CS2209 – DESIGN AND ANALYSIS OF ALGORITHM (DAA) TEST II – MAY 2025

PART A (7 x 2 = 14 Marks)

1. What is an algorithm? (Reference unit page 3**)**
An algorithm is a **finite set of well-defined steps** that provides a solution to a specific problem. It must be:

- Clear and unambiguous
- Finite and complete
- Effective and logically sequenced

2. List any four characteristics that define a good algorithm?

1. **Correctness** – Produces expected output.
2. **Efficiency** – Optimal time and space usage.
3. **Definiteness** – Each step is clearly defined.
4. **Finiteness** – Terminates after a finite number of steps.

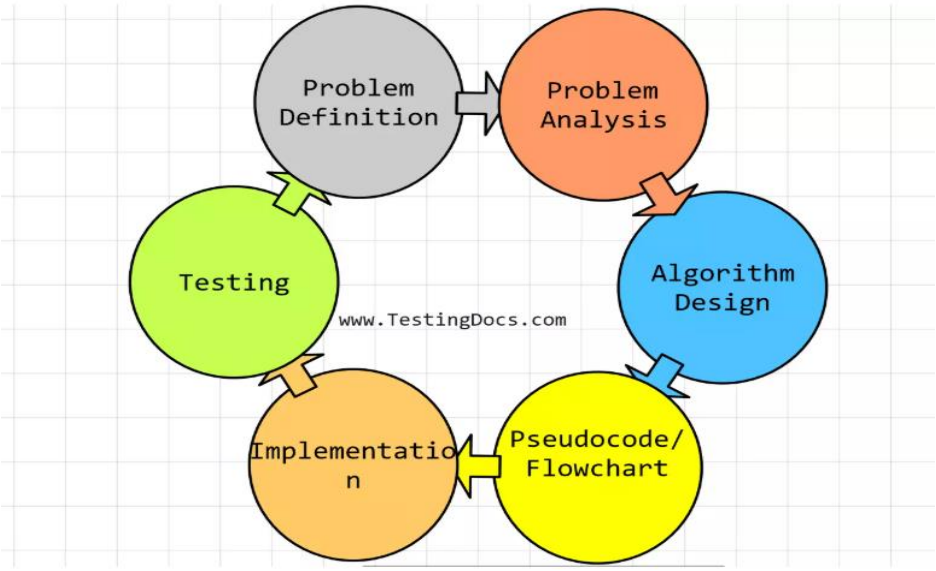
3. Define brute force algorithm.
A brute force algorithm solves problems by **trying all possible solutions** and choosing the best one.

- Simple to implement
- Often inefficient for large datasets
- Example: Linear Search, Checking all subsets in Knapsack

4. List six fundamental steps of algorithmic problem solving. (Reference ya page 6 and 7)

The general steps involved in algorithm development are as follows: -

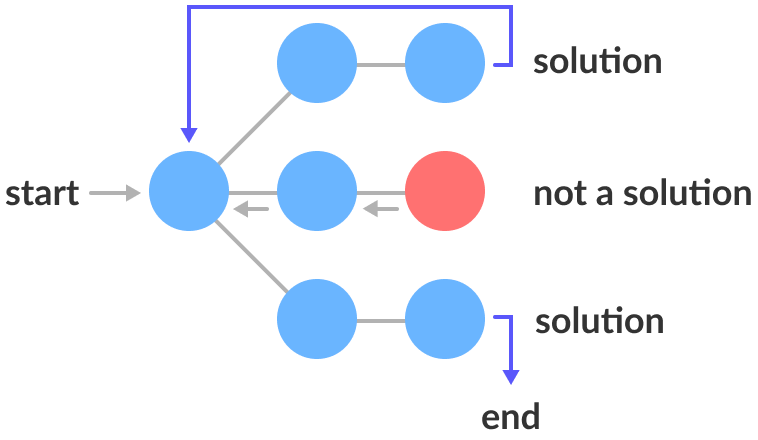
1. Problem Definition
2. Problem Analysis
3. Algorithm Design
4. Flowchart
5. Implementation
6. Testing
7. Documentation



5. **What is backtracking algorithm?**

   Backtracking is a **recursive problem-solving strategy** that explores all possibilities by building a solution incrementally, and **removing solutions that fail** (backtrack).

- Used in puzzles, permutations, n-Queens, Sudoku.



