Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim: Implementing the Sorting Algorithms and understanding the time and space complexities**	
Experimrnt 1	**Date: 03-08-2025**	**Enrollment No:**

❖ Insertion Sort

Code:

```
#include <iostream>
#include <vector>
using namespace std;
void Print_Array(vector<int> Array)
{
   for (int i = 0; i < Array.size(); i++)
   {
      cout << Array[i] << " ";
   }
   cout << endl;
}
void Insertion_Sort(vector<int> &Array)
{
   for (int i = 1; i < Array.size(); i++)
   {
      int key = Array[i];
      int j = i - 1;
      while (j >= 0 && Array[j] > key)
      {
         Array[j + 1] = Array[j];
         j--;
      }
      Array[j + 1] = key;
   }
}
int main()
{
   vector<int> Array = {12, 45, 57, 78, 89, 62, 7, 49, 21, 23};
   cout << "Array Before Sorting :- " << endl;
   Print_Array(Array);
   Insertion_Sort(Array);
   cout << "Array After Sorting :- " << endl;
   Print_Array(Array);
   return 0;
}
```

Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim: Implementing the Sorting Algorithms and understanding the time and space complexities**	
Experimrnt 1	**Date: 03-08-2025**	**Enrollment No:**

Output:

Time Complexity: O(n)
Space Complexity: O(1)

❖ Bubble Sort

Code:

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

void Print_Array(vector<int> Array)
{
   for (int i = 0; i < Array.size(); i++)
   {
      cout << Array[i] << " ";
   }
   cout << endl;
}

void Swap(int &x, int &y)
{
   int temp = x;
   x = y;
   y = temp;
}

void Bubble_Sort(vector<int> &Array)
{
   int size = Array.size();
```

```
    for (int i = 0; i < size - 1; i++)
    {
        bool Swapped = false;

        for (int j = 0; j < size - i - 1; j++)
        {
            if (Array[j] > Array[j + 1])

            {
                Swap(Array[j], Array[j + 1]);
                Swapped = true;
            }
        }
        if (Swapped == false)
        {
            break;
        }
    }
    return;
}

int main()
{
    vector<int> Array = {11,4,54,24,35,22,55,65,33,15,76,63};
    cout << "Array Before Sorting :- " << endl;

    Print_Array(Array);
    Bubble_Sort(Array);

    cout << "Array After Sorting :- " << endl;
    Print_Array(Array);

    return 0;
}
```

Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim: Implementing the Sorting Algorithms and understanding the time and space complexities**	
Experimrnt 1	**Date: 03-08-2025**	**Enrollment No:**

Output:

Time Complexity: $O(n^2)$
Space Complexity: $O(1)$

❖ Selection Sort

Code:

```
#include <iostream>
#include <vector>
using namespace std;
void Print_Array(vector<int> Array){
    for (int i = 0; i < Array.size(); i++){
        cout << Array[i] << " ";
    }
    cout << endl;
}
void Swap(int &x, int &y){
    int temp = x;
    x = y;
    y = temp;
}
void Selection_Sort(vector<int> &Array){
    for (int i = 0; i < Array.size(); i++){
        int min_index = i;
        for (int j = i + 1; j < Array.size(); j++){
            if (Array[j] < Array[min_index]){
                min_index = j;
            }
        }
        Swap(Array[i], Array[min_index]);
    }
    return;
}
```

```
int main()
{
   vector<int> Array = {22,4,5,3,32,4,34,54,23,87,22,54};
   cout << "Array Before Sorting :- " << endl;
   Print_Array(Array);
   Selection_Sort(Array);
   cout << "Array After Sorting :- " << endl;
   Print_Array(Array);
   return 0;
}
```

Output:

```
PROBLEMS  SPELL CHECKER  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS                                   Code

cd "/Users/nikhilbhanderi/Documents/Semester 5/DAA/Lab - Manual/Experiment - 1/" && g++ Selection_Sort.cpp -o Selection_Sort && "/Users/nikhilbhanderi/Documents/Semes
 5/DAA/Lab - Manual/Experiment - 1/"Selection_Sort
nikhilbhanderi@Nikhils-MacBook-Air Experiment - 1 % cd "/Users/nikhilbhanderi/Documents/Semester 5/DAA/Lab - Manual/Experiment - 1/" && g++ Selection_Sort.cpp -o Sele
on_Sort && "/Users/nikhilbhanderi/Documents/Semester 5/DAA/Lab - Manual/Experiment - 1/"Selection_Sort
Array Before Sorting :-
22 4 5 3 32 4 34 54 23 87 22 54
Array After Sorting :-
3 4 4 5 22 22 23 32 34 54 54 87
nikhilbhanderi@Nikhils-MacBook-Air Experiment - 1 %
```

Time Complexity: $O(n^2)$
Space Complexity: $O(1)$

❖ Counting Sort

Code:

