Assignment Solution: Recursion -3

Write a program to find the minimum element of an array using recursion.
Ans:

```
#include <iostream>
#include <climits>
using namespace std;
// Recursive function to find the minimum element in an array
int findMin(int arr[], int n) {
// Base case: If array has only one element
if (n == 1) {
return arr[0];
}
// Recursive case: Find the minimum of the rest of the array
int restMin = findMin(arr + 1, n - 1);
// Return the minimum of the first element and the minimum of the rest
return (arr[0] < restMin) ? arr[0] : restMin;
}
int main() {
int n;
cout << "Enter the number of elements in the array: ";
cin >> n;
int arr[n];
cout << "Enter the elements of the array: ";
for (int i = 0; i < n; i++) {
cin >> arr[i];
}
int minElement = findMin(arr, n);
cout << "The minimum element in the array is: " << minElement << endl;

return 0;
```

Write a program to find the sum of all elements in an array using recursion.
Ans:

```
#include <iostream>
using namespace std;
// Recursive function to find the sum of elements in an array
int sumArray(int arr[], int n) {
// Base case: If array has no elements, return 0
if (n <= 0) {
return 0;
}
// Recursive case: Sum the first element and the sum of the rest of the array
return arr[n - 1] + sumArray(arr, n - 1);
}
```

```
int main() {
int n;
cout << "Enter the number of elements in the array: ";
cin >> n;
int arr[n];
cout << "Enter the elements of the array: ";
for (int i = 0; i < n; i++) {
cin >> arr[i];
}
int sum = sumArray(arr, n);
cout << "The sum of the elements in the array is: " << sum << endl;
return 0;
```

Write a program to find the index of a given element in an array using recursion. If the element is present, return its index; otherwise, return -1.
Ans:

```
#include <iostream>
using namespace std;
// Recursive function to find the index of a given element in the array

int findIndex(int arr[], int start, int n, int target) {
// Base case: If start index is out of bounds, return -1
if (start >= n) {
return -1;
}
// If the target element is found, return the current index
if (arr[start] == target) {
return start;
}
// Recursive case: Search in the remaining part of the array
return findIndex(arr, start + 1, n, target);
}
int main() {
int n, target;
cout << "Enter the number of elements in the array: ";
cin >> n;
int arr[n];
cout << "Enter the elements of the array: ";
for (int i = 0; i < n; i++) {
cin >> arr[i];
}
cout << "Enter the element to find: ";
cin >> target;
int index = findIndex(arr, 0, n, target);
if (index != -1) {
cout << "Element " << target << " found at index: " << index << endl;
} else {
```

```
cout << "Element " << target << " is not present in the array." << endl;
}
return 0;
;
```

Print all the elements of an array in reverse order using recursion
Ans:

```
#include <iostream>
using namespace std;
// Recursive function to print the elements of the array in reverse order
void printReverse(int arr[], int start, int n) {

// Base case: If start index is out of bounds, return
if (start >= n) {
return;
}
// Recursive call: Print the reverse of the remaining array elements
printReverse(arr, start + 1, n);
// Print the current element after the recursive call
cout << arr[start] << " ";
}
int main() {
int n;
cout << "Enter the number of elements in the array: ";
cin >> n;
int arr[n];
cout << "Enter the elements of the array: ";
for (int i = 0; i < n; i++) {
cin >> arr[i];
}
cout << "The elements of the array in reverse order are: ";
printReverse(arr, 0, n);
cout << endl;
return 0;
```

Given an integer array nums that may contain duplicates, return all possible subsets (the power set). The solution set must not contain duplicate subsets. Return the solution in any order.
Example 1:
Input: nums = [1,2,2]
Output: [[],[1],[1,2],[1,2,2],[2],[2,2]]
Ans:

```
class Solution {
public:
vector<vector<int>> subsetsWithDup(vector<int>& nums) {
vector<vector<int>> ans; //to store all susbets
```

```
vector<int> curr; //to store current individual subset (that we will build)
sort(nums.begin(),nums.end()); //sort the array so that duplicates are adjacent
helper(nums,ans,curr,0); //we start from index 0
return ans;
}
void helper(vector<int>& nums, vector<vector<int>>& ans, vector<int>& curr, int idx){
ans.push_back(curr); //we include current susbet into final ans
for(int i=idx;i<nums.size();i++){ //check for all possibilites
if(i>idx && nums[i]==nums[i-1]) continue; //if duplicate then we continue
curr.push_back(nums[i]); //we include nums[i] in current subset
helper(nums,ans,curr,i+1);
curr.pop_back(); //to get subset without nums[i]
}
}
};
```