(n\to\infty) f(n)/g(n)=\infty$, when the limit exists.
Example: $n^2 = \omega(n)$.

Unlike O and Ω, little-o and little-ω describe strict asymptotic separation.

Question 8

List the properties of a Binomial Heap and state the maximum number of Binomial Trees it can contain for n nodes.

Answer:

A binomial heap is a collection of binomial trees satisfying the heap-order property. Its important properties are:
1. There is at most one binomial tree of any particular degree.
2. Each binomial tree obeys the chosen heap order, such as min-heap order.
3. A binomial tree Bk contains 2^k nodes and has degree k.
4. The root list is maintained in increasing order of tree degree.

Because the tree sizes correspond to powers of two, the number of binomial trees required for n nodes equals the number of 1s in the binary representation of n. Therefore, the maximum number is $\lfloor \log_2 n \rfloor + 1$.

Question 9

State the best-case and worst-case time complexity of Quick Sort, and name one in-place and one non-in-place sorting algorithm.

Answer:

For Quick Sort, the best-case time complexity is $\Theta(n \log n)$, when partitions are reasonably balanced. The worst-case complexity is $\Theta(n^2)$, for example when the pivot repeatedly produces partitions of sizes 0 and $n-1$.

An example of an in-place sorting algorithm is Quick Sort (standard array partitioning uses O(log n) stack space on balanced recursion). An example of a generally non-in-place sorting algorithm is Merge Sort, which normally requires $\Theta(n)$ auxiliary array space.

Question 10

Arrange the following functions in increasing order of growth rate and justify the ordering: $f_1(n) = n \log n$, $f_2(n) = 2^n$, $f_3(n) = n^{1.5}$, $f_4(n) = \log_2 n$, $f_5(n) = n^2$ and $f_6(n) = \sqrt{n}$. Explain how the asymptotic growth rate of two functions are compared with the help of diagrammatic representation.

Answer:

The increasing order is:

$\log_2 n < \sqrt{n} < n \log n < n^{1.5} < n^2 < 2^n$.

The justification follows standard asymptotic growth classes: logarithmic functions grow more slowly than polynomial functions; among polynomials, the smaller exponent grows more slowly; exponential

2^n eventually grows faster than every fixed-degree polynomial.

To compare two functions diagrammatically, plot their values against n on the same coordinate system. For sufficiently large n, the function whose curve rises faster has the higher asymptotic growth rate. A log-log plot can also be useful for comparing polynomial growth, while normalized ratios such as f(n)/g(n) provide a mathematical comparison.

Question 11

Justify how the performance of an algorithm is analysed in the best, average and worst cases with a suitable example. Also determine the upper bound, the lower bound and the tight bound for the function $f(n) = 5n^2 + 3n + 7$.

Answer:

The best case is the minimum running time over inputs of size n; the worst case is the maximum; the average case describes expected performance under a specified input distribution.

Example: In linear search, the best case is Θ(1) when the item is first; the worst case is Θ(n) when it is last or absent; the average case is Θ(n) under a standard uniform-position assumption.

For $f(n)=5n^2+3n+7$, the highest-order term is $5n^2$. Therefore:
- Upper bound: $O(n^2)$
- Lower bound: $\Omega(n^2)$
- Tight bound: $\Theta(n^2)$.

Question 12

Write the pseudo-code of the Heap Sort algorithm and its time complexities, and illustrate the operation of Heap Sort on the array A = {19, 4, 31, 12, 8, 27, 45, 6}. Explain what is meant by an in-place sorting algorithm.

Answer:

Heap Sort (ascending order) using a max-heap:

```
BUILD-MAX-HEAP(A)
for i = n downto 2:
  swap A[1] and A[i]
  heap_size = heap_size − 1
  MAX-HEAPIFY(A,1)
```

BUILD-MAX-HEAP takes Θ(n); each extraction takes Θ(log n), and there are n−1 extractions. Hence best, average and worst-case time are all Θ(n log n). Standard Heap Sort uses O(1) auxiliary array space.