```
#include <iostream>
#include <vector>
using namespace std;
void Print_Array(vector<int> Array)
{
   for (int i = 0; i < Array.size(); i++)
   {
      cout << Array[i] << " ";
   }
   cout << endl;
}
int Find_Max(vector<int> Array)
{
```

```
    int max_num = Array[0];
    for (int i = 1; i < Array.size(); i++)
    {
        if (Array[i] > max_num)
        {
            max_num = Array[i];
        }
    }
    return max_num;
}
void Counting_Sort(vector<int> &Array)
{
    int max_num = Find_Max(Array);
    vector<int> count(max_num + 1, 0);

    for (int i = 0; i < Array.size(); i++)
    {
        count[Array[i]]++;
    }
    int index = 0;
    for (int i = 0; i <= max_num; i++){
        while (count[i] > 0){
            Array[index++] = i;
            count[i]--;
        }
    }
    return;
}
int main()
{
    vector<int> Array = {11,3,35,55,76,34,24,2,42,67,45};
    cout << "Array Before Sorting :- " << endl;
    Print_Array(Array);
    Counting_Sort(Array);
    cout << "Array After Sorting :- " << endl;
    Print_Array(Array);
    return 0;
}
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim: Implementing the Sorting Algorithms and understanding the time and space complexities**	
Experimrnt 1	**Date: 03-08-2025**	**Enrollment No:**

Output:

Time Complexity: O(n + k)
Space Complexity: O(k + n)

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim: Implementing the Searching Algorithms and understanding the time and space complexities**	
Experimrnt 2	**Date: 03-08-2025**	**Enrollment No:**

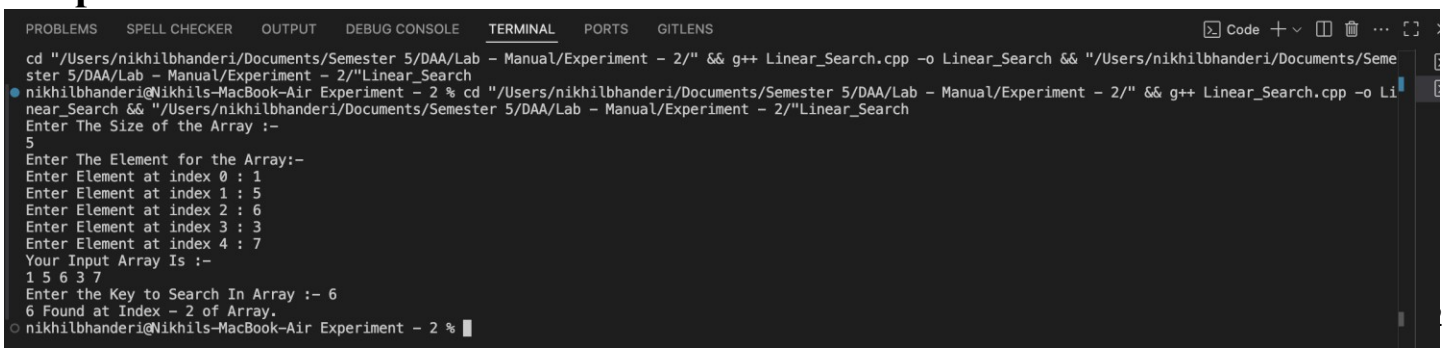
❖ Linear Serch

Code:

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

int Linear_Search(vector<int> Array, int key)
{
   for (int i = 0; i < Array.size(); i++)
   {
      if (Array[i] == key)
      {
         return i;
      }
   }
   return -1;
}

void Print_Array(vector<int> Array)
{
   for (int i = 0; i < Array.size(); i++)
   {
      cout << Array[i] << " ";
   }
   cout << endl;
}
void Input_Array(vector<int> &Array)
{
   for (int i = 0; i < Array.size(); i++)
   {
      cout << "Enter Element at index " << i << " : ";
      cin >> Array[i];
   }
}
int main()
{
   int size;
```

