Competitive Programming Approaches

Can you please teach me these approaches of Competitive programming for my viva, with a few examples(only names of examples).
Teach in such a way, that I understand well and am able to explain nicely in a crisp concise manner.
Don't use emojis

Brute Force
Greedy Algorithms
Divide and Conquer
Dynamic Programming (DP)
Backtracking
Recursion

Sure, Ritesh. Here's a clean, crisp explanation of each approach that you can use for your viva. I'll explain each one in a way that makes it easy to remember and explain, along with a few example problems (just the names).

---

1. Brute Force

Concept:
Try all possible solutions and pick the best one. It's simple but inefficient for large inputs.

When to use:
When the input size is small or optimization isn't needed.

Example Problems:

- Generate all subsets of an array
- Check all permutations to solve the Traveling Salesman Problem
- Count pairs with a given sum in an array

How to explain:
"I try every possible way to solve the problem and select the one that works. It's slow, but it always works for small inputs."

---

2. Greedy Algorithms

Concept:
Make the best choice at each step, hoping it leads to the global optimum.

When to use:
When a local optimal choice leads to the global optimal solution. Usually proven by problem properties (like greedy-choice property and optimal substructure).

Example Problems:

- Activity Selection
- Fractional Knapsack
- Huffman Encoding

How to explain:
"I pick the best option available at every step. If the problem allows it, this gives the overall best solution efficiently."

---

3. Divide and Conquer

Concept:
Break the problem into smaller parts, solve them independently, and combine their results.

When to use:
When the problem can be divided into smaller subproblems of the same type.

Example Problems:

- Merge Sort
- Quick Sort
- Binary Search

How to explain:
"I divide the problem into parts, solve each one, and then merge the answers. This reduces the size of the problem step-by-step."

---

4. Dynamic Programming (DP)

Concept:
Break the problem into overlapping subproblems and store the results to avoid recomputation.

When to use:
When the problem has **optimal substructure** and **overlapping subproblems**.

Example Problems:

- Fibonacci Numbers
- Longest Common Subsequence
- 0/1 Knapsack

How to explain:
"I store solutions to smaller problems and reuse them. This saves time and avoids solving the same thing again."

---

5. Backtracking

Concept:
Build the solution step-by-step and undo (backtrack) when a wrong choice is found.

When to use:
When exploring all possible combinations or paths is necessary, but pruning invalid paths early is possible.

Example Problems:

- N-Queens Problem
- Sudoku Solver
- Subset Sum

How to explain:
"I explore all valid choices, and if something doesn't work, I backtrack and try a different path."

---

6. Recursion

Concept:
A function calls itself to solve a problem by breaking it into subproblems.

When to use:
When a problem can be defined in terms of smaller instances of itself.

Example Problems:

- Factorial Calculation
- Tower of Hanoi
- Tree Traversals (Inorder, Preorder, Postorder)

How to explain:
"I solve a big problem by solving smaller instances of the same problem using function calls."

Let me know if you want me to create a one-page summary or viva notes for quick revision.
Do you like this personality