For A={19,4,31,12,8,27,45,6}, the max-heap after building can be represented as: {45,12,31,6,8,27,19,4}.
Repeatedly moving the maximum to the end and restoring the heap produces the sorted array: {4,6,8,12,19,27,31,45}.

An in-place sorting algorithm rearranges elements using only a small amount of extra memory beyond the input array.

Question 13

Explain the structure of a Binomial Heap and illustrate, for n = 13, how the binary representation of n determines the binomial trees present in the heap. Also differentiate between a Binomial Heap and a Fibonacci Heap with respect to any three operations.

Answer:

A binomial heap is a collection of heap-ordered binomial trees with at most one tree of each degree. A Bk tree has 2^k nodes.

For n=13, binary representation is 1101_2 = 8+4+1. Therefore the heap contains one each of B3, B2 and B0 trees, with 8, 4 and 1 nodes respectively.

Comparison:
• Insert: Binomial Heap O(log n) worst case; Fibonacci Heap O(1) amortized.
• Find-Min: Binomial Heap O(log n) if the roots are scanned; Fibonacci Heap O(1) amortized.
• Decrease-Key: Binomial Heap O(log n); Fibonacci Heap O(1) amortized.

Fibonacci Heap achieves these amortized bounds through lazy consolidation and cascading cuts.

Question 14

Solve the following recurrence relations and obtain the time complexity in each case: (i) $T(n) = 8T(n/2) + n^2$ using the master's theorem; (ii) $T(n) = T(n – 1) + n^3$ using the substitution method. Hence arrange the two complexities so obtained in increasing order of growth rate.

Answer:

(i) $T(n)=8T(n/2)+n^2$
Here a=8, b=2, so n^(log_b a)=n^3. Since $f(n)=n^2$ is polynomially smaller than n^3, Master Theorem Case 1 applies. Therefore $T(n)=\Theta(n^3)$.

(ii) $T(n)=T(n-1)+n^3$
Expanding gives $T(n)=T(n-2)+(n-1)^3+n^3$, and continuing gives $T(n)=T(1)+\Sigma(k^3)$, k=2..n. Since $\Sigma k^3=\Theta(n^4)$, $T(n)=\Theta(n^4)$.

Increasing order: $\Theta(n^3) < \Theta(n^4)$.

Question 15

Let Q be a Quick Sort program that sorts numbers in ascending order using the last element as the pivot. Let t_1 and t_2 be the number of comparisons made by Q for the inputs {5, 4, 3, 2, 1} and {3, 5, 1, 4, 2} respectively. Find t_1 and t_2. Also write the pseudo-code of Quick Sort and the partition procedure, and state its recurrence relation.

Answer:

Using the standard Lomuto partition scheme, each partition makes one comparison for every element other than the pivot.

For {5,4,3,2,1}, the last element is repeatedly the smallest element. The comparison counts are 4+3+2+1 = t_1=10.

For {3,5,1,4,2}: first pivot 2 gives 4 comparisons and leaves {1} and {3,5,4}; then pivot 4 gives 2 comparisons and leaves {3} and {5}. Thus t_2=4+2=6.

```
QuickSort(A,p,r):
if p < r:
  q = Partition(A,p,r)
  QuickSort(A,p,q−1)
  QuickSort(A,q+1,r)
```

Partition: choose A[r] as pivot; scan j from p to r−1; whenever A[j] ≤ pivot, increase i and exchange A[i] and A[j]; finally exchange A[i+1] and A[r], and return i+1.

Balanced recurrence: $T(n)=2T(n/2)+\Theta(n)$; worst case: $T(n)=T(n-1)+\Theta(n)$.

Question 16

Explain the cases involved in insertion operations in Red-Black tree. Applying these cases, insert the keys 61,58,51,32,39,29 into an initially empty Red-Black tree.