```
    int key;
    while (true)
    {
        cout << "Enter The Size of the Array :- " << endl;
        cin >> size;
        if (size >= 1)
        {
            break;
        }
        cout << "Invalid Size. Size must be a Positive Integer." << endl;
    }
    vector<int> Array(size, 0);
    cout << "Enter The Element for the Array:- " << endl;
    Input_Array(Array);
    cout << "Your Input Array Is :- " << endl;
    Print_Array(Array);
    cout << "Enter the Key to Search In Array :- ";
    cin >> key;
    int ans = Linear_Search(Array, key);
    if (ans != -1)
    {
        cout << key << " Found at Index - " << ans << " of Array." << endl;
    }
    else
    {
        cout << "Key is not exists in Array.";
    }
    return 0;
}
```

Output:



Time Complexity: O(n)
Space Complexity: O(1)

❖ Binary Search

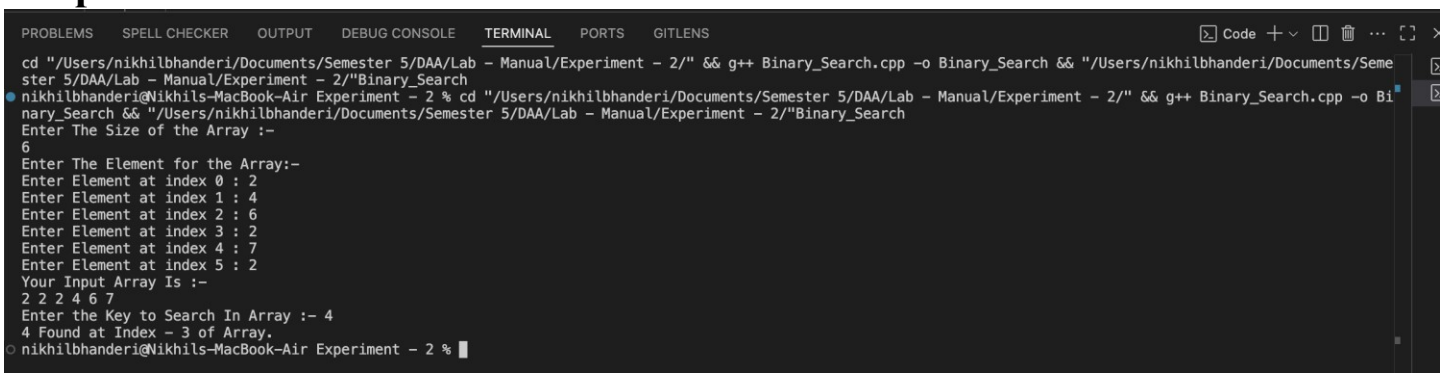
Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int Binary_Search(vector<int> Array, int left, int right, int key)
{
   while (left <= right){
      int mid = left + (right - left) / 2;
      if (key == Array[mid]){
         return mid;
      }
      if (key < Array[mid]){
         right = mid - 1;
      }
      else
      {
         left = mid + 1;
      }
   }
   return -1;
}
void Print_Array(vector<int> Array)
{
   for (int i = 0; i < Array.size(); i++)
   {
      cout << Array[i] << " ";
   }
   cout << endl;
}

void Input_Array(vector<int> &Array)
{
   for (int i = 0; i < Array.size(); i++)
```

```
    {
        cout << "Enter Element at index " << i << " : ";
        cin >> Array[i];
    }
}
int main()
{
    int size;
    int key;

    while (true)
    {
        cout << "Enter The Size of the Array :- " << endl;
        cin >> size;
        if (size >= 1)
        {
            break;
        }
        cout << "Invalid Size. Size must be a Positive Integer." << endl;
    }
    vector<int> Array(size, 0);
    cout << "Enter The Element for the Array:- " << endl;
    Input_Array(Array);
    sort(Array.begin(), Array.end());
    cout << "Your Input Array Is :- " << endl;
    Print_Array(Array);
    cout << "Enter the Key to Search In Array :- ";
    cin >> key;
    int ans = Binary_Search(Array, 0, size, key);
    if (ans != -1)
    {
        cout << key << " Found at Index - " << ans << " of Array." << endl;
    }
    else
    {
        cout << "Key is not exists in Array.";
    }
    return 0;
}
```

Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim: Implementing the Searching Algorithms and understanding the time and space complexities**	
Experimrnt 2	**Date: 03-08-2025**	**Enrollment No:**

Output:



Time Complexity:O(log n)
Space Complexity:O(1)

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Exponential Function with O(N) and O(logN).	
Experiment No: 03	**Date:**	**Enrollment No:**

I. Exponential Function using Iterative (Naive) Approach :-

Theory: -

This method calculates the exponential value x^n by simply multiplying x by itself n times. It starts with a result of 1 and keeps multiplying it by x in each step until it has done this n times.While this approach is simple and easy to understand, it becomes inefficient for large values of n, since it performs one multiplication per step — resulting in a time complexity of O(n).