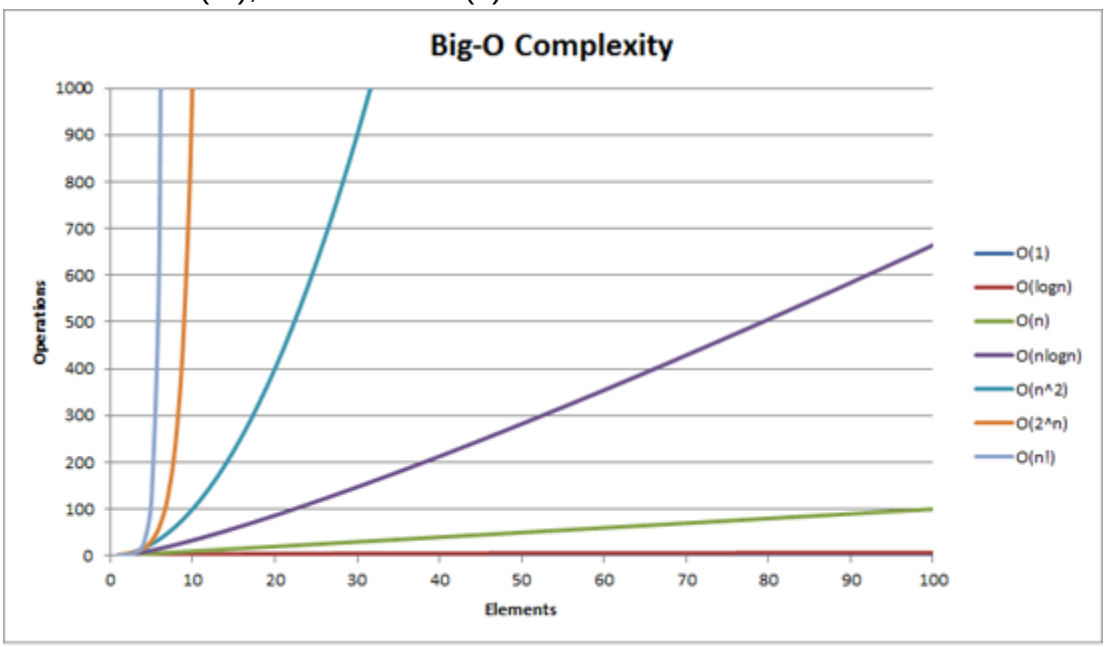
6. Mention four problem types in the realm of algorithmic problem solving. (reference ya mwalimu page 14 unit one)

1. **Search Problems** – Find items in a dataset
2. **Optimization Problems** – Maximize or minimize a function
3. **Decision Problems** – Yes/No based problems
4. **Sorting Problems** – Rearrange elements in a dataset
5. **Graphs Problems** – Deal with graphs example Dijkstra's algorithm

7. What do you understand by O(n log n)?

It represents the **asymptotic time complexity** of an algorithm, meaning its performance grows in proportion to **n × log(n)**.

- Used to describe average complexity of Quick Sort, Merge Sort
- Faster than $O(n^2)$, slower than O(n)



PART B (3 x 4 = 12 Marks)

8(a). Discuss the typical stages involved in the functioning of an algorithm. (Reference unit 1 page 4 and 5**)**

1. **Input:** Accept input data
2. **Processing:** Perform computation using logic
    1. **Decision-making:** Use conditionals
    2. **Looping (Iteration/Recursion):** Looping constructs
3. **Output:** Generate and return result
4. **Termination :** Ending the program

OR

8(b). Discuss the advantages and disadvantages of recursive algorithms.

(reference ya mwalimu 4 and 5)
Advantages:

- **Reduced Time Complexity:**
- **Efficient Tree Traversal:**
- **Improved Readability:**
- **allows breaking down large problems into smaller:**
- Elegant and simple to write
- Matches natural problem definitions (e.g., factorial, tree traversal)
- Reduces complex loops
-

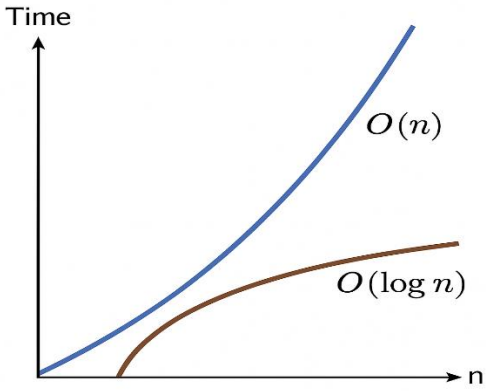
Disadvantages:

