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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

Aim: Implementing the Sorting Algorithms

IDE: Visual Studio Code

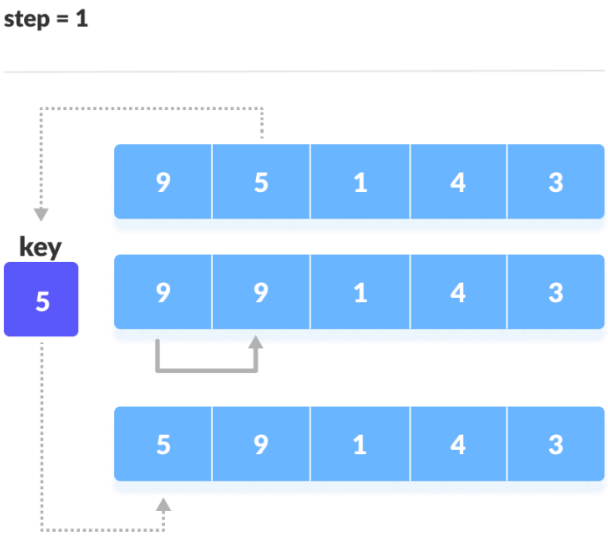
Insertion Sort

Theory: -

- Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.
- Insertion sort works similarly as we sort cards in our hands in a card game.
- We assume that the first card is already sorted then, we select an unsorted card. If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left. In the same way, other unsorted cards are taken and put in their right place.

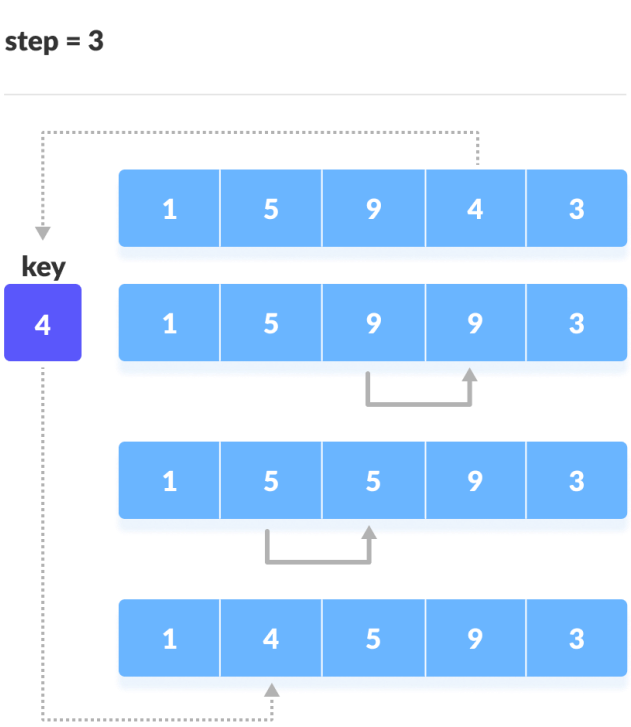
Working of Insertion Sort

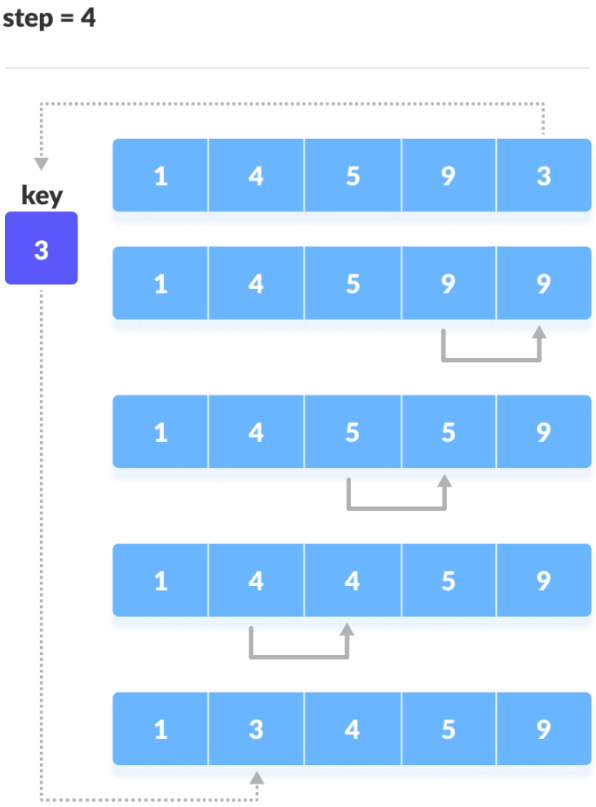
1) The first element in the array is assumed to be sorted. Take the second element and store it separately in the key. Compare the key with the first element. If the first element is greater than the key, then the key is placed in front of the first element.



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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

2) Now, the first two elements are sorted. Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If no element is smaller than it, place it at the beginning of the array.





3) Similarly, place every unsorted element in its correct position.

Algorithm: -

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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

Programming Language: - C++

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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1> cd "
d:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1\" ; if (
$?) { g++ Insertion_Sort.cpp -o Insertion_Sort } ; if ($?) { .\Insertion_Sort }
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
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1>
```

Space Complexity:- _____
Justification: -

_____

_____

Time Complexity:

Best Case Time Complexity: _____
Justification: -

_____

_____

Worst Case Time Complexity:- _____
Justification: -

_____

_____

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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

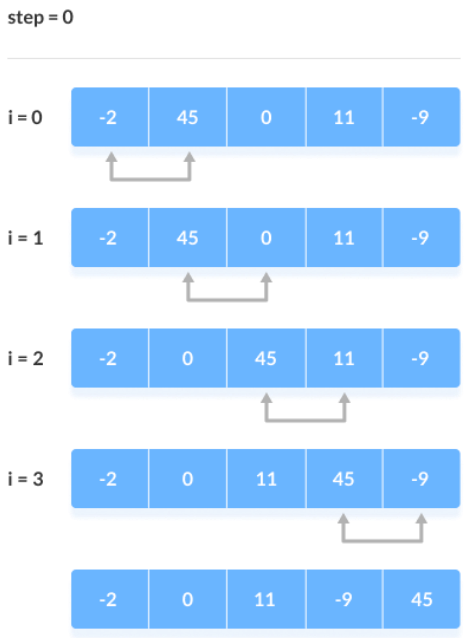
Bubble Sort

Theory: -

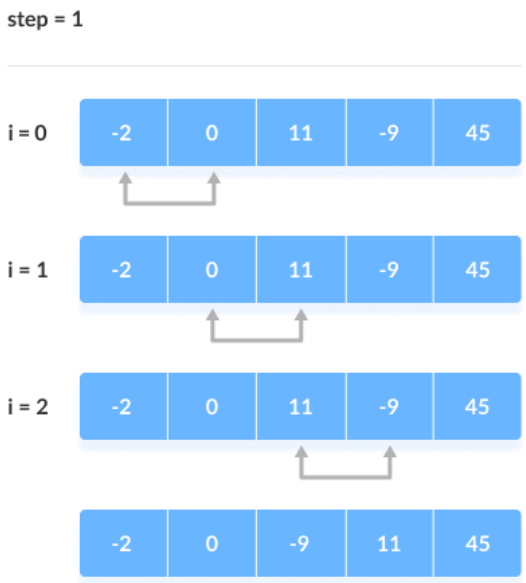
- Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.
- Just like the movement of air bubbles in the water that rise up to the surface, each array element moves to the end in each iteration. Therefore, it is called a bubble sort.

Working of Bubble Sort

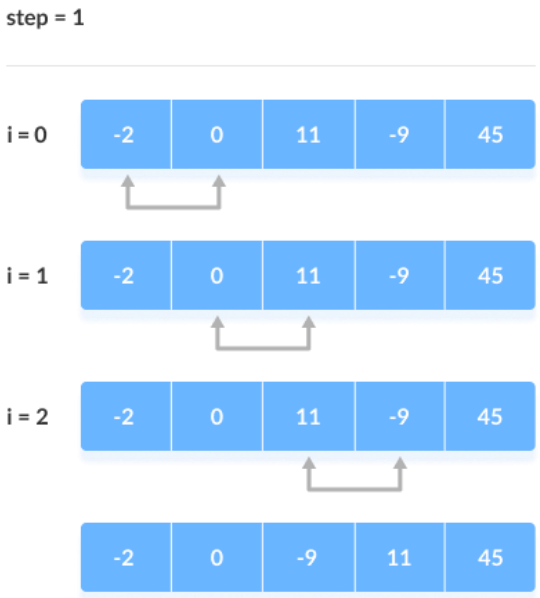
1) First Iteration (Compare and Swap)
    1) Starting from the first index, compare the first and the second elements.
    2) If the first element is greater than the second element, they are swapped.
    3) Now, compare the second and the third elements. Swap them if they are not in order.
    4) The above process goes on until the last element



2) Remaining Iteration
    - The same process goes on for the remaining iterations.
    - After each iteration, the largest element among the unsorted elements is placed at the end.

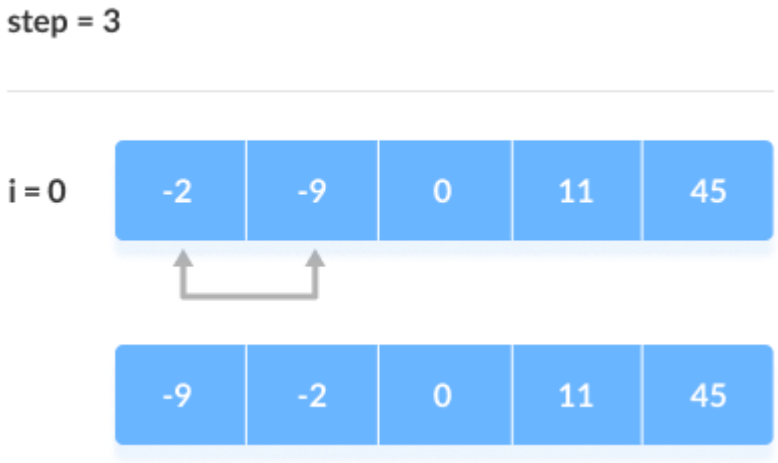


- The array is sorted when all the unsorted elements are placed at their correct positions.



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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

- The array is sorted when all the unsorted elements are placed at their correct positions.



Algorithm: -

Programming Language: - C++

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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

Code :-

