RECURSION - 3| Week 10

1.FIND OUT MINIMUM ELEMENT OF ARRAY USING RECURSION.

```
#include <iostream>
using namespace std;

// Function to find the minimum element in an array
recursively
int find_min(int arr[], int size) {
    // Base case: if size is 1, return the single element as
the minimum
    if (size == 1) {
        return arr[0];
    } else {
        // Recursive case: find the minimum of the rest of
the array
        int min_of_rest = find_min(arr, size - 1);
        // Compare the minimum of the rest with the current
element
        return (arr[size - 1] < min_of_rest) ? arr[size -
1] : min_of_rest;
    }
}

int main() {
    int arr[] = {5, 3, 8, 1, 9, 2};
    int size = sizeof(arr) / sizeof(arr[0]);

    int min_element = find_min(arr, size);

    cout << "The minimum element in the array is: " <<
min_element << endl;

    return 0;
}
```

2.FIND SUM OF ALL ELEMENTS OF ARRAY USING RECURSION.

```
#include <iostream>
using namespace std;

// Function to find the sum of all elements in an array
recursively
int sum_of_elements(int arr[], int size) {
    // Base case: if size is 0, return 0 (sum of an empty
array)
    if (size == 0) {
        return 0;
    } else {
        // Recursive case: add the last element to the sum of
the rest of the array
        return arr[size - 1] + sum_of_elements(arr, size -
1);
    }
}

int main() {
    int arr[] = {5, 3, 8, 1, 9, 2};
    int size = sizeof(arr) / sizeof(arr[0]);

    int sum = sum_of_elements(arr, size);

    cout << "The sum of all elements in the array is: " <<
sum << endl;

    return 0;
}
3.PRINT INDEX OF A GIVEN ELEMENT IN ARRAY. IF NOT PRESENT
PRINT-1.
#include <iostream>
using namespace std;

// Function to find index of a given element in an array
recursively
int find_index(int arr[], int size, int element, int
current_index) {
    // Base case: if current_index is equal to size, element
not found
    if (current_index == size) {
        return -1;
    }

    // Base case: if element found at current_index
    if (arr[current_index] == element) {
        return current_index;
    }

    // Recursive case: increment current_index and search in
the rest of the array
    return find_index(arr, size, element, current_index + 1);
}

int main() {
    int arr[] = {5, 3, 8, 1, 9, 2};
    int size = sizeof(arr) / sizeof(arr[0]);
    int element_to_find = 8;

    int index = find_index(arr, size, element_to_find, 0);

    if (index != -1) {
        cout << "Index of " << element_to_find << " in the
array is: " << index << endl;
    } else {
        cout << element_to_find << " is not present in the
array." << endl;
    }

    return 0;
}
```

4.PRINT ALL ELEMENTS OF ARRAY USING REVERSE ORDER.

```
#include <iostream>
using namespace std;

// Function to print all elements of array in reverse order
recursively
void print_reverse(int arr[], int size) {
    // Base case: if size is 0, return (no elements to print)
    if (size == 0) {
        return;
    } else {
        // Print the last element of the array
        cout << arr[size - 1] << " ";
        // Recursive case: print the rest of the array in
reverse order
        print_reverse(arr, size - 1);
    }
}

int main() {
    int arr[] = {5, 3, 8, 1, 9, 2};
    int size = sizeof(arr) / sizeof(arr[0]);

    cout << "Array elements in reverse order: ";
    print_reverse(arr, size);
    cout << endl;

    return 0;
}
```

5.PRINT AND STORE SUBSET OF ARRAY CONTAINING DUPLICATE CHARACTERS.

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;

// Function to print and store subsets containing duplicate
characters
void find_subsets_with_duplicates(vector<char>& subset,
vector<char>& arr, int index, unordered_set<string>& subsets)
{
    // Convert subset vector to a string (to handle
duplicates using unordered_set)
    string subset_str(subset.begin(), subset.end());
    subsets.insert(subset_str);  // Insert subset into
unordered_set (to avoid duplicates)

    for (int i = index; i < arr.size(); ++i) {
        // Include current element in the subset
        subset.push_back(arr[i]);
        // Recur for next elements, starting from i + 1
        find_subsets_with_duplicates(subset, arr, i + 1,
subsets);
        // Backtrack: remove current element from subset
        subset.pop_back();
    }
}

int main() {
    vector<char> arr = {'a', 'b', 'c', 'a'};
    vector<char> subset;
    unordered_set<string> subsets;

    find_subsets_with_duplicates(subset, arr, 0, subsets);

    // Print subsets containing duplicate characters
    cout << "Subsets containing duplicate characters:" <<
endl;
    for (auto subset_str : subsets) {
        cout << subset_str << endl;
    }

    return 0;
}
```