- Higher memory usage (stack)
- Slower due to overhead
- Risk of stack overflow for deep recursion

9(a). Compare and contrast linear and binary search algorithms (reference ya madam unit 54 and 65 **)**

Feature	Linear Search	Binary Search
Data	Works on any dataset	Requires sorted dataset
Time Complexity	O(n)	O(log n)
Technique	Sequential checking	Divide and conquer
Use Case	Small or unsorted data	Large sorted datasets

Feature Linear Search Binary Search



OR

9(b). Explain the importance and steps of empirical analysis of algorithms. (reference ya madam 48 **)**

Empirical analysis involves **testing an algorithm** using real data to evaluate:

- Actual execution time
- Memory consumption
- Performance trends across input sizes

Steps:

- Define Performance Metrics:
  - Determine what aspects of the algorithm's performance you want to measure (e.g., execution time, memory usage, number of operations).
- Choose Representative Inputs:
  - Select a set of inputs that are representative of the range of data the algorithm will be used with in practice.
- Implement the Algorithm:
  - Implement the algorithm in a programming language and on a specific computer or platform.
- Run Experiments:
  - Run the algorithm on the chosen inputs and measure its performance using the defined metrics.
- Analyze Results:
  - Analyze the data collected to identify trends, patterns, and insights into the algorithm's performance.
- Draw Conclusions:

- Based on the analysis, draw conclusions about the algorithm's strengths, weaknesses, and suitability for the intended application

Importance of empirical analysis:

1. Practical Performance Insights:
2. Identification of Bottlenecks:
3. Comparison of Algorithms:
4. Optimization:

10(a). Explain the role of the call stack, mathematical analysis of time and space complexity in recursive algorithms.

- The **call stack** tracks function calls, parameters, return points
- Each recursive call pushes data to the stack
- **Time complexity**: Depends on number of recursive calls
- **Space complexity**: Includes stack space + variables
- Example: Factorial – O(n) time and space

OR

10(b). Describe the process of converting a recursive algorithm to a non-recursive version. (Reference page 21 unit 2 **)**

1. **Use an explicit stack** to replace the call stack
2. Convert function calls into loop constructs
3. Simulate state transitions manually

- Example: Convert DFS from recursive to iterative using stack
- Ensures memory efficiency and avoids stack overflow

1. Identify the Recursive Structure:
2. Replace Recursion with Iteration:
3. Handle the Call Stack Explicitly (if needed):
4. Optimize for Space:

PART C (2 x 12 = 24 Marks)

11(a). Discuss the importance of analyzing algorithm efficiency.

- Helps in selecting the **best solution** among alternatives
- Reduces resource usage (CPU time, memory)
- Ensures **scalability** for large inputs
- Supports real-time system performance guarantees