```
#include<bits/stdc++.h>
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 = { 12 ,45 , 57 , 78 , 89 , 62 , 7 , 49 , 21 , 23 };
    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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1> cd "
d:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1\" ; if (
$?) { g++ Bubble_Sort.cpp -o Bubble_Sort } ; if ($?) { .\Bubble_Sort }
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
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1>
```

Space Complexity:- _____
Justification: -

_____

_____

Time Complexity:

Best Case Time Complexity: _____
Justification: -

_____

_____

Worst Case Time Complexity:- _____
Justification: -

_____

_____

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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

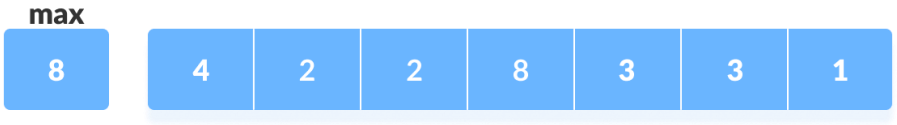
Counting Sort

Theory: -

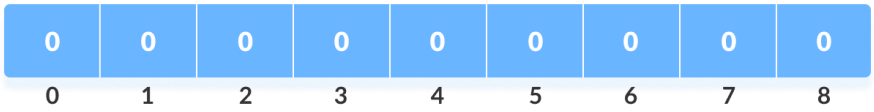
- Counting sort is a sorting algorithm that sorts the elements of an array by counting the number of occurrences of each unique element in the array.
- The count is stored in an auxiliary array and the sorting is done by mapping the count as an index of the auxiliary array.

Working of Counting Sort

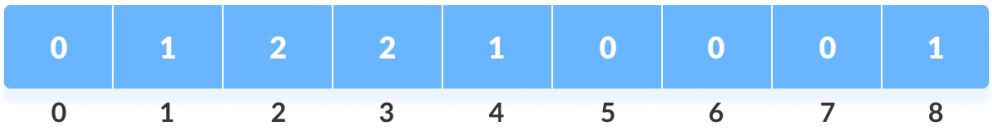
1) Find out the maximum element (let it be max) from the given array.



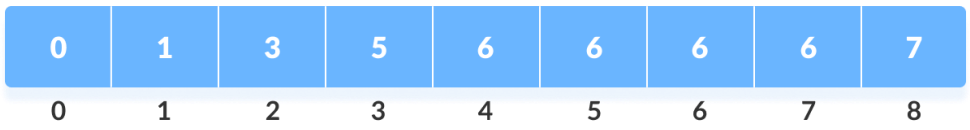
2) Initialize an array of length max+1 with all elements 0. This array is used for storing the count of the elements in the array.



3) Store the count of each element at their respective index in count array

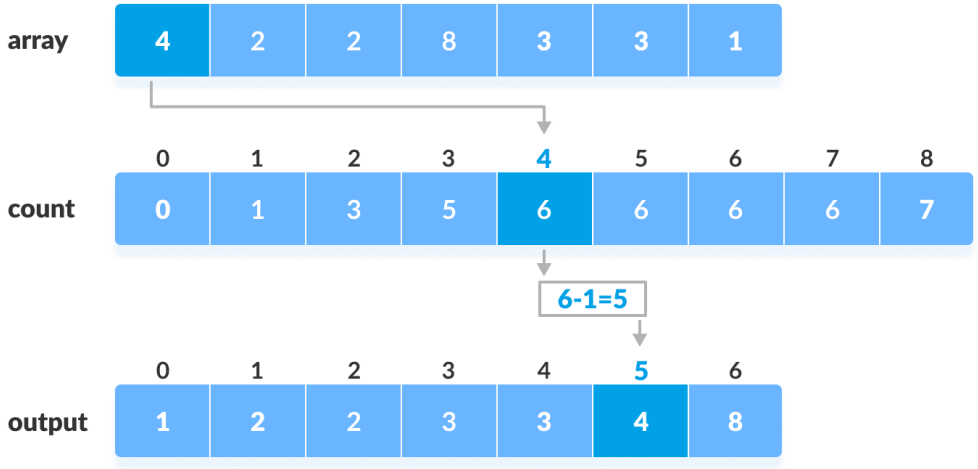


4) Store cumulative sum of the elements of the count array. It helps in placing the elements into the correct index of the sorted array.



5) Find the index of each element of the original array in the count array. This gives the cumulative count. Place the element at the index calculated.

Marwari University Faculty of Technology Department of Information and Communication Technology		
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Sorting Algorithms	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**



6) After placing each element at its correct position, decrease its count by one.

Algorithm: -

Programming Language: - C++

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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

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;
}
```

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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

```
int main() {

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

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1> cd "
d:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1\" ; if (
$?) { g++ Counting_Sort.cpp -o Counting_Sort } ; if ($?) { .\Counting_Sort }
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
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1>
```

Space Complexity:- _____
Justification: -

_____

_____

Time Complexity:

Best Case Time Complexity: _____
Justification: -

_____

_____

Worst Case Time Complexity:- _____
Justification: -

_____

_____

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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

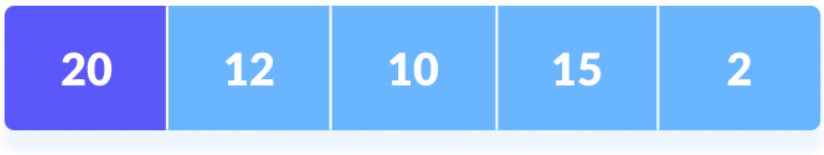
Selection Sort

Theory: -

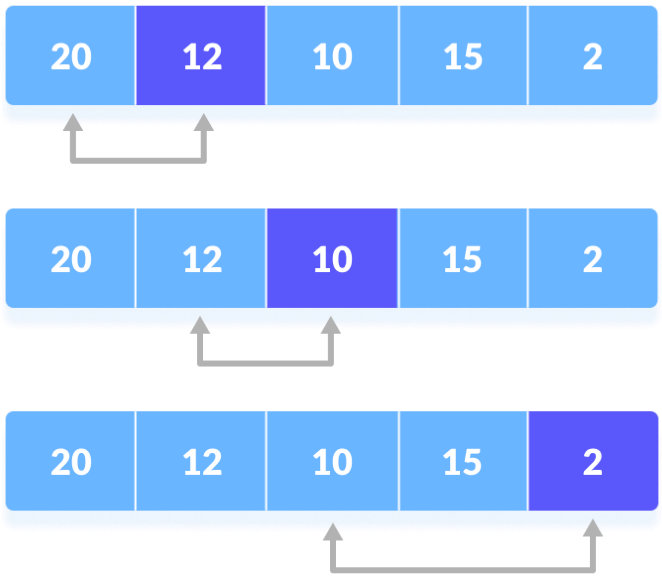
- Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.
- The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted part. This process is repeated for the remaining unsorted portion until the entire list is sorted.
-

Working of Selection Sort

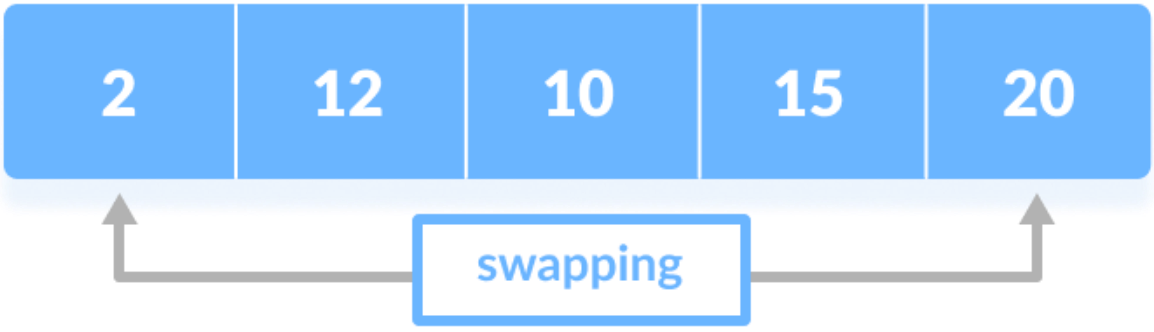
1) Set the first element as a minimum.



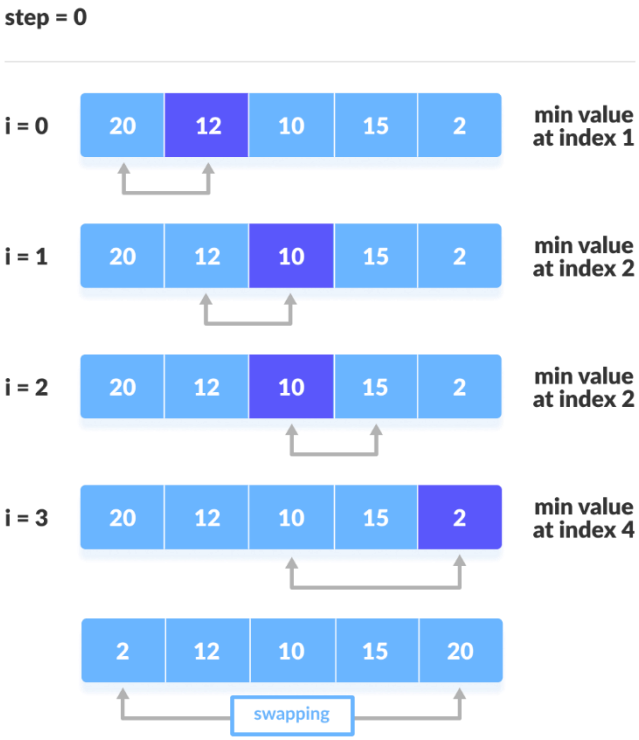
2) Compare the minimum with the second element. If the second element is smaller than the minimum, assign the second element as the minimum. Compare minimum with the third element. Again, if the third element is smaller, then assign a minimum to the third element otherwise do nothing. The process goes on until the last element.

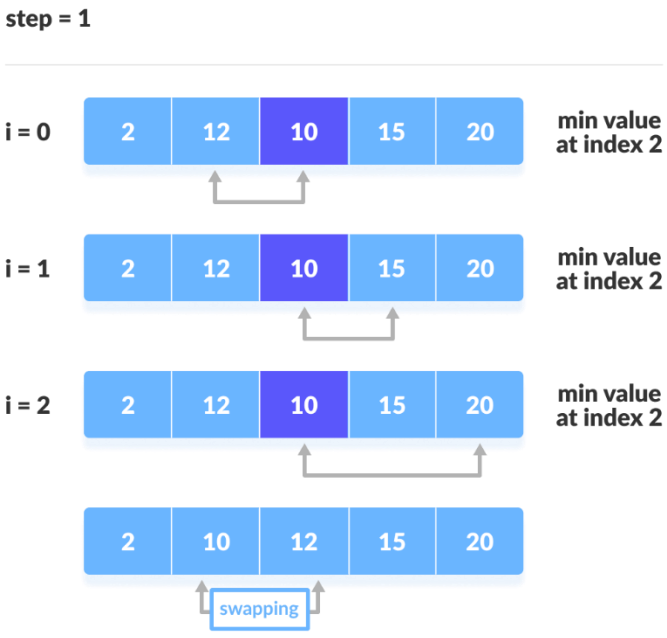


3) After each iteration, the minimum is placed in the front of the unsorted list.

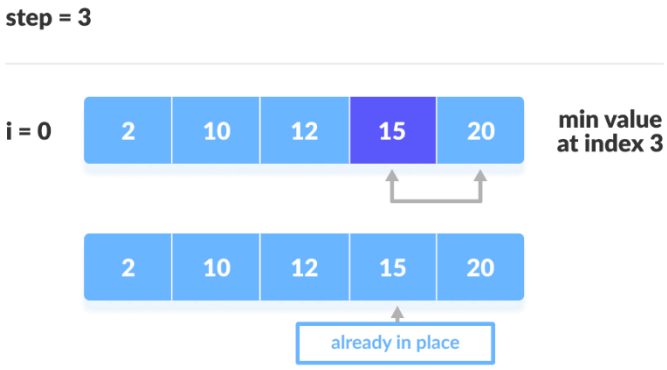


4) For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.