Answer:

Red-Black insertion starts by inserting the new key as a red node. If its parent is black, no violation occurs. If the parent is red, the following cases are considered:
1. Uncle is red: recolor parent and uncle black and grandparent red, then continue upward.
2. Uncle is black and node forms a triangle: rotate at the parent to convert it to the line case.
3. Uncle is black and node forms a line: rotate at the grandparent and recolor parent/grandparent appropriately.
The root is always recolored black at the end.

For the sequence 61,58,51,32,39,29, applying standard insertion fix-up gives the final tree:

```
    39(B)
  /     \
32(R)    58(R)
/  \     /  \
29(B) —    51(B) 61(B)
```

Here B denotes black and R denotes red. The tree satisfies the Red-Black properties.

Question 17

State the height bound of a B-tree of minimum degree t with n keys. Taking t = 3, insert the keys 12, 24, 36, 48, 60, 72, 84, 96, 18 and 30 into an initially empty B-tree. Also state the properties of a B-tree for minimum degree t.

Answer:

For a B-Tree of minimum degree t, the height h satisfies an upper bound of approximately O(log_t n). More precisely, for n≥1, h ≤ log_t((n+1)/2) for the standard minimum-occupancy bound.

For t=3, every non-root node contains at least 2 and at most 5 keys. A node can therefore have between 3 and 6 children, except the root.

Inserting 12,24,36,48,60,72,84,96,18,30 using standard B-Tree insertion gives a valid final tree:

```
   [36]
  /     \
```

[12,18,24,30] [48,60,72,84,96]

Properties: keys in each node are sorted; all leaves are at the same depth; each non-root node has between t−1 and 2t−1 keys; each internal node has between t and 2t children; and the root has at least one key and, when non-leaf, at least two children.

Question 18

Apply the merge sort algorithm and illustrate it by sorting the sequence {38, 27, 43, 3, 9, 82, 10, 15}, showing each divide and merge step. Write the recurrence relation for its running time and solve it using the recursion-tree method to obtain the worst-case time complexity.

Answer:

Divide steps:
{38,27,43,3,9,82,10,15}
→ {38,27,43,3} and {9,82,10,15}
→ {38,27},{43,3} and {9,82},{10,15}
→ {38},{27},{43},{3},{9},{82},{10},{15}

Merge steps:
{38}+{27} → {27,38}
{43}+{3} → {3,43}
{27,38}+{3,43} → {3,27,38,43}
{9}+{82} → {9,82}
{10}+{15} → {10,15}
{9,82}+{10,15} → {9,10,15,82}
Final merge → {3,9,10,15,27,38,43,82}.

The recurrence is $T(n)=2T(n/2)+\Theta(n)$, with $T(1)=\Theta(1)$. A recursion tree has $\log_2 n$ levels, each contributing $\Theta(n)$, so total cost is $\Theta(n \log n)$. Therefore worst-case time complexity is $\Theta(n \log n)$.

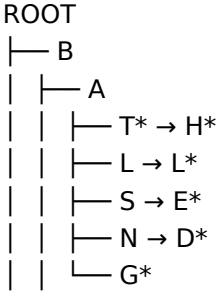
Question 19

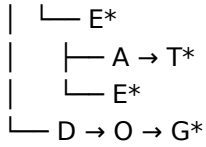
Describe the standard Trie data structure and construct a Trie by inserting the strings BAT, BATH, BALL, BASE, BAND, BE, BEAT, BEE, BAG and DOG, into an initially empty Trie. State the total number of nodes in the final Trie.

Answer:

A Trie is a tree-based data structure used to store strings by sharing common prefixes. Each edge represents a character, and a terminal marker identifies the end of a stored word.

The strings share prefixes such as BA and BE. A compact representation is:





Here * marks the end of a word. Counting the root and every distinct prefix node gives 25 nodes in the final Trie.