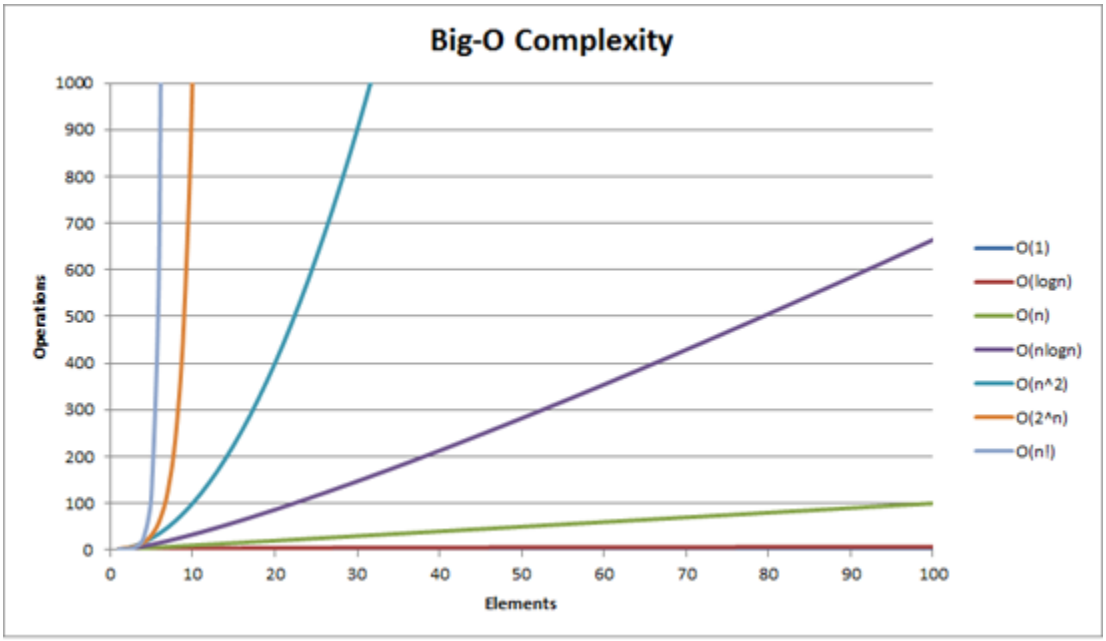
OR

11(b). Explain the broad concept of Big O(n) notation as applied in Algorithm analysis and design. Support your answers with graphs.

- **Big O notation** expresses worst-case time or space complexity
- Shows how performance grows with input size

Complexity	Growth Trend	
O(1)	Accessing array element	Constant
O(log n)	Binary search	Slow growth
O(n)	Linear search	Linear growth
O(n log n)	Merge sort	Faster than $O(n^2)$
$O(n^2)$	Bubble sort	Quadratic

Graphs can visually represent curve trends of each Big O function.



12(a). Describe the different types of algorithms based on their strategies.

1. **Brute Force:** Try all possibilities (e.g., linear search)
2. **Divide and Conquer:** Divide problem recursively (e.g., merge sort)
3. **Greedy:** Make best local choice (e.g., Prim's algorithm)

4. **Dynamic Programming:** Use overlapping subproblems (e.g., Fibonacci)
5. **Backtracking:** Reject infeasible paths (e.g., n-Queens)
6. **Branch and Bound:** Eliminate subsets using bounds (e.g., TSP)
7. **Backtracking:** Reject infeasible paths (e.g., n-Queens)
8. **Recursive algorithms: algorithms call themselves**
9. **Randomized algorithms**. Randomized algorithms use random numbers
10.

OR

12(b). Write C program to execute a binary search algorithm.

```
#include <stdio.h>


int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == key) return mid;
        else if (arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}


int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int key = 30;
    int result = binarySearch(arr, 5, key);
    if (result != -1) printf("Found at index %d", result);
    else printf("Not found");
    return 0;
}
```

CS2209 – DESIGN AND ANALYSIS OF ALGORITHM (DAA)

DEGREE CONTINUOUS ASSESSMENT TEST – I, JUNE 2025
TOTAL MARKS: 50 | DURATION: 2 HOURS

PART A (7 x 2 = 14 Marks)

1. **What is the key difference between linear search and binary search?**

(reference ya mwalimu , page ya

Feature	Linear **Search**	**Binary Search**
Data Requirement	Works on **unsorted** or sorted data	Requires **sorted** data
Time Complexity	O(n) – checks every element	O(log n) – divides list and eliminates half
Search Strategy	Sequential search	Divide and conquer strategy

Binary Search is significantly more efficient for large, sorted datasets, whereas Linear Search is suitable for small or unsorted lists.

2. Which sorting algorithm uses the divide and conquer technique?

- **Merge Sort**, **Quick Sort**, and **Heap Sort** use the divide and conquer technique.
- **Merge Sort** explicitly divides, sorts recursively, and merges results.

3. Which searching algorithm is inefficient for large datasets and why?

- **Linear Search** is inefficient because it checks each item one by one — time complexity is O(n), which becomes slow for large datasets.

4. What is the worst-case time complexity of Quick Sort and when does it occur?

(reference page 38 unit 3)

- **Worst-case:** $O(n^2)$
- Occurs when the pivot chosen is always the smallest or largest element (already sorted or reverse sorted arrays).

5. State the n-Queen's problem.

- Place n queens on an n×n chessboard such that no two queens threaten each other (no two in same row, column, or diagonal).

The **n-Queen's problem** is a classic computational puzzle that involves placing **n queens** on an **n × n chessboard** in such a way that **no two queens attack each other**. This means:

- No two queens share the same **row**.
- No two queens share the same **column**.
- No two queens share the same **diagonal**.

It is widely used in **algorithm design** to demonstrate **backtracking**, **constraint satisfaction**, and **combinatorial optimization**. The complexity increases exponentially with *n*, making it a valuable example for understanding the power of pruning and intelligent state-space exploration.