Programming Language: - C++

Code :-

```
#include <bits/stdc++.h>
using namespace std;

long long Expotential(long base, long power) {
    if (power == 0) {
        return 1;
    }
    if (power == 1) {
        return base;
    }
    return base * Expotential(base, power - 1);
}

int main() {

    long base;
    long power;
    cout << "Enter the Base of the Ecpotentail :- ";
    cin >> base;
    cout << "Enter the Power of the Expotential :- ";
    cin >> power;

    long long result = Expotential(base, power);

    cout << "The " << base << " raised to " << power << " is " << result << " ." << endl;
    return 0;
}
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Exponential Function with O(N) and O(logN).	
Experiment No: 03	**Date:**	**Enrollment No:**

Output :-

```
Output                                                    Clear
Enter the Base of the Ecpotentail :- 3
Enter the Power of the Expotential :- 2
The 3 raised to 2 is 9 .


=== Code Execution Successful ===
```

Space Complexity:- O(N)

Justification: -

The function uses recursion, and for each recursive call, a new stack frame is added to the call stack.So, if the power is n, the recursion will go n levels deep (from power down to 0).There is no additional data structure used, only the function call stack consumes space.

Time Complexity:O(1)

Best Case Time Complexity: O(1)

Justification: -

If power == 0, the function immediately returns 1.This is a constant-time operation and doesn't make any recursive call.

II. Exponential Function with O(N) using Divide and Conquer Approach :-

Theory: -This recursive method is smarter than the basic one because it uses a divide-and-conquer strategy to reduce the number of multiplications.

- If the exponent n is even, it calculates $x^{n/2}$ just once, and then squares the result.
- If n is odd, it calculates $x^{(n-1)/2}$, squares it, and then multiplies once more by x.

By breaking the problem in half each time, this approach significantly speeds things up, especially for large values of n.

Programming Language: - C++

Code :-

```
#include <bits/stdc++.h>
using namespace std;

long long Expotential(long base, long power) {
    if (power == 0) {
        return 1;
```

```
    }

    else if (power % 2 == 0) {
        return Expotential(base, power / 2) * Expotential(base, power / 2);
    }
    else {
        return base * Expotential(base, power / 2) * Expotential(base, power / 2);
    }
}
int main() {

    long base;
    long power;
    cout << "Enter the Base of the Ecpotentail :- ";
    cin >> base;

    cout << "Enter the Power of the Expotential :- ";
    cin >> power;

    long long result = Expotential(base, power);
    cout << "The " << base << " raised to " << power << " is " << result << " ." << endl;
    return 0;
}
```

Output:-

```
Output                                                    Clear
Enter the Base of the Ecpotentail :- 3
Enter the Power of the Expotential :- 4
The 3 raised to 4 is 81 .


=== Code Execution Successful ===
```

Space Complexity:- O(power)

Justification: - Although the algorithm uses divide and conquer, it calls the recursive function twice in each step instead of storing the result and reusing it.This leads to redundant recursive calls, which results in a tree of recursive calls — similar to the naive recursion.Hence, the recursion depth can reach up to power in the worst case.So, space complexity remains O(power) due to the recursive call stack.

Time Complexity:-O(1)

Justification: - When power == 0, the function returns 1 immediately with no recursion, which takes constant time.

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Exponential Function with O(N) and O(logN).	
Experiment No: 03	**Date:**	**Enrollment No:**

III. Exponential Function with O(logN) using Divide and Conquer Approach :-

Theory: -

This improved divide-and-conquer method uses the math behind exponentiation to make the process much faster:

- If the exponent nnn is even, it calculates $x^{n/2}$ once and squares it.
- If n is odd, it calculates $x^{(n-1)/2}$, squares that, and multiplies once more by x.

By cutting the exponent in half each time, this approach only needs about log(n) steps, making it very efficient — especially when dealing with large values of n. It significantly reduces the number of multiplications compared to the basic method.

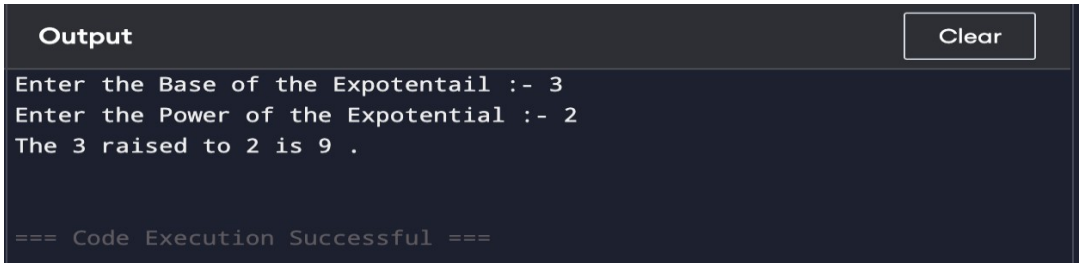
Programming Language: - C++

Code :-