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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**



Algorithm: -

Programming Language: - C++

Code :-

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

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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

```
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 = { 12 ,45 , 57 , 78 , 89 , 62 , 7 , 49 , 21 , 23 };

      cout << "Array Before Sorting :- " << endl;
      Print_Array(Array);
      Selection_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	
Experiment No: 01	**Date:**	**Enrollment No: 92200133030**

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1> cd "
d:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1\" ; if (
$?) { g++ Selection_Sort.cpp -o Selection_Sort } ; if ($?) { .\Selection_Sort }
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
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 1>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-

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	
Experiment No: 02	**Date:**	**Enrollment No: 92200133030**

Aim: Implementing the Searching Algorithms

IDE: Visual Studio Code

Linear Search :-

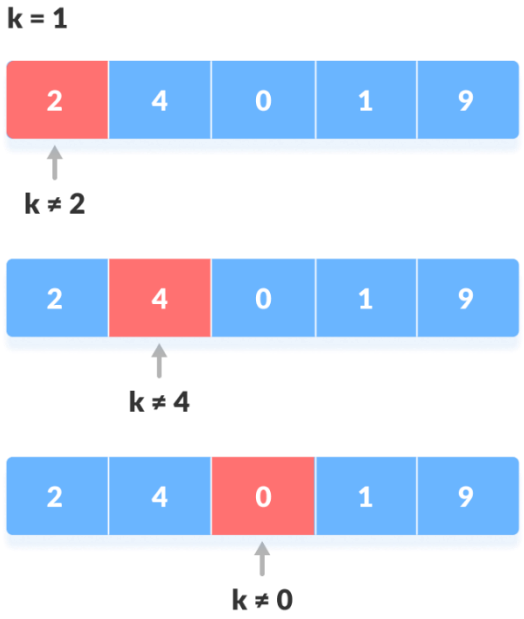
Theory: -

- Linear search is a sequential searching algorithm where we start from one end and check every element of the list until the desired element is found. It is the simplest searching algorithm.

Working of Linear Search :-

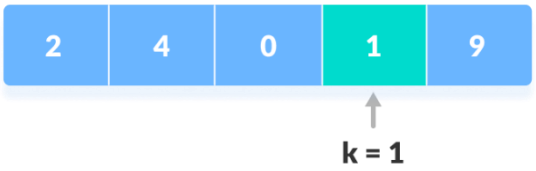
- The following steps are followed to search for an element k = 1 in the list below.

1) Start from the first element, compare k with each element x.



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	
Experiment No: 02	**Date:**	**Enrollment No: 92200133030**

2) If x == k, return the index.



3) Else, return not found.

Algorithm: -

Programming Language: - C++

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;
}
```

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	
Experiment No: 02	**Date:**	**Enrollment No: 92200133030**

```
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;
    }
```

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	
Experiment No: 02	**Date:**	**Enrollment No: 92200133030**

```
        else {
            cout << "Key is not exists in Array.";
        }
        return 0;
    }
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 2> cd "d:\Arya
n Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 2\" ; if ($?) { g++ Line
ar_Search.cpp -o Linear_Search } ; if ($?) { .\Linear_Search }
Enter The Size of the Array :-
10
Enter The Element for the Array:-
Enter Element at index 0 : 23
Enter Element at index 1 : 45
Enter Element at index 2 : 67
Enter Element at index 3 : 89
Enter Element at index 4 : 90
Enter Element at index 5 : 23
Enter Element at index 6 : 17
Enter Element at index 7 : 65
Enter Element at index 8 : 39
Enter Element at index 9 : 71
Your Input Array Is :-
23 45 67 89 90 23 17 65 39 71
Enter the Key to Search In Array :- 23
23 Found at Index - 0 of Array.
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 2>
```

Space Complexity:- _____

Justification: -

Time Complexity:

Best Case Time Complexity: _____

Justification: -

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	
Experiment No: 02	**Date:**	**Enrollment No: 92200133030**

Worst Case Time Complexity:- _____
Justification: -

Binary Search

Theory: -

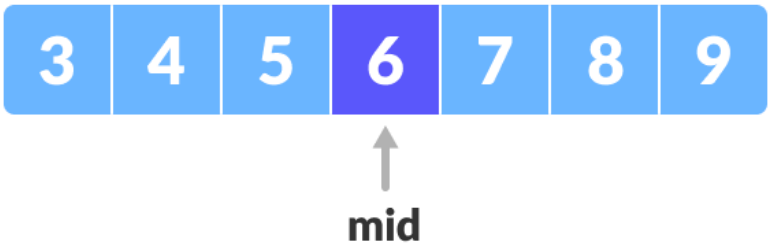
- Binary Search is a searching algorithm for finding an element's position in a sorted array.
- In this approach, the element is always searched in the middle of a portion of an array.
- Binary search can be implemented only on a sorted list of items. If the elements are not sorted already, we need to sort them first.

Working of Bubble Sort

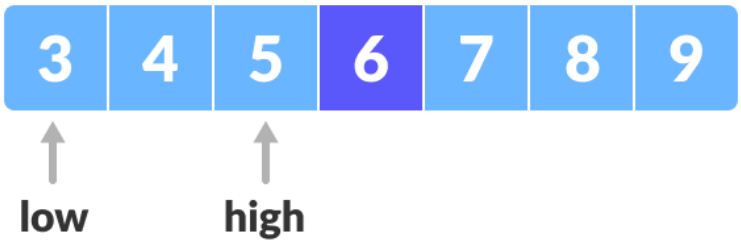
1) Binary Search Algorithm can be implemented using Recursion using divide and conquer approach.
2) The array in which searching is to be performed is: let x = 4 be the element to be searched.

3) Set two pointers low and high at the lowest and the highest positions respectively.

4) Find the middle element mid of the array ie. arr[(low + high)/2] = 6.



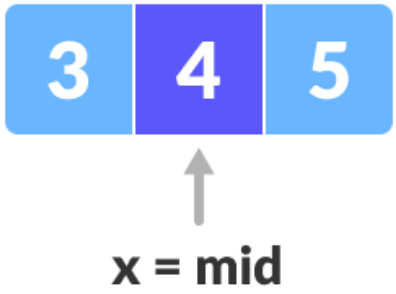
5) If x == mid, then return mid. Otherwise, compare the elements to be searched for with m.
6) If x > mid, compare x with the middle element of the elements on the right side of mid. This is done by setting low to low = mid + 1.
7) Else, compare x with the middle element of the elements on the left side of mid. This is done by setting high to high = mid - 1.



8) Repeat steps 4 to 7 until low meets high.

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	
Experiment No: 02	**Date:**	**Enrollment No: 92200133030**

9) x = 4 is found.



Algorithm: -

Programming Language: - C++

Code :-

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

int Binary_Search(vector<int> Array, int left, int right, int key) {

    if (left <= right) {
        int mid = left + (right - left) / 2;
```

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	
Experiment No: 02	**Date:**	**Enrollment No: 92200133030**

```
        if (key == Array[mid]) {
            return mid;
        }

        if (key < Array[mid]) {
            return Binary_Search(Array, left, mid - 1, key);
        }

        else {
            return Binary_Search(Array, mid + 1, right, key);
        }
    }
}

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);
```

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	
Experiment No: 02	**Date:**	**Enrollment No: 92200133030**

```
    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;
}
```

Output:-

```
PS C:\Users\Aaryan> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 2\" ; if ($?) { g++ Binary_Search.cpp -
o Binary_Search } ; if ($?) { .\Binary_Search }
Enter The Size of the Array :-
10
Enter The Element for the Array:-
Enter Element at index 0 : 25
Enter Element at index 1 : 48
Enter Element at index 2 : 89
Enter Element at index 3 : 74
Enter Element at index 4 : 56
Enter Element at index 5 : 125
Enter Element at index 6 : 478
Enter Element at index 7 : 365
Enter Element at index 8 : 247
Enter Element at index 9 : 29
Your Input Array Is :-
25 29 48 56 74 89 125 247 365 478
Enter the Key to Search In Array :- 247
247 Found at Index - 7 of Array.
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 2>
```

Space Complexity:- _____

Justification: -

_____

_____

_____

_____

_____

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	
Experiment No: 02	**Date:**	**Enrollment No: 92200133030**

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-

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: 92200133030**

Aim: Exponential Function with O(N) and O(logN)

IDE: Visual Studio Code

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

Theory: -

This approach calculates the exponential function x^n by multiplying x by itself n times in a loop. Starting from an initial value of 1, each iteration multiplies the result by x, continuing until we've multiplied x n times. This approach is straightforward and easy to implement, but it has a time complexity of O(n) since it requires n multiplications, making it inefficient for large values of n.

Algorithm: -

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);
}
```

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: 92200133030**

```
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 :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 3> cd "d:\Aryan Data\Usefull Data\Sem
ester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 3\" ; if ($?) { g++ Expotentail_Using_Naive.cpp -o Expotentail_Using_N
aive } ; if ($?) { .\Expotentail_Using_Naive }
Enter the Base of the Ecpotentail :- 25
Enter the Power of the Expotential :- 5
The 25 raised to 5 is 9765625 .
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 3>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

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: 92200133030**

Worst Case Time Complexity:- _____
Justification: -

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

Theory: -

This recursive method improves upon the naive approach by leveraging the properties of exponentiation through division:

- If n is even, the function computes x^(n/2) and squares the result.
- If n is odd, it computes x^[(n-1)/2], squares it, and then multiplies by x. Using divide and conquer

Algorithm: -

Programming Language: - C++

Code :-

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

long long Expotential(long base, long power) {
    if (power == 0) {
        return 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: 92200133030**

```
        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:-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 3> cd "d:\Aryan Data\Usefull Data\Sem
ester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 3\" ; if ($?) { g++ Expotentail_Using_DandC.cpp -o Expotentail_Using_D
andC } ; if ($?) { .\Expotentail_Using_DandC }
Enter the Base of the Ecpotentail :- 25
Enter the Power of the Expotential :- 5
The 25 raised to 5 is 9765625 .
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 3>
```

Space Complexity:- _____

Justification: -

_____

_____

_____

_____

_____

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: 92200133030**

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

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

Theory: -

This optimized divide-and-conquer method also leverages the properties of exponentiation::

- If n is even, the function computes x^(n/2) and squares the result.
- If n is odd, it computes x^[(n-1)/2], squares it, and then multiplies by x. By dividing n by 2 on O(logN). This approach is highly efficient and ideal for large values of n as it minimizes the number of multiplications needed to compute the result.

Algorithm: -

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: 92200133030**

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;
}
```

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: 92200133030**

Output:-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 3> cd "d:\Aryan Data\Usefull Data\Sem
ester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 3\" ; if ($?) { g++ Expotential_Using_DandC_Optimized.cpp -o Expotenti
al_Using_DandC_Optimized } ; if ($?) { .\Expotential_Using_DandC_Optimized }
Enter the Base of the Expotentail :- 25
Enter the Power of the Expotential :- 5
The 25 raised to 5 is 9765625 .
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 3>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-

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:**	**Enrollment No: 92200133030**

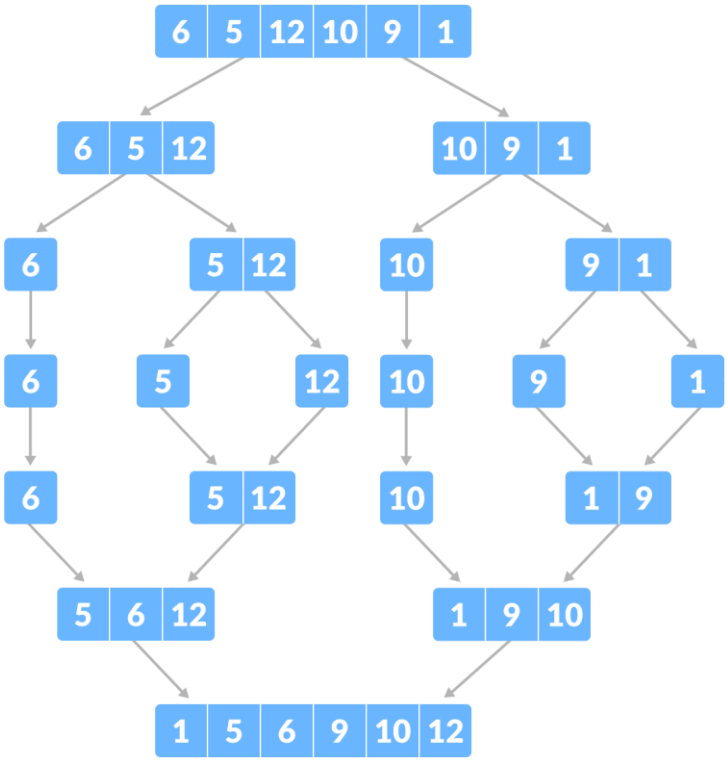
Aim: Implementing the Sorting Algorithms Using Divide and Conquer Approach

IDE: Visual Studio Code

Merge Sort

Theory: -

- Merge Sort is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm.
- Here, a problem is divided into multiple sub-problems. Each sub-problem is solved individually. Finally, sub-problems are combined to form the final solution.