6. What is the Knapsack problem?

- An optimization problem where you must choose a subset of items with given weights and values to maximize value within a fixed weight capacity.

7. What is the Traveling Salesman Problem (TSP)?

- Find the shortest possible route that visits each city once and returns to the origin city — a classic NP-hard problem in optimization.

PART B (3 x 4 = 12 Marks)

8(a). What are the main differences between stable and unstable sorting algorithms?

Feature	Stable Sorting	Unstable Sorting
Definition	Maintains relative order of equal elements	May change order of equal elements
Example	Merge Sort, Insertion Sort	Quick Sort, Heap Sort
Use Case	When data has duplicate keys	When order doesn't matter

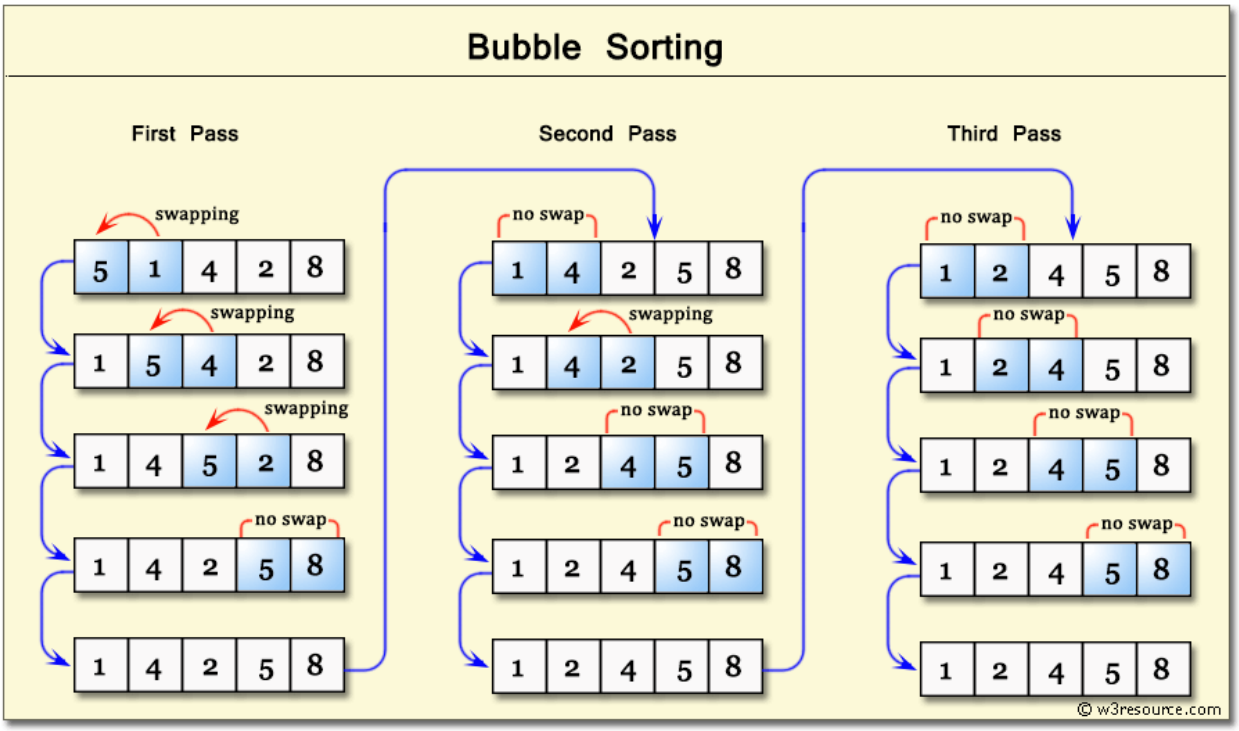
OR

8(b). In what scenario is Linear Search preferable over Binary Search?

- When the data is **unsorted**.

- When dataset is **small**.
- When no need for additional memory.
- When **requirements works on any data type**

9(a). Explain the working principle of Bubble Sort.

- Repeatedly compares adjacent elements and swaps them if they are in wrong order.
- After each pass, the largest element 'bubbles up' to the end.
- Best case: O(n) (already sorted); Worst: $O(n^2)$
- 

OR

9(b). How do approximation algorithms help in solving NP-Hard problems?