```
#include <bits/stdc++.h>
using namespace std;

long long Expotential(long base, long power) {
    if (power == 0) {
        return 1;
    }
    if (base == 0) {
        return 0;
    }
    if (power < 0) {
        return  1 / Expotential(base, power * -1);
    }
    if (power == 1) {
        return base;
    }
    long long Half_Power = Expotential(base, power / 2);
    if (power % 2 == 0) {
        return Half_Power * Half_Power ;
    }
    else {
        return base * Half_Power * Half_Power ;
    }
}
int main() {
```

```
    long base;
    long power;
    cout << "Enter the Base of the Expotentail :- ";

    cin >> base;
    cout << "Enter the Power of the Expotential :- ";
    cin >> power;
    long long result = Expotential(base, power);
    cout << "The " << base << " raised to " << power << " is " << result << " ." << endl;
    return 0;
}
```

Output:-



Space Complexity:-O(log power)
Justification: - The function uses recursion, and in each call, it divides the power by 2.This means the depth of the recursive call stack is $\log_2$(power) in the worst case.No extra memory (like arrays or maps) is used—just the call stack.

Time Complexity:O(1)
Justification: - When power == 0, the function immediately returns 1 without any recursion or multiplication. This takes constant time: O(1).

Conclusion:- In this experiment, we calculated x^n using three C++ methods—naive recursion, basic divide-and-conquer, and an optimized recursive approach—to understand their effect on time and space complexity.

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Sorting Algorithms Using Divide and Conquer Approach	
Experiment No: 04	**Date:13/09/2025**	**Enrollment No:**

Merge Sort

Code :-

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

void Print_Array(vector<int> Array) {
    for (int i = 0; i < Array.size(); i++) {
        cout << Array[i] << " ";
    }
    cout << endl;
}



void Merge(vector<int>& Array, int low, int mid, int high) {
    int lower_bound = mid - low + 1;
    int upper_bound = high - mid;
    vector<int> Left_Array(lower_bound);
    vector<int> Right_Array(upper_bound);
        for (int i = 0; i < lower_bound; i++) {
        Left_Array[i] = Array[low + i];
    }
    for (int i = 0; i < upper_bound; i++) {
        Right_Array[i] = Array[mid + 1 + i];
    }
    int i = 0;
    int j = 0;
    int k = low;
    while (i < lower_bound && j < upper_bound) {
        if (Left_Array[i] <= Right_Array[j]) {
            Array[k] = Left_Array[i];
            i++;
        }
        else {
            Array[k] = Right_Array[j];
            j++;
        }
        k++;
    }
    while (i < lower_bound) {
```

Marwari University Faculty of Technology Department of Information and Communication Technology		
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Sorting Algorithms Using Divide and Conquer Approach	
Experiment No: 04	**Date:13/09/2025**	**Enrollment No:**

```
            Array[k] = Left_Array[i];
            i++;
            k++;
        }
        while (j < upper_bound) {
            Array[k] = Right_Array[j];
            j++;
            k++;
        }
 }


void Merge_Sort(vector<int>& Array, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        Merge_Sort(Array, left, mid);
        Merge_Sort(Array, mid + 1, right);
        Merge(Array, left, mid, right);
    }
}

int main() {
    vector<int> Array = { 12, 45, 57, 78, 89, 62, 7, 49, 21, 23 };
    int size = Array.size();
    cout << "Array Before Sorting :- " << endl;
    Print_Array(Array);
    Merge_Sort(Array, 0, size - 1);
    cout << "Array After Sorting :- " << endl;
    Print_Array(Array);
    return 0;
}
```

