Sample 1:

Input	Output
4	1 1 1 1
1 2 3 4	4 3 2 1
4	1 1 3 2 1 1
1 -5 1 -5	
6	
-5 -1 -1 2 -2 -3	

Explanation:

Example case 1. No two elements have different signs, so any alternating sub array may only consist of a single number.

Example case 2. Every sub array is alternating.

Example case 3. The only alternating sub array of length 3 is A3..5.

- **Procedure/Program:**

```
#include <stdio.h>
int main(){
  int T;
  scanf("%d", &T);
  while (T--){
    int N;
    scanf("%d", &N);
    int A[N], result[N];
    for(int i=0; i<N; i++){
      scanf("%d", &A[i]); }
```

```
result [N-1] = 1;

for(int i = N-2;  i>=0;  i--) {
   if(A[i] * A[i+1] < 0) {

      result [i] = result[i+1] + 1;

        }else{
        result [i]= 1;
    }
 }

for(int i=0;  i<N;  i++) {

     Printf("%d",  result [i]);
      }
   Printf ("\n");

 }
return 0;

}
```

- **Data and Results:**

Data: Array of integers and number of test cases provided

Result: Lengths of longest alternating subarrays starting from each index

- **Analysis and Inferences:**

Analysis: Backtracking checks each element to compute alternating subarray lengths efficiently

In-Lab:

Problem Statement: Real-World Scenario Utilizing Backtracking **(Sum of Subsets)**

Problem: Optimal Packing in Logistics

1) In logistics and transportation, optimizing the packing of goods in containers or vehicles to utilize available space efficiently is critical. The problem is akin to the "sum of subsets" where you aim to find subsets of items whose total weight or volume equals a target value, ensuring the most efficient use of available capacity.

- **Procedure/Program:**

Scenario

efficiently pack goods into containers or vehicles, akin to the "sum of subsets" Problem

Objective

Find item schould subsets that equal a target weight or volume.

Approach: Backtracking

1) Input

- List of item weights/volumes
- Target weight/volume

2) Recursive Function

- Base case: If current sum equals target, return subset

- Include/exclude: Decide to include or exclude each item

3) Output

- valid subsets matching the target.

Example 2

- Items: [2, 3, 5, 7]
- Target: 10
- valid subsets : [3, 7], [5, 2, 3]

complexity 2

- Time: exponential
- space: O(n) for recursion.

- **Data and Results:**

Data 2

list of item weights: [2, 3, 5, 7]

Target: 10

Result 2 : valid subsets found: [3, 7]

[2, 5, 3]

- **Analysis and Inferences:**

Analysis

Backtracking efficiently identified all combinations matching the target

inferences

optimal packing improves logistics and resource utilization significantly.

Experiment #10		Student ID	
Date		Student Name	

Post-Lab:

Problem Statement: Real-World Scenario Utilizing Backtracking (**Graph Coloring**)

Problem: Scheduling Examinations in Educational Institutes

Scenario:

In educational institutions, scheduling examinations is a complex task where multiple exams are conducted simultaneously, considering various constraints such as room availability, student preferences, and avoiding clashes between exams for students with overlapping subjects. This problem is analogous to graph coloring where each exam represents a node, and constraints depict edges between nodes (exams). Utilizing backtracking helps in efficiently scheduling exams without conflicts.

- **Procedure/Program:**

scenario

scheduling multiple exams simultaneously, considering room availability, student preferences and avoiding conflicts

objective

- schedule exams to prevent clashes for students with overlapping subjects

Approach: Backtracking (graph coloring)

1) Input:
   - List of exams (nodes)
   - constraints (edges) indicating conflicts

between exams.

2) Recursive Function

- Base case: if all exams are schedvled without conflicts, return the schedule.
- Assign colors: Try to assign a colour (time slot) to each exam without conflicts
- Backtrack: If a conflict occurs, backtrack and try the next option.

3) output

- A valid schedule of exams with no overlaps

example

- exams: A, B, C, D
- constraints: A-B, A-C (indicating A conflicts with B and C)

complexity

- Time: exponential in the worst case

space: O(V) for the recursion stack, where V is the number of exams.

Experiment #10		Student ID	
Date		Student Name	

- **Data and Results:**

Data:

Exams: A, B, C, D

constraints: A-B, A-C

Result

valid schedule: A at time 1, B at time 2, C at time 3

- **Analysis and Inferences:**

Analysis

Backtracking effectively handled scheduling, ensuring no conflicts occurred

inferences

efficient scheduling enhances academic management and student satisfaction significantly.

- **Sample VIVA-VOCE Questions (In-Lab):**

1. What is the Eight Queens problem?
2. Why is the Eight Queens problem significant in computer science?
3. What is backtracking in the context of algorithm design?
4. How does backtracking help in solving the Eight Queens problem?
5. Can you explain the difference between backtracking and brute force?

Evaluator Remark (if Any):	Marks Secured:___out of 50
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

1) Place 8 queens on chess board, no two attacking each other

2) significant for illustrating backtracking and constraint satisfaction problems.

3) Back tracking explores possible solutions backtracks upon valid configurations

4) Back tracking tries positions, retracts when queens conflict, find solutions

5) Backtracking prunes search space; brute force explores, all possibilities.