Divide and Conquer Strategy

- Using the Divide and Conquer technique, we divide a problem into subproblems. When the solution to each subproblem is ready, we 'combine' the results from the subproblems to solve the main problem.
- Suppose we had to sort an array A. A subproblem would be to sort a sub-section of this array starting at index p and ending at index r, denoted as A[p..r].

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:**	**Enrollment No: 92200133030**

Divide

- If q is the half-way point between p and r, then we can split the subarray A[p..r] into two arrays A[p..q] and A[q+1, r].

Conquer

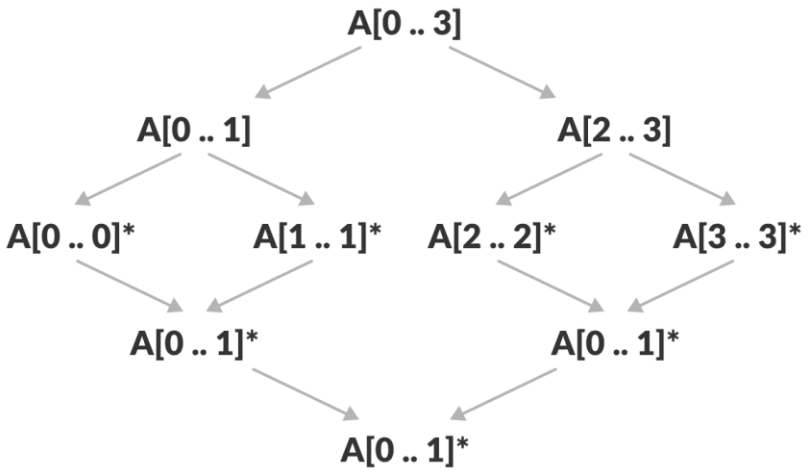
- In the conquer step, we try to sort both the subarrays A[p..q] and A[q+1, r]. If we haven't yet reached the base case, we again divide both these subarrays and try to sort them.

Combine

- When the conquer step reaches the base step and we get two sorted subarrays A[p..q] and A[q+1, r] for array A[p..r], we combine the results by creating a sorted array A[p..r] from two sorted subarrays A[p..q] and A[q+1, r].

Working of Merge Sort

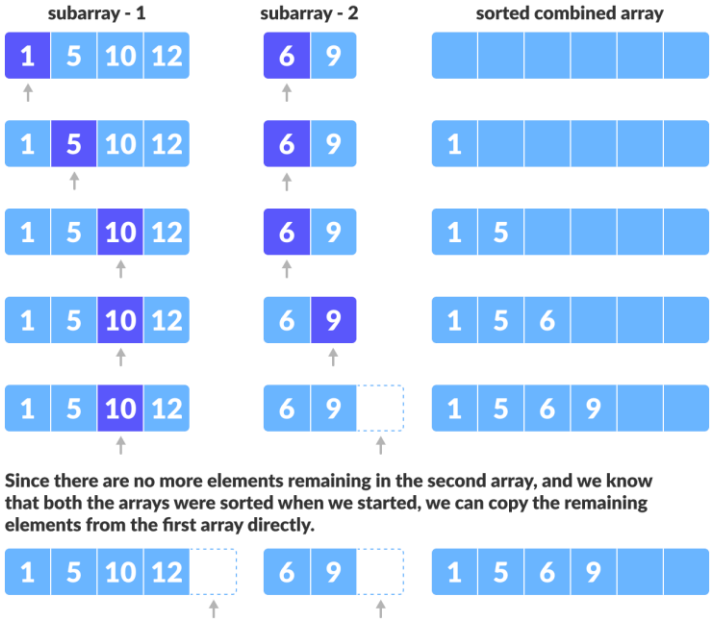
- The MergeSort function repeatedly divides the array into two halves until we reach a stage where we try to perform MergeSort on a subarray of size 1 i.e. p == r.
- After that, the merge function comes into play and combines the sorted arrays into larger arrays until the whole array is merged.
- To sort an entire array, we need to call MergeSort(A, 0, length(A)-1).
- As shown in the image below, the merge sort algorithm recursively divides the array into halves until we reach the base case of array with 1 element. After that, the merge function picks up the sorted sub-arrays and merges them to gradually sort the entire array.



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:**	**Enrollment No: 92200133030**

The merge Step

- Every recursive algorithm is dependent on a base case and the ability to combine the results from base cases. Merge sort is no different. The most important part of the merge sort algorithm is, you guessed it, merge step.
- The merge step is the solution to the simple problem of merging two sorted lists(arrays) to build one large sorted list(array).
- The algorithm maintains three-pointers, one for each of the two arrays and one for maintaining the current index of the final sorted array.



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:**	**Enrollment No: 92200133030**

Algorithm: -

Programming Language: - C++

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;
}
```

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:**	**Enrollment No: 92200133030**

```
  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) {
          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);
```

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:**	**Enrollment No: 92200133030**

```
        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 :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 4> cd "
d:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 4\" ; if (
$?) { g++ Merge_Sort.cpp -o Merge_Sort } ; if ($?) { .\Merge_Sort }
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
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 4>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

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:**	**Enrollment No: 92200133030**

Worst Case Time Complexity:- _____
Justification: -

Quick Sort

Theory: -

- Quicksort is a sorting algorithm based on the divide and conquer approach where
  1) An array is divided into subarrays by selecting a pivot element (element selected from the array).
  2) While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.
  3) The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.
  4) At this point, elements are already sorted. Finally, elements are combined to form a sorted array.

Working of Quick Sort

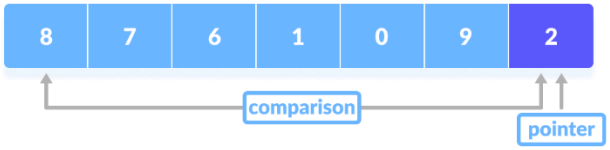
1) **Select the Pivot Element**
   - There are different variations of quicksort where the pivot element is selected from different positions. Here, we will be selecting the rightmost element of the array as the pivot element.

2) **Rearrange the Array**
   - Now the elements of the array are rearranged so that elements that are smaller than the pivot are put on the left and the elements greater than the pivot are put on the right.

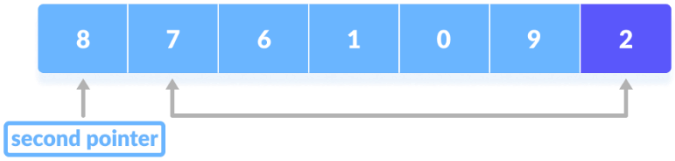
   - **Here's how we rearrange the array:**

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:**	**Enrollment No: 92200133030**

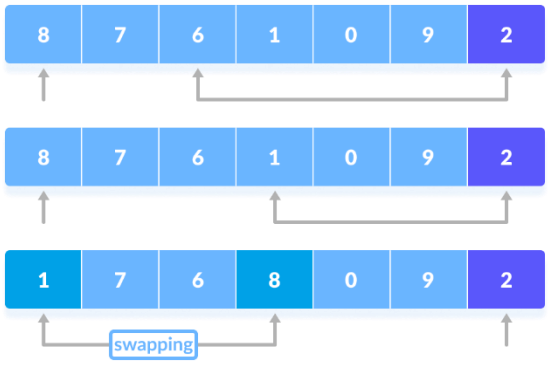
1) A pointer is fixed at the pivot element. The pivot element is compared with the elements beginning from the first index.



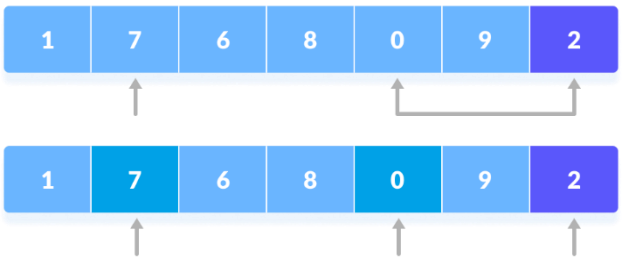
2) If the element is greater than the pivot element, a second pointer is set for that element.



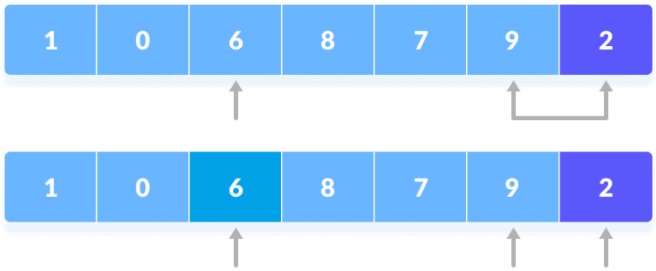
3) Now, pivot is compared with other elements. If an element smaller than the pivot element is reached, the smaller element is swapped with the greater element found earlier.



4) Again, the process is repeated to set the next greater element as the second pointer. And, swap it with another smaller element.



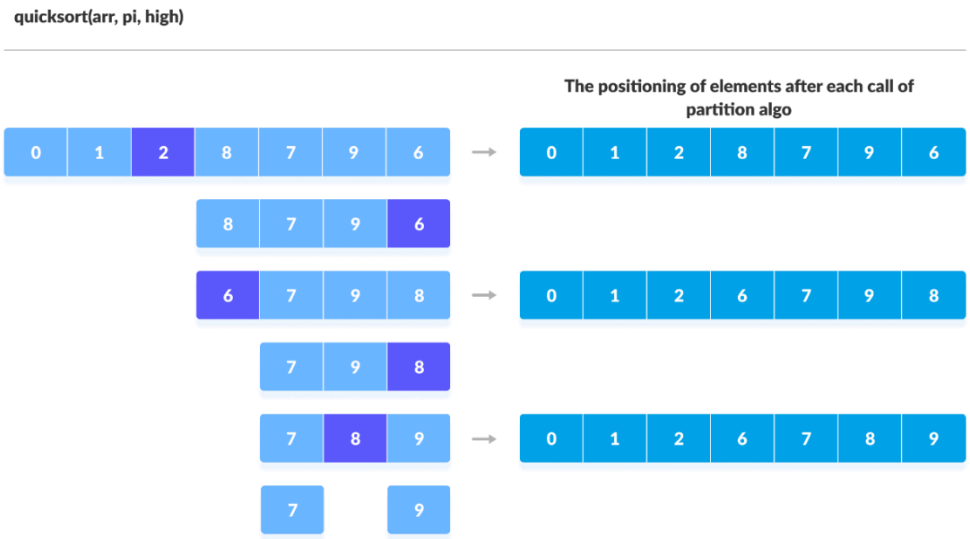
5) The process goes on until the second last element is reached.