Output :-

```
Array Before Sorting :-
12 45 57 78 89 62 7 49 21 23
Array After Sorting :-
7 12 21 23 45 49 57 62 78 89
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Sorting Algorithms Using Divide and Conquer Approach	
Experiment No: 04	**Date:13/09/2025**	**Enrollment No:**

Quick Sort

Code :-

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

void Swap(int& x, int& y) {
    int temp = x;
    x = y;
    y = temp;
}

void Print_Array(vector<int> Array) {
    for (int i = 0; i < Array.size(); i++) {
        cout << Array[i] << " ";
    }
    cout << endl;
}

int Partion(vector<int>& Array, int left, int right) {
    int pivot = Array[left];
    while (true) {
        while (left < right && Array[left] < pivot) {
            left++;
        }
        while (left < right && Array[right] > pivot) {
            right--;
        }
        if (left >= right) {
            break;
        }

        Swap(Array[left], Array[right]);
    }


        Swap(Array[right], pivot);
    return right;
}

void Quick_Sort(vector<int>& Array, int left, int right) {
    if (left < right) {
        int Pivot_Index = Partion(Array, left, right);
        Quick_Sort(Array, left, Pivot_Index - 1);
```

```
        Quick_Sort(Array, Pivot_Index + 1, right);
    }
}

int main() {
    vector<int> Array = { 12, 45, 57, 78, 89, 62, 7, 49, 21, 23 };
    int size = Array.size();
    cout << "Array Before Sorting :- " << endl;
    Print_Array(Array);
    Quick_Sort(Array, 0, size - 1);
    cout << "Array After Sorting :- " << endl;
    Print_Array(Array);
    return 0;
}
```

Output :-

```
Array Before Sorting :-
12 45 57 78 89 62 7 49 21 23
Array After Sorting :-
7 12 21 23 45 49 57 62 78 89
```

Conclusion: We learnt in this experiment that both Merge Sort and Quick Sort use the Divide and Conquer technique to sort data efficiently. Merge Sort gives consistent performance with extra memory, while Quick Sort is faster on average but depends on pivot selection.

Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Application-based Algorithm using D&C Approach	
Experiment No: 05	**Date: 13\09\2025**	**Enrollment No:**

Karatsuba Algorithm

Code :-

```
#include <bits/stdc++.h>
 using namespace std;

 int Get_Size(long long num) {
    return num == 0 ? 1 : static_cast<int>(log10(num)) + 1;
 }

 long long int Karatsuba(long long num1, long long num2) {

    if (num1 < 10 || num2 < 10) {
       return num1 * num2;
    }

    int length = max(Get_Size(num1), Get_Size(num2));
    int half = length / 2 + length % 2;
    long long powerOf10 = static_cast<long long>(pow(10, half));
    long long powerOf102x = powerOf10 * powerOf10;
    long long a = num1 / powerOf10;
    long long b = num1 % powerOf10;
    long long c = num2 / powerOf10;
    long long d = num2 % powerOf10;
    long long ac = Karatsuba(a, c);
    long long bd = Karatsuba(b, d);
    long long ab_cd = Karatsuba(a + b, c + d);
    long long int ans = ac * powerOf102x + (ab_cd - ac - bd) * powerOf10 + bd;

    return ans;
 }

 int main() {

    long long x;
    cout << "Enter the First Number :- ";
    cin >> x;

    long long y;
    cout << "Enter the Second Number :- ";
    cin >> y;

    long long int ans = Karatsuba(x, y);
    cout << "The Product of " << x << " and " << y << " is: " << ans;
```

```
    return 0;

}
```

Output :-

```
Enter the First Number :- 4
Enter the Second Number :- 5
The Product of 4 and 5 is: 20
```

Conclusion: We learnt in this experiment that the Karatsuba Algorithm is an efficient multiplication method that reduces the complexity from $O(n^2)$ in normal multiplication to about using the Divide and Conquer approach. It splits large numbers into parts and combines the results, making multiplication faster for very large inputs.

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Knapsack Problem using Greedy Approach.	
Experiment No: 06	**Date: 13\09\2025**	**Enrollment No:**

Knapsack Problem

Code :-

```
#include<bits/stdc++.h>
using namespace std;

int Knapsack_0_1(vector<int>& Profit, vector<int>& Weigth, int total_weight) {
    vector<pair<double, int>> Profit_weight;
    for (int i = 0; i < Profit.size(); i++) {
        Profit_weight.push_back({ (double)Profit[i] / Weigth[i], Weigth[i] });
    }

    sort(Profit_weight.begin(), Profit_weight.end(), [](pair<double, int>& a, pair<double, int>& b) {
        return a.first > b.first;
        });

    int max_profit = 0;

    for (int i = 0; i < Profit_weight.size(); i++) {
        if (Profit_weight[i].second <= total_weight) {
            max_profit += Profit_weight[i].first * Profit_weight[i].second;
            total_weight -= Profit_weight[i].second;
        }
        else {
            break;
        }
    }
    return max_profit;
}



int main() {
    vector<int> Profit = { 60,100,120 };
    vector<int> Weight = { 10,20,20 };
    int total_weight = 40;
    int total_profit = Knapsack_0_1(Profit, Weight, total_weight);
    cout << "Total Profit Is : " << total_profit << endl;
    return 0;
}
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Knapsack Problem using Greedy Approach.	
Experiment No: 06	**Date: 13\09\2025**	**Enrollment No:**

Output :-