- NP-Hard problems are intractable for large inputs.
- Approximation algorithms provide **near-optimal** solutions in **polynomial time**.
- Example: TSP, Knapsack, Vertex Cover.
- They guarantee solution within a **provable bound** from optimal.

10(a). Explain how backtracking is used to solve the n-Queen's problem.

- Place a queen row-by-row, checking column and diagonal safety.
- If a conflict arises, backtrack and place queen in a new column.
- It prunes infeasible branches, hence improves search.

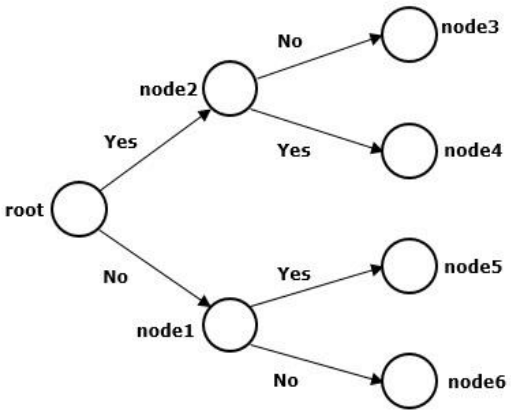
OR

10(b). Explain how the Branch and Bound technique improves efficiency compared to brute force. Provide an example.

The **branch and bound algorithm** is a technique used for solving combinatorial optimization problems. First, this algorithm breaks the given problem into multiple sub-problems and then using a bounding function, it eliminates only those solutions that cannot provide optimal solution.

Combinatorial optimization problems refer to those problems that involve finding the best solution from a finite set of possible solutions, such as the travelling salesman problem and many more.

- Uses **bounds** to eliminate large subsets of solutions.
- Builds solution space as a **tree**, prunes nodes that cannot yield better solution.
- Example: Solving **TSP** using least cost branch and bound.
- Reduces time complexity vs exhaustive search.



PART C (2 x 12 = 24 Marks)

11(a). Develop an algorithm for Binary Search.

```
Algorithm BinarySearch(arr, key):
  1. Set low = 0, high = len(arr) - 1
  2. While low <= high:
     a. mid = (low + high) // 2
     b. If arr[mid] == key: return mid
     c. If arr[mid] > key: high = mid - 1
     d. Else: low = mid + 1
  3. Return -1 (not found)
```

Time Complexity: O(log n), only applicable on sorted data.

```
#include <stdio.h>

int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == key) return mid;
        else if (arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int key = 30;
    int result = binarySearch(arr, 5, key);
    if (result != -1) printf("Found at index %d", result);
    else printf("Not found");
    return 0;
}
```

OR

11(b). Design an algorithm to merge two sorted arrays (used in Merge Sort).

```
Algorithm MergeArrays(A, B):
  1. Initialize result = []
  2. While both A and B not empty:
     a. Compare front of A and B
     b. Append smaller to result and remove from that array
  3. Append remaining elements from A or B
```

Used in Merge Sort to combine two sorted halves into a sorted whole.

Code sample:

```
#include <stdio.h>
void merge(int A[], int B[], int n, int m, int R[]) {
   int i=0, j=0, k=0;
   while (i<n && j<m)
      R[k++] = (A[i] < B[j]) ? A[i++] : B[j++];
   while (i<n) R[k++] = A[i++];
   while (j<m) R[k++] = B[j++];
}
int main() {
   int A[] = {1, 4, 7}, B[] = {2, 3, 6}, R[6];
   merge(A, B, 3, 3, R);
   for (int i = 0; i < 6; i++) printf("%d ", R[i]);
   return 0;
}
```

12(a). How does Quick Sort differ from Merge Sort in terms of space and performance?

Feature	Quick Sort	Merge Sort
Approach	Divide & Conquer, No-order	Divide & Conquer, In- order
Time (Avg)	O(n log n)	O(n log n)
Time (Worst)	$O(n^2)$	O(n log n)
Space	O(log n) stack	O(n) extra memory
Use Case	Preferred for Unstable sort	Stable sort for linked lists

OR

12(b). Explain the working of Heap Sort algorithm in detail.

- Build a **max heap** from input array.
- Swap root (max) with last element.
- Reduce heap size and **heapify** root.
- Repeat until all elements are sorted.

```
Time Complexity: O(n log n)
Space Complexity: O(1) (in-place)
Stable: No
```

Efficient for large datasets; guarantees O(n log n) even in worst case.