6) Finally, the pivot element is swapped with the second pointer.

3) Divide Subarrays

- Pivot elements are again chosen for the left and the right sub-parts separately. And, step 2 is repeated.



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:**	**Enrollment No: 92200133030**

Algorithm: -

Programming Language: - C++

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;
}
```

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:**	**Enrollment No: 92200133030**

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 4> cd "d:
\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 4\" ; if ($?)
{ g++ Quick_sort.cpp -o Quick_sort } ; if ($?) { .\Quick_sort }
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
PS D:\Aryan Data\Usefull Data\Semester - 5\Semester-5\Design And Analysis of Algorithms\Lab - Manual\Experiment - 4>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-

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:**	**Enrollment No: 92200133030**

Aim: Implementing Application-based Algorithm using the D&C Approach

IDE: Visual Studio Code

Karatsuba Algorithm

Theory: -

The Karatsuba algorithm is based on **divide and conquer**. It decomposes the multiplication of two large numbers into smaller, simpler multiplications by splitting each number into parts. Here's a step-by-step breakdown of how it works:

Step 1: Splitting the Numbers

Given two n-digit numbers X and Y:

- Split X and Y into two halves:
  - X can be represented as $X_1 \times 10^m + X0$
  - Y can be represented as $Y_1 \times 10^m + Y0$

Here:

- X_1 and X_1 are the higher parts of X and Y.
- X_0 and X_0 are the lower parts of X and Y.
- m is chosen to be about half of n.

For example, if X=1234 and Y=5678, we could split them as:

- $X = 12 \times 10^2 + 34$
- $Y = 56 \times 10^2 + 78$

Step 2: Recursive Multiplication

The product P= X × Y can be expanded as :

$P = (X_1 \times 10^m + X_0) \times (Y_1 \times 10^m + Y_0)$

Expanding this, we get:

$P = X_1 \times Y_1 \times 10^{2m} + (X_1 \times Y_0 + X_0 \times Y_1) \times 10^m + X_0 \times Y_0$

The Karatsuba algorithm optimizes this by rewriting the middle term:

1. Compute $Z_0 = X_0 \times Y_0$
2. Compute $Z_2 = X_1 \times Y_1$
3. Compute $Z_1 = (X_1 + X_0) \times (Y_1 + Y_0) - Z_2 - Z_0$

Thus, we have:

$P = Z_2 \times 10^{2m} + Z_1 \times 10^m + Z_0$

Step 3: Recursive Calls and Time Complexity

Each multiplication now requires only three multiplications of half-size numbers instead of four, as in the standard multiplication. This division and recursive approach lead to a recurrence relation that results in a time complexity of $O(n \wedge \log_2 3) \approx O(n \wedge 1.585)$.

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:**	**Enrollment No: 92200133030**

Algorithm: -

Programming Language: - C++

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;
    }
```

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:**	**Enrollment No: 92200133030**

```
    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 :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 5> cd "d:\Aryan Data\Usefull Data\Sem
ester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 5\" ; if ($?) { g++ Karatsuba_Multiplication.cpp -o Karatsuba_Multipli
cation } ; if ($?) { .\Karatsuba_Multiplication }
Enter the First Number :- 123456
Enter the Second Number :- 456789
The Product of 123456 and 456789 is: 56393342784
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 5>
```

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:**	**Enrollment No: 92200133030**

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-

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:**	**Enrollment No: 92200133030**

Aim: Implementing Knapsack Problem using Greedy Approach

IDE: Visual Studio Code

0/1 Knapsack Problem

Theory: -

The **0/1 Knapsack Problem** is a combinatorial optimization problem, typically solved using dynamic programming due to its structure. However, it's often contrasted with the Fractional Knapsack problem to illustrate the limits of the greedy approach.

- **Problem Definition**:
  - Given n items, each with a weight wi and a profit/value vi.
  - There is a knapsack with a maximum weight capacity of W.
  - The objective is to maximize the total profit without exceeding the weight capacity W.
  - In the 0/1 knapsack problem, each item can either be fully included in the knapsack or not included at all, hence the "0/1" nature.
- **Greedy Approach and Its Limitations**:
  - A greedy approach would typically sort items by their value-to-weight ratio (i.e., vi/wi) and then add items to the knapsack in descending order of this ratio.
  - Unfortunately, in the 0/1 knapsack problem, this approach does not always yield the optimal solution. This is because once an item is either chosen or discarded, we cannot split it, which might lead to suboptimal use of the knapsack's capacity.
  - Example: Consider two items where item 1 has a high value-to-weight ratio but would not allow any other item to fit if taken; a lower ratio item combination could achieve a higher overall value.

Algorithm: -

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:**	**Enrollment No: 92200133030**

Programming Language: - C++

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;
}
```

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:**	**Enrollment No: 92200133030**

```
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;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 6> cd "d:\Aryan Data\Usefull Data\Sem
ester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 6\" ; if ($?) { g++ 0_1_Knapsack.cpp -o 0_1_Knapsack } ; if ($?) { .\0
_1_Knapsack }
Total Profit Is : 180
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 6>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

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:**	**Enrollment No: 92200133030**

Fractional Knapsack Problem :-

Theory: -

The **Fractional Knapsack Problem** is similar to the 0/1 Knapsack Problem but allows for items to be split into fractions, making it more amenable to a greedy approach.

1. **Problem Definition**:
   - Given n items, each with a weight wi and a profit/value vi.
   - A knapsack has a weight capacity W.
   - The objective is to maximize the total profit, with the added ability to take fractions of items if necessary.
2. **Greedy Approach**:
   - In the fractional knapsack, we can break items into smaller parts, allowing the use of a **greedy algorithm** effectively.
   - The steps are:
     - Calculate the value-to-weight ratio (vi/wi) for each item.
     - Sort items in descending order of this ratio.
     - Start adding items to the knapsack from the top of the sorted list until the knapsack cannot take the full weight of the next item.
     - If an item cannot fully fit, add the fraction of it that maximizes the remaining capacity W.
   - Since fractions of items are allowed, this strategy always yields the optimal solution.

Algorithm: -

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:**	**Enrollment No: 92200133030**

Programming Language: - C++

Code :-

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

double Fractional_Knapsack(vector<int>& Profit, vector<int>& Weight, int total_weight) {
    vector<pair<double, int>> Profit_weight;
    for (int i = 0; i < Profit.size(); i++) {
        Profit_weight.push_back({ (double)Profit[i] / Weight[i], Weight[i] });
    }
    sort(Profit_weight.begin(), Profit_weight.end(), [](pair<double, int>& a, pair<double, int>&
b) {
        return a.first > b.first;
        });

double Max_Profit = 0.0;
    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 {
            Max_Profit += Profit_weight[i].first * total_weight;
            break;
        }
    }

    return Max_Profit;
}

int main() {
    vector<int> Profit = { 60, 100, 120 };
    vector<int> Weight = { 10, 20, 40 };
    int total_weight = 40;
    double total_profit = Fractional_Knapsack(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:**	**Enrollment No: 92200133030**

Output :-

```
PS C:\Users\Aaryan> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 6\" ; if ($?) { g++ Fractional_Knapsack
.cpp -o Fractional_Knapsack } ; if ($?) { .\Fractional_Knapsack }
Total Profit Is : 190
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 6>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-

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:**	**Enrollment No: 92200133030**

Aim: Implementing application-based algorithms using a Greedy Approach

IDE: Visual Studio Code

I. Job Scheduling Problem

Theory: -

- The Job Scheduling Problem (JSP) is a classic optimization problem that involves scheduling a set of jobs on resources (like machines) in a way that optimizes a particular objective, such as minimizing total completion time, minimizing delays, or maximizing resource utilization. A greedy algorithm approach is a common and efficient way to address certain variants of JSP, as it focuses on building an optimal solution step-by-step by making a locally optimal choice at each step with the hope of reaching a globally optimal solution.

1. Problem Definition

- Given a set of n jobs, each with a specific processing time and possibly a deadline, the objective is to assign these jobs to resources or time slots in such a way that some objective function, f, is minimized or maximized. Some common objectives include:
  - Minimizing total completion time.
  - Minimizing the maximum completion time (makespan).
  - Minimizing delays or tardiness, where each job has a due date.

2. Greedy Approach to Job Scheduling

- The greedy approach to JSP is based on a strategy of prioritizing jobs according to a specific criterion and scheduling them in that order. This approach can provide optimal solutions for certain types of job scheduling problems, especially when the objective function aligns well with the chosen greedy criterion.

3. Greedy Criteria for Different Variants of JSP

- In job scheduling problems, different greedy criteria can be selected depending on the objective:
  - Shortest Processing Time First (SPT): For minimizing total completion time, a common greedy criterion is to select the job with the shortest processing time next. This heuristic works well for reducing average job completion time.
  - Earliest Deadline First (EDF): For minimizing tardiness or meeting deadlines, jobs are sorted by their due dates, and the job with the earliest due date is scheduled first. This strategy can be particularly effective when minimizing the number of late jobs.
  - Highest Profit First: In situations where jobs have associated profits or rewards, a greedy algorithm can prioritize jobs with the highest profit-to-processing-time ratio, scheduling them in order to maximize total profit.

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:**	**Enrollment No: 92200133030**

4. Greedy Algorithm for Job Scheduling

- For a general greedy algorithm for JSP:
  1. Sort the Jobs: Sort the jobs based on a chosen criterion, such as processing time, deadline, or profit.
  2. Iterate Through the Jobs: For each job in the sorted list:
     - Assign the job to the next available time slot or resource, if feasible.
     - Update the schedule or resource allocation.
  3. Repeat Until All Jobs are Scheduled or until constraints (such as deadlines or resource limits) prevent further scheduling.

4. Greedy Solution Properties

- The greedy approach to JSP is efficient and straightforward, often requiring O(nlogn) time complexity for sorting, followed by O(n) for iterating through the jobs. Greedy algorithms work best when:
  - The problem has the optimal substructure property (i.e., an optimal solution to the problem contains optimal solutions to its subproblems).
  - The problem has no overlapping subproblems, making it ideal for direct, step-by-step scheduling decisions.

Algorithm: -

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:**	**Enrollment No: 92200133030**

Programming Language: - C++

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 };
}
```