```
Total Profit  : 180


=== Code Execution Successful ===
```

Conclusion:-

We learnt in this experiment that the Knapsack Problem can be solved using a greedy approach by selecting items based on their profit-to-weight ratio. This method helps maximize profit within the given capacity and is useful in real-world optimization tasks like resource management.

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing application-based algorithms using Greedy Approach	
Experiment No: 07	**Date: 13\09\2025**	**Enrollment No:**

I. Job Scheduling Problem

Code :-

```
#include <bits/stdc++.h>
using namespace std;

class Job {
public:
    int id;
    int deadline;
    int profit;
    Job(int id, int deadline, int profit): id(id), deadline(deadline), profit(profit) {}
};

pair<int, long long> Job_Scheduling(vector<Job> jobs) {
    long long Max_Profit = 0.0;
    int count_of_jobs = 0;
    int max_deadline = -1;
    sort(jobs.begin(), jobs.end(), [](const Job& a, const Job& b) {
        return a.profit > b.profit;
        });
    for (const Job& job : jobs) {
        max_deadline = max(max_deadline, job.deadline);
    }
    vector<int> selectedJobs(max_deadline + 1, -1);
    for (int i = 0; i < jobs.size(); i++) {
        for (int j = jobs[i].deadline; j >= 0; j--) {
            if (selectedJobs[j] == -1) {
                selectedJobs[j] = jobs[i].id;
                count_of_jobs++;
                Max_Profit += jobs[i].profit;
                break;
            }
        }
    }

    return { count_of_jobs , Max_Profit };
}


int main() {
    vector<Job> jobs;
```

```
    vector<int> deadline = { 2, 4, 6, 5, 2, 2, 6, 4 };
    vector<int> profit = { 22, 25, 20, 10, 65, 60, 70, 80 };

    for (int i = 0; i < 8; i++) {
        jobs.push_back(Job(i + 1, deadline[i], profit[i]));
    }

    pair<int, long long> Count_and_profit = Job_Scheduling(jobs);

    cout << "We can Perform " << Count_and_profit.first << " and Earn Profit of Rs. " << Count_and_profit.second << " ." << endl;

    return 0;

}
```

Output :-

II. Activity Selection Problem :-

Code :-

```
#include <bits/stdc++.h>
using namespace std;
class Activity {
public:
    int id;
    int start;
    int end;
    Activity(int id , int start, int end) : id(id) , start(start), end(end) {};
};
vector<Activity> Activity_Selection(vector<Activity>& activities) {
    vector<Activity> selected_activities;
    sort(activities.begin(), activities.end(), [](Activity a, Activity b) {
        return a.end < b.end;
        });
    selected_activities.push_back(activities[0]);
    int last_activity_end = selected_activities.back().end;
    for (int i = 1; i < activities.size(); i++) {
        if (activities[i].start >= last_activity_end) {
            selected_activities.push_back(activities[i]);
            last_activity_end = selected_activities.back().end;
        }
    }
    return selected_activities;
}
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing application-based algorithms using Greedy Approach	
Experiment No: 07	**Date: 13\09\2025**	**Enrollment No:**

```
int main() {

    vector<Activity> activities;
    vector<int> start_time = { 5,1,3,0,5,8 };
    vector<int> end_time = { 9,2,4,6,7,9 };
    for (int i = 0; i < start_time.size(); i++) {
        activities.push_back(Activity(i+ 1 ,start_time[i], end_time[i]));
    }
    vector<Activity> selected_activities = Activity_Selection(activities);
    cout << "Selected Activities are :- " << endl;
    for (auto activity : selected_activities) {
        cout << "Activity ID :- " << activity.id <<  " Start Time: " << activity.start << ", End Time: " << activity.end << endl;
    }

    return 0;
}
```

Output :-

```
Selected Activities are :-
Activity  :- 5 Start Time: 7, End Time: 1
Activity  :- 2 Start Time: 4, End Time: 5
Activity  :- 3 Start Time: 5, End Time: 6
Activity  :- 6 Start Time: 9, End Time: 8
```

Conclusion:

We learnt in this experiment that greedy algorithms solve the Job Scheduling and Activity Selection problems efficiently. Job scheduling maximizes profit within deadlines, while activity selection maximizes non-overlapping activities.

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Longest Common Sub-sequence using Dynamic Programming Approach	
Experiment No: 08	**Date: 13\09\2025**	**Enrollment No:**

Longest Common Sub-sequence

Code :-

```
// LCS
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
int lcs(string &str1, string &str2)
{
   int m = str1.length();
   int n = str2.length();

   vector<vector<int>> l(m + 1, vector<int>(n + 1, 0));

   for (int i = 1; i < m + 1; i++)
   {
      for (int j = 1; j < n + 1; j++)
      {
         if (str1[i - 1] == str2[j - 1])
         {
            l[i][j] = l[i - 1][j - 1] + 1;
         }
         else
         {
            l[i][j] = max(l[i][j - 1], l[i - 1][j]);
         }
      }
   }
   return l[m][n];
}

int main()
{
   string str1 = "longest";
   string str2 = "stone";
   cout << lcs(str1, str2);
   return 0;
      }}
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Longest Common Sub-sequence using Dynamic Programming Approach	
Experiment No: 08	**Date: 13\09\2025**	**Enrollment No:**

Output :-

```
3
```

Conclusion:-

We learnt in this experiment that the Longest Common Subsequence (LCS) problem can be efficiently solved using dynamic programming. It helps in finding the longest sequence present in both strings, which is useful in fields like text comparison, bioinformatics, and data analysis.

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing 0/1 Knapsack Problem using Dynamic Programming Approach	
Experiment No: 09	**Date: 13\09\2025**	**Enrollment No:**

Knapsack Problem

Code :-

```
#include<bits/stdc++.h>
using namespace std;

int knapsack(int weights[], int profits[], int n, int capacity) {
    vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));


    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= capacity; w++) {
            if (weights[i - 1] <= w) {
                dp[i][w] = max(dp[i - 1][w], profits[i - 1] + dp[i - 1][w - weights[i - 1]]);
            }
            else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    cout << "DP Table (Max Value for Each Capacity):\n";
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= capacity; w++) {
            cout << setw(4) << dp[i][w] << "\t";
        }

        cout << endl;
    }

    return dp[n][capacity];
}

int main() {
    int weights[] = { 2, 3, 4, 5 };
    int profits[] = { 3015, 4026, 5789, 6147 };
    int capacity = 5;
    int n = sizeof(weights) / sizeof(weights[0]);

    int max_profit = knapsack(weights, profits, n, capacity);

    cout << "Maximum value in Knapsack = " << max_profit << endl;

    return 0;
  }
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing 0/1 Knapsack Problem using Dynamic Programming Approach	
Experiment No: 09	**Date: 13\09\2025**	**Enrollment No:**

Output :-

Conclusion:

We learnt in this experiment that the Knapsack Problem can be efficiently solved using dynamic programming. It helps in selecting items to maximize profit without exceeding the capacity, which is useful in resource allocation and optimization problems.

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Matrix Chain Multiplication using Dynamic Programming Approach	
Experiment No: 10	**Date: 13\09\2025**	**Enrollment No:**

Code :-

```
#include <bits/stdc++.h>
using namespace std;

int matrixMultiplication(vector<int>& arr) {
    int n = arr.size();
    vector<vector<int>> dp(n, vector<int>(n, 0));

    for (int len = 2; len < n; len++) {
        for (int i = 0; i < n - len; i++) {
            int j = i + len;
            dp[i][j] = INT_MAX;
            for (int k = i + 1; k < j; k++) {
                int cost = dp[i][k] + dp[k][j] + arr[i] * arr[k] * arr[j];
                dp[i][j] = min(dp[i][j], cost);
            }
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << setw(4) << dp[i][j] << " ";
        }
        cout << endl;
    }
    return dp[0][n - 1];
}

int main() {
    vector<int> arr;
    int n;
    cout << "Enter the Number of Matrix :- ";
    cin >> n;


for( int i = 0; i < n; i++) {
        int row, column;
        cout << "Enter the Number of Rows and Columns for Matrix " << i + 1 << " :- ";
        cin >> row >> column;
```

```
        if (i == 0) {
            arr.push_back(row);
            arr.push_back(column);
        }
        else {
            arr.push_back(column);
        }
    }
    cout << matrixMultiplication(arr);
    return 0;
}
```

Output :-

```
Enter the Number of Matrix :- 3
Enter the Number of Rows and Columns for Matrix 1 :- 2
2
Enter the Number of Rows and Columns for Matrix 2 :- 3
3
Enter the Number of Rows and Columns for Matrix 3 :- 3

6
   0    0   12   48
   0    0    0   36
   0    0    0    0
   0    0    0    0
48
```