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:**	**Enrollment No: 92200133030**

```
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 :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 7> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-An
alysis-of-Algorithms\Lab - Manual\Experiment - 7\" ; if ($?) { g++ Job_Scheduling.cpp -o Job_Scheduling } ; if ($?) { .\Job_Scheduling }
We can Perform 7 and Earn Profit of Rs. 342 .
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 7>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

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:**	**Enrollment No: 92200133030**

Worst Case Time Complexity:- _____
Justification: -

_____
_____
_____
_____
_____

II. Activity Selection Problem :-

Theory: -

- The **Activity Selection Problem** is a classic optimization problem in computer science and operations research that aims to select the maximum number of non-overlapping activities from a given set, each with a start and end time. This problem is commonly solved using the **greedy approach**, which involves making a series of locally optimal choices to reach a global optimum.

❖ **Problem Definition**

- Given a set of activities, where each activity iii has a **start time** si and an **end time** ei, the objective is to select the maximum number of activities that do not overlap. In other words, if we choose an activity i, then no other selected activity j can have a start time sj that is less than ei.

Approach Using the Greedy Method

- The greedy method is particularly effective for this problem because it allows us to make an optimal choice at each step, without needing to look ahead or reconsider previous choices. The greedy approach for the Activity Selection Problem works as follows:
  1. **Sort the Activities by End Time**: The first step in the greedy approach is to sort all activities in ascending order based on their end times eie_iei. Sorting by end times helps ensure that the activity with the earliest finish time is considered first, maximizing the time remaining for other activities.
  2. **Select the Activity with the Earliest Finish Time**: After sorting, the first activity in the sorted list (the one that finishes earliest) is always selected, as it leaves the maximum room for other activities. We choose this activity and then discard any other activities that overlap with it.
  3. **Repeat the Process**: For each subsequent activity in the sorted list, check if its start time sjs_jsj is greater than or equal to the end time eie_iei of the previously selected activity. If it is, select this activity and update the end time to the finish time of this activity.
  4. **Continue Until All Activities Are Examined**: Repeat the above process until all activities have been considered.

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:**	**Enrollment No: 92200133030**

Why the Greedy Approach Works

The greedy approach works well for the Activity Selection Problem because:

- **Minimizing End Time Maximizes Remaining Time**: By always selecting the activity that finishes the earliest (among the activities that can start next), we maximize the remaining time in which other activities can be scheduled.
- **Locally Optimal Choices Lead to Global Optimum**: The decision to select the earliest finishing activity at each step (greedily) leads to the overall optimal solution, as it allows the maximum possible number of non-overlapping activities.

Algorithm: -

Programming Language: - C++

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:**	**Enrollment No: 92200133030**

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;
}
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;
}
```

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:**	**Enrollment No: 92200133030**

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 7> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-An
alysis-of-Algorithms\Lab - Manual\Experiment - 7\" ; if ($?) { g++ Activity_Selection.cpp -o Activity_Selection } ; if ($?) { .\Activity_Selection }
Selected Activities are :-
Activity ID :- 2 Start Time: 1, End Time: 2
Activity ID :- 3 Start Time: 3, End Time: 4
Activity ID :- 5 Start Time: 5, End Time: 7
Activity ID :- 6 Start Time: 8, End Time: 9
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 7>
```

Space Complexity:- _____

Justification: -

Time Complexity:

Best Case Time Complexity: _____

Justification: -

Worst Case Time Complexity:- _____

Justification: -

Conclusion:-

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:**	**Enrollment No: 92200133030**

Aim: Implementing the Longest Common Sub-sequence using Dynamic Programming Approach

IDE: Visual Studio Code

Longest Common Sub-sequence

Theory: -

- The Longest Common Subsequence (LCS) problem is a classic problem in computer science and bioinformatics that involves finding the longest sequence that can be derived from two given sequences without altering the order of their elements. It is an important problem in string processing and is widely used in applications such as text comparison, DNA sequence analysis, and data diffing tools.

1. Problem Definition

- Given two sequences, the task is to find the longest subsequence that is present in both sequences. A subsequence is a sequence derived by deleting some or no elements from the original sequence without rearranging the order of the remaining elements.
  - For example:
    Sequence 1: "abcde"
    Sequence 2: "ace"
    The LCS is "ace" because it is the longest sequence that appears in the same order in both.

2. Applications of LCS

- Text comparison: Helps detect similarities between text files or documents.
- DNA sequence alignment: Identifies common patterns in genetic sequences.
- Version control systems: Finds differences between versions of files.
- Spell checkers: Matches words with similar spellings.

3. Approach to find LCS :-

- **The most common approach to solving the LCS problem is using Dynamic Programming.**
  1. Define a DP table: Create a table where each cell represents the length of the LCS for two substrings of the input sequences.
  2. Initialize the table: Set the first row and column to 0 since the LCS of any sequence with an empty sequence is 0.
  3. Use a recurrence relation:
     - If the current characters of both sequences match:
       $dp[i][j]=dp[i-1][j-1]+1$
     - If they don't match:

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:**	**Enrollment No: 92200133030**

dp[i][j]=max(dp[i−1][j],dp[i][j−1])

4. Retrieve the result: The value in the bottom-right cell of the table gives the length of the LCS.

Algorithm: -

Programming Language: - C++

Code :-

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

string Longest_Common_Subsequence(string X, string Y) {
    int m = X.length();
    int n = Y.length();
    int dp[m + 1][n + 1];

    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
```

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:**	**Enrollment No: 92200133030**

```
            if (i == 0 || j == 0) {
                dp[i][j] = 0;
            }
            else if (X[i - 1] == Y[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            }
            else {
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }

    string lcsString;
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcsString = X[i - 1] + lcsString;
            i--;
            j--;
        }
        else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
        }
        else {
            j--;
        }
    }

    return lcsString;
}

int main() {

    string X, Y;

    cout << "Enter The First String :- ";
    cin >> X;

    cout << "Enter The Second String :- ";
    cin >> Y;

    string result = Longest_Common_Subsequence(X, Y);
    cout << "Length of LCS is " << result.length() << endl;
    cout << "LCS is: " << result << endl;
```

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:**	**Enrollment No: 92200133030**

```
    return 0;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 8> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-An
alysis-of-Algorithms\Lab - Manual\Experiment - 8\" ; if ($?) { g++ Longest_Common_Subsequence.cpp -o Longest_Common_Subsequence } ; if ($?) { .\Longest_Common_Subs
equence }
Enter The First String :- AGGTAB
Enter The Second String :- GXTXAYB
Length of LCS is 4
LCS is: GTAB
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 8>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-

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:**	**Enrollment No: 92200133030**

Aim: Implementing 0/1 Knapsack Problem using Dynamic Programming Approach

IDE: Visual Studio Code

0/1 Knapsack Problem Using Dynamic Programming

Theory: -

- The 0/1 Knapsack Problem is a classic optimization problem in computer science and operations research. It involves selecting items with given weights and values to maximize the total value while staying within a weight limit.

1. Problem Definition

- You are given:
  1. A set of n items, where each item has:
     - A weight w[i]
     - A value v[i]
  2. A knapsack with a maximum weight capacity W.

- The task is to determine the maximum value you can achieve by selecting a subset of items such that the total weight does not exceed W. Each item can either be included once (1) or not included at all (0), which is why it is called the 0/1 Knapsack Problem.

2. Dynamic Programming Approach :-

- The problem has overlapping subproblems and optimal substructure properties, making it suitable for a **dynamic programming (DP)** solution.

1. **Define the DP Table**: Let dp[i][w] represent the maximum value that can be obtained using the first iii items with a weight limit of www.
2. **Recurrence Relation**:
   For each item i, we have two choices:
   a. **Exclude the item**:
      The maximum value remains the same as for the previous item with the same weight limit.
      $dp[i][w]=dp[i-1][w]$
   b. **Include the item (if the weight allows)**:
      The value is the item's value plus the maximum value achievable with the remaining weight.
      $dp[i][w]=v[i-1]+dp[i-1][w-w[i-1]]$

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:**	**Enrollment No: 92200133030**

The final value is the maximum of these two choices:
dp[i][w]=max(dp[i−1][w],v[i−1]+dp[i−1][w−w[i−1]])

3. **Initialization**:
   dp[0][w]= 0 for all w, since no items mean no value.
   dp[i][0]= 0 for all iii, since a knapsack with 0 capacity can hold no value.
4. **Final Result**:
   The value at dp[n][W] gives the maximum value that can be achieved for the given weight limit W.

Algorithm: -

Programming Language: - C++

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));
```

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:**	**Enrollment No: 92200133030**

```
    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;
 }
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 9> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 9\" ; if ($?) { g++ 0_1_Knapsack_DP.cpp -o 0_1_Knapsack_DP } ; if ($?) { .\0_1_Knapsack_DP }
DP Table (Max Value for Each Capacity):
   0      0      0      0      0      0
   0      0    3015    3015    3015    3015
   0      0    3015    4026    4026    7041
   0      0    3015    4026    5789    7041
   0      0    3015    4026    5789    7041
Maximum value in Knapsack = 7041
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 9>
```

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:**	**Enrollment No: 92200133030**

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-

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:**	**Enrollment No: 92200133030**

Aim: Implementing Matrix Chain Multiplication using a Dynamic Programming Approach

IDE: Visual Studio Code

Implementing Matrix Chain Multiplication using a Dynamic Programming Approach

Theory: -

- Matrix Chain Multiplication is a classic optimization problem that seeks to find the most efficient way to multiply a sequence of matrices. The goal is to minimize the number of scalar multiplications required. Since matrix multiplication is associative, the order in which the matrices are multiplied can significantly affect the total computational cost.

1. Problem Definition

- Given nnn matrices $A1, A2, \ldots, An$ with dimensions $p0 \times p1, p1 \times p2, \ldots, pn-1 \times pn$, determine the optimal parenthesization to minimize the total number of scalar multiplications.

2. Key Concepts:-

a. Associativity of Matrix Multiplication:

- The multiplication (A1A2)A3 is equivalent to A1(A2A3) but can have different computational costs depending on the dimensions.

b. Cost of Multiplication:

- Multiplying two matrices A of dimensions $p \times q$ and B of dimensions $q \times r$ requires $p \times q \times r$ scalar multiplications.

c. Dynamic Programming Approach:

- The problem is broken into subproblems where the solution to smaller chains is reused to solve larger chains.
- We use a table to store the minimum number of scalar multiplications for each subproblem.

3. Dynamic Programming Approach :-

1. **Define the Problem:**
   - Let m[i][j] represent the minimum number of scalar multiplications required to multiply the subchain $Ai, Ai+1, \ldots, Aj$.
2. **Base Case:**
   - If i=ji = ji=j, a single matrix does not require any multiplication: m[i][i]=0m[i][i] = 0m[i][i]=0
3. **Recursive Relation:**
   - For a subchain Ai to Aj, split it into two parts Ai to Ak and Ak+1 to Aj for $i \leq k < j$: $m[i][j] = \min(i \leq k < j)\{m[i][k] + m[k+1][j] + pi-1 \times pk \times pj\}$

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:**	**Enrollment No: 92200133030**

- Here, p_{i-1}, p_k, p_j are the dimensions involved in multiplying the two resulting matrices.

4. **Iterative Computation:**
   - The chain length L is varied from 2 to n (length of subchains).
   - For each subchain, compute m[i][j] for all valid i and j.
5. **Result:**
   - The optimal cost for multiplying the entire chain is stored in m[1][n].

Algorithm: -

Programming Language: - C++

Code :-

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

int matrixMultiplication(vector<int>& arr) {
    int n = arr.size();
```

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:**	**Enrollment No: 92200133030**

```
    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 :-

```
PS C:\Users\Aaryan> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 10\" ; if ($?) { g++ Matrix_Chain_Multi
plication.cpp -o Matrix_Chain_Multiplication } ; if ($?) { .\Matrix_Chain_Multiplication }
Enter the Number of Matrix :- 4
Enter the Number of Rows and Columns for Matrix 1 :- 5 4
Enter the Number of Rows and Columns for Matrix 2 :- 4 6
Enter the Number of Rows and Columns for Matrix 3 :- 6 2
Enter the Number of Rows and Columns for Matrix 4 :- 2 7
   0    0  120   88  158
   0    0    0   48  104
   0    0    0    0   84
   0    0    0    0    0
   0    0    0    0    0
158
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 10>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

	**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:**	**Enrollment No: 92200133030**

Conclusion:-

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing String Matching Approach	
Experiment No: 11	**Date:**	**Enrollment No: 92200133030**

Aim: Implementing String Matching Approach

IDE: Visual Studio Code

1. **Implement the Naive based approach to find the pattern within the string**

Theory: -

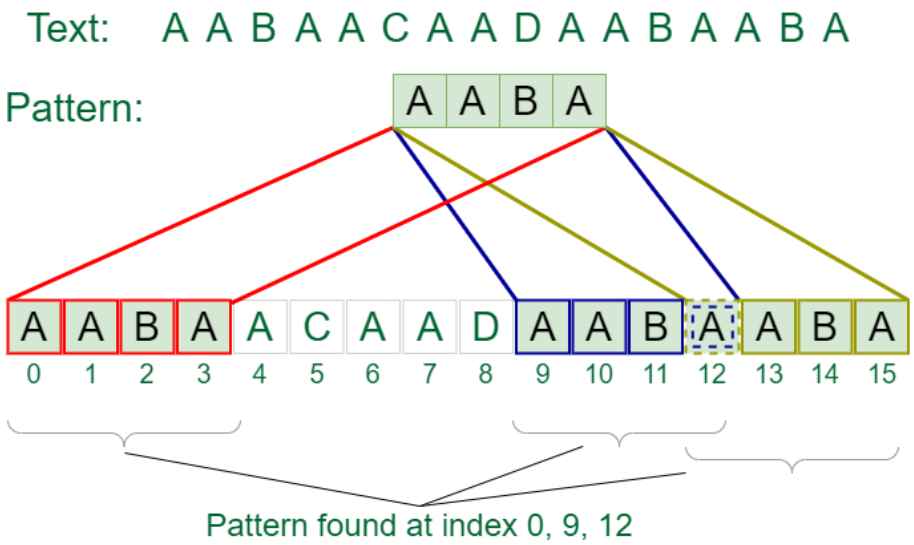
- The problem involves identifying all the positions where a given pattern appears in a longer text. Given the text of length n and a pattern of length mmm (n > m), the goal is to efficiently find all the starting indices in the text where the pattern matches.

1. **Key Concepts:-**

- **Pattern Matching:** The task of locating a specific sequence of characters (the pattern) within a larger sequence (the text).
- **Occurrences:** The indices in the text where the first character of the pattern aligns with a matching substring.

2. Naive Approach:-

- A straightforward way to solve the problem is to:
  1. Slide the pattern over the text from the beginning to n−m.
  2. At each position, compare the substring of length mmm with the pattern.
  3. If all characters match, record the starting index.



Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing String Matching Approach	
Experiment No: 11	**Date:**	**Enrollment No: 92200133030**

Algorithm: -

Programming Language: - C++

Code :-

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

void search(string& pat, string& txt) {
    int M = pat.size();
    int N = txt.size();

    for (int i = 0; i <= N - M; i++) {
        int j;
        for (j = 0; j < M; j++) {
            if (txt[i + j] != pat[j]) {
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing String Matching Approach	
Experiment No: 11	**Date:**	**Enrollment No: 92200133030**

```
                break;
            }
        }
        if (j == M) {
            cout << "Pattern found at index " << i << endl;
        }
    }
}

int main() {
    string txt1 = "AABAACAADAABAABA";
    string pat1 = "AABA";
    cout << "Example 1: " << endl;
    search(pat1, txt1);
    return 0;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11\" ; if ($?) { g++ Naive_Approach.cpp -o Naive_Approach } ; if ($?) { .\Naive_Approach }
Example 1:
Pattern found at index 0
Pattern found at index 9
Pattern found at index 12
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11>
```

Space Complexity:- _____

Justification: -

_____

_____

_____

_____

_____

Time Complexity:

Best Case Time Complexity: _____

Justification: -

_____

_____

_____

_____

_____

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing String Matching Approach	
Experiment No: 11	**Date:**	**Enrollment No: 92200133030**

Worst Case Time Complexity:- _____
Justification: -

_____
_____
_____
_____
_____

2. Implement the Rabin-Karp approach to find the pattern within the string

Theory: -

- The Rabin-Karp algorithm is an efficient pattern-matching algorithm that uses hashing to find all occurrences of a pattern in a given text. Instead of comparing substrings character by character, it compares their hash values, significantly improving performance in many cases.

Key Concepts:-

1) Hash Function:

- A hash function maps a string to a numeric value.
- For this algorithm, a rolling hash function is used, which allows efficient computation of hash values for consecutive substrings in O(1) time.

2) Rolling Hash:

- The hash of a substring s[i:i+m] is computed based on the hash of s[i−1:i+m−1].
- Formula: $\text{hash}(s[i:i+m]) = (\text{base} \cdot (\text{hash}(s[i-1:i+m-1]) - \text{ord}(s[i-1]) \cdot \text{base}^{m-1}) + \text{ord}(s[i+m-1])) \bmod \text{modulus}$
- This avoids recalculating the entire hash from scratch.

3) Collision:

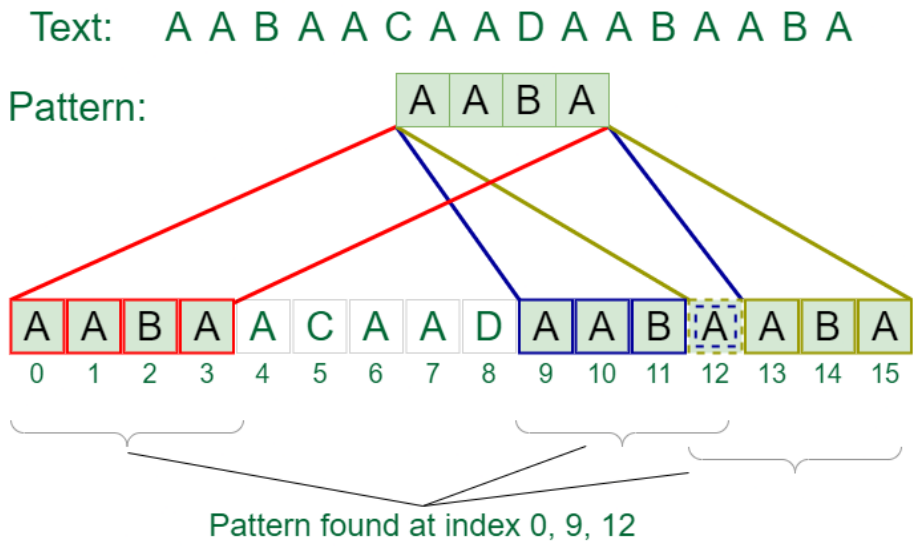
- Two different strings may have the same hash value due to hash collisions.
- To verify a match, the algorithm performs a character-by-character comparison of the substring and the pattern.

Steps of the Rabin-Karp Algorithm:-

1. Compute the hash value of the pattern ($h_{pattern}$) and the first window of the text (h_{text}).
2. Slide the pattern over the text:
    - Update the hash value of the current window using the rolling hash formula.
    - Compare $h_{pattern}$ and h_{text}.

- If the hash values match, perform a direct string comparison to confirm.

3. Record the index if a match is found.
4. Repeat until the entire text has been scanned.



Algorithm: -

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing String Matching Approach	
Experiment No: 11	**Date:**	**Enrollment No: 92200133030**

Programming Language: - C++

Code :-

```
#include <bits/stdc++.h>
using namespace std;
#define d 256

void search(char pat[], char txt[], int q)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int i, j;
    int p = 0;
    int t = 0;
    int h = 1;
    for (i = 0; i < M - 1; i++)
        h = (h * d) % q;
    for (i = 0; i < M; i++) {
        p = (d * p + pat[i]) % q;
        t = (d * t + txt[i]) % q;
    }
    for (i = 0; i <= N - M; i++) {

        if (p == t) {
            for (j = 0; j < M; j++) {
                if (txt[i + j] != pat[j]) {
                    break;
                }
            }
            if (j == M)
                cout << "Pattern found at index " << i
                << endl;
        }

        if (i < N - M) {
            t = (d * (t - txt[i] * h) + txt[i + M]) % q;
            if (t < 0)
                t = (t + q);
        }
    }
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing String Matching Approach	
Experiment No: 11	**Date:**	**Enrollment No: 92200133030**

```
}

int main()
{
    char txt[] = "AABAACAADAABAABA";
    char pat[] = "AABA";

    int q = INT_MAX;

    search(pat, txt, q);
    return 0;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11\" ; if ($?) { g++ Rabin-Karp.cpp -o Rabin-Karp } ; if ($?) { .\Rabin-Karp }
Pattern found at index 0
Pattern found at index 9
Pattern found at index 12
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11>
```

Space Complexity:- _____

Justification: -

Time Complexity:

Best Case Time Complexity: _____

Justification: -

	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing String Matching Approach	
Experiment No: 11	**Date:**	**Enrollment No: 92200133030**

Worst Case Time Complexity:- _____

Justification: -

Conclusion:-

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Graph Traversal Approach	
Experiment No: 12	**Date:**	**Enrollment No: 92200133030**

Aim: Implementing the Graph Traversal Approach

IDE: Visual Studio Code

1. Implement the Breadth-First Search

Theory: -

- Breadth First Search (BFS) is a fundamental graph traversal algorithm. It begins with a node, then first traverses all its adjacent. Once all adjacent are visited, then their adjacent are traversed. This is different from DFS in a way that closest vertices are visited before others. We mainly traverse vertices level by level. A lot of popular graph algorithms like Dijkstra's shortest path, Kahn's Algorithm, and Prim's algorithm are based on BFS. BFS itself can be used to detect cycle in a directed and undirected graph, find shortest path in an unweighted graph and many more problems.

BFS from a Given Source:

- The algorithm starts from a given source and explores all reachable vertices from the given source. It is similar to the Breadth-First Traversal of a tree. Like tree, we begin with the given source (in tree, we begin with root) and traverse vertices level by level using a queue data structure. The only catch here is that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.

- **Initialization**: Enqueue the given source vertex into a queue and mark it as visited.

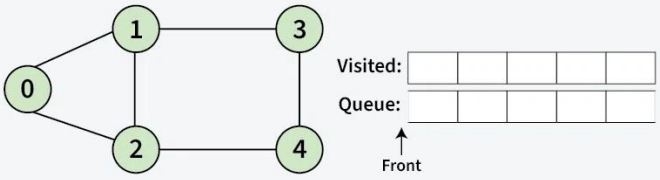
1. **Exploration**: While the queue is not empty:
   - Dequeue a node from the queue and visit it (e.g., print its value).
   - For each unvisited neighbor of the dequeued node:
     - Enqueue the neighbor into the queue.
     - Mark the neighbor as visited.
2. **Termination**: Repeat step 2 until the queue is empty.

- This algorithm ensures that all nodes in the graph are visited in a breadth-first manner, starting from the starting node.

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Graph Traversal Approach	
Experiment No: 12	**Date:**	**Enrollment No: 92200133030**

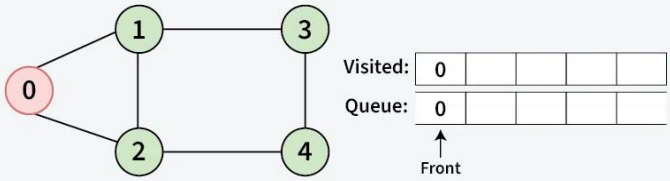
How Does the BFS Algorithm Work :-

01 Step Initially queue and visited array are empty.



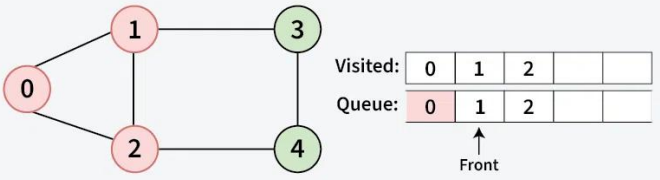
BFS on Graph

02 Step Push 0 into queue and mark it visited.



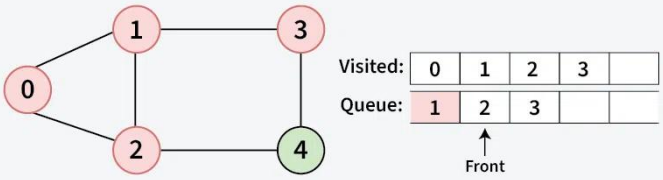
BFS on Graph

03 Step Remove 0 from the front of queue and visit the unvisited neighbours and push them into queue.



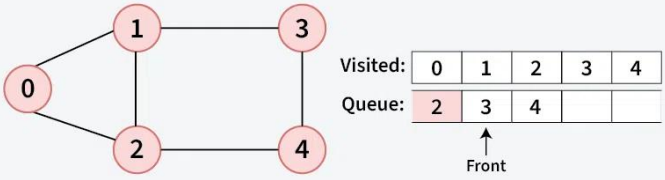
BFS on Graph

04 Step Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.



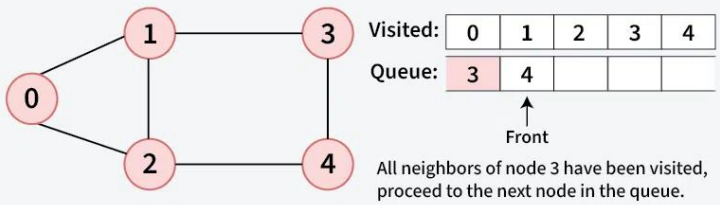
BFS on Graph

05 Step Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.



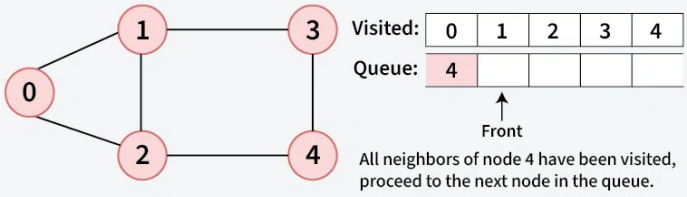
BFS on Graph

06 Step Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.



BFS on Graph

07 Step Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.



BFS on Graph

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Graph Traversal Approach	
Experiment No: 12	**Date:**	**Enrollment No: 92200133030**

BFS algorithm :-

- A standard BFS implementation puts each vertex of the graph into one of two categories:

1) Visited
2) Not Visited

- The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
- The algorithm works as follows:

1) Start by putting any one of the graph's vertices at the back of a queue.
2) Take the front item of the queue and add it to the visited list.
3) Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4) Keep repeating steps 2 and 3 until the queue is empty.

- The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

BFS Algorithm Applications

1. To build index by search index
2. For GPS navigation
3. Path finding algorithms
4. In Ford-Fulkerson algorithm to find maximum flow in a network
5. Cycle detection in an undirected graph
6. In minimum spanning tree

Algorithm: -

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Graph Traversal Approach	
Experiment No: 12	**Date:**	**Enrollment No: 92200133030**

Programming Language: - C++

Code :-

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

void bfs(vector<vector<int>>& adj, int s, vector<bool>& visited);

void bfsDisconnected(vector<vector<int>>& adj) {
    vector<bool> visited(adj.size(), false);

    for (int i = 0; i < adj.size(); ++i) {
        if (!visited[i]) {
            bfs(adj, i, visited);
        }
    }
}

void bfs(vector<vector<int>>& adj, int s,vector<bool>& visited) {
    queue<int> q;

    visited[s] = true;
    q.push(s);

    while (!q.empty()) {
        int curr = q.front();
        q.pop();
        cout << curr << " ";

        for (int x : adj[curr]) {
            if (!visited[x]) {
                visited[x] = true;
                q.push(x);
            }
        }
    }
}

void addEdge(vector<vector<int>>& adj, int u, int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Graph Traversal Approach	
Experiment No: 12	**Date:**	**Enrollment No: 92200133030**

```
int main(){
    int V = 6;
    vector<vector<int>> adj(V);
    vector<vector<int>> edges = { {1, 2}, {2, 0}, {0, 3}, {4, 5} };
    for (auto& e : edges)
        addEdge(adj, e[0], e[1]);
    cout << "Complete BFS of the graph:" << endl;
    bfsDisconnected(adj);
    return 0;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-A
nalysis-of-Algorithms\Lab - Manual\Experiment - 12\" ; if ($?) { g++ Breadth_First_Search.cpp -o Breadth_First_Search } ; if ($?) { .\Breadth_First_Search }
Complete BFS of the graph:
0 2 3 1 4 5
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Graph Traversal Approach	
Experiment No: 12	**Date:**	**Enrollment No: 92200133030**

2. Implement the Depth-First Search

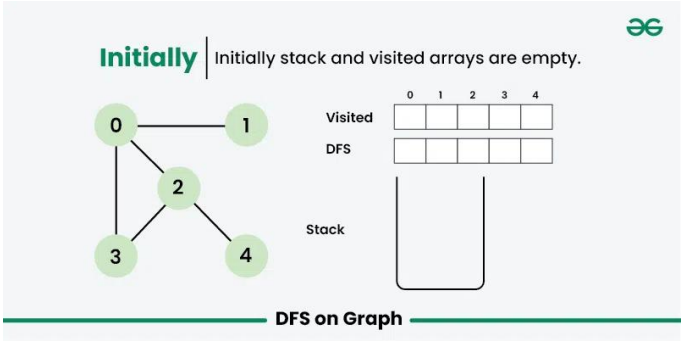
Theory: -

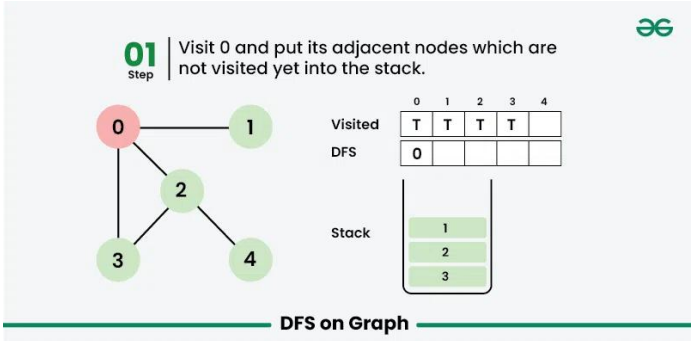
- Depth, First Search (or DFS) for a graph is similar to the Depth First Traversal of a tree. Like trees, we traverse all adjacent vertices one by one. When we traverse an adjacent vertex, we finish the traversal of all vertices reachable through that adjacent vertex. After we finish traversing one adjacent vertex and its reachable vertices, we move to the next adjacent vertex and repeat the process. This is similar to a tree, where we first completely traverse the left subtree and then move to the right subtree. The key difference is that, unlike trees, graphs may contain cycles (a node may be visited more than once). We use a boolean visited array to avoid processing a node multiple times.

Depth First Search Algorithm:-

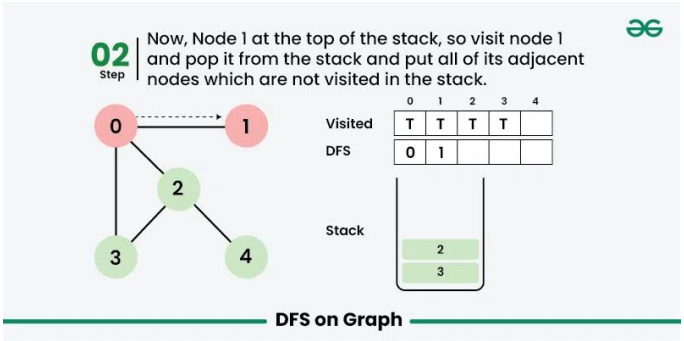
- A standard DFS implementation puts each vertex of the graph into one of two categories:
  1. Visited
  2. Not Visited
- The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
- The DFS algorithm works as follows:
  1. Start by putting any one of the graph's vertices on top of a stack.
  2. Take the top item of the stack and add it to the visited list.
  3. Create a list of that vertex's adjacent nodes. Add the ones that aren't in the visited list to the top of the stack.
  4. Keep repeating steps 2 and 3 until the stack is empty.

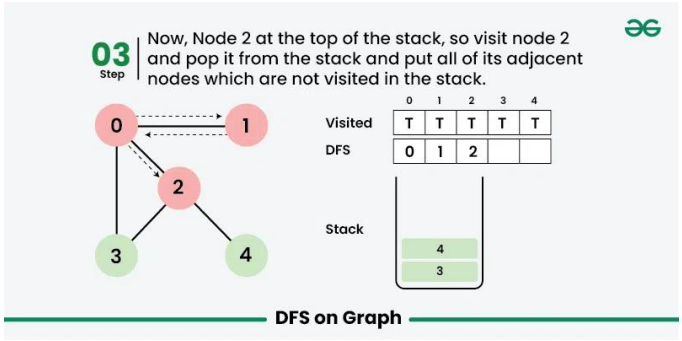
DFS from a Given Source of Undirected Graph:-

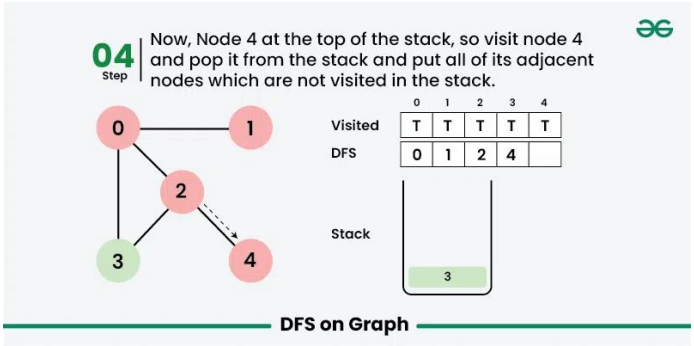


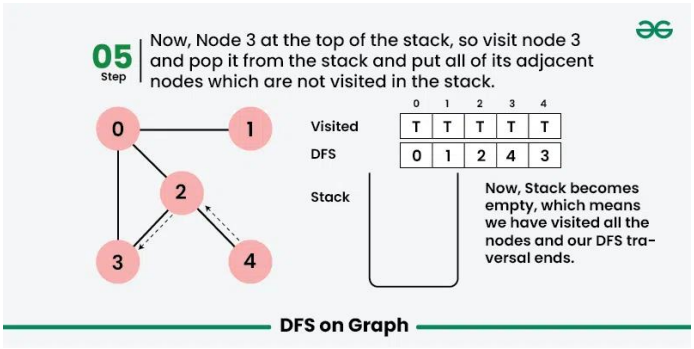


Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Graph Traversal Approach	
Experiment No: 12	**Date:**	**Enrollment No: 92200133030**









Algorithm: -

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Graph Traversal Approach	
Experiment No: 12	**Date:**	**Enrollment No: 92200133030**

Programming Language: - C++

Code :-

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

void addEdge(vector<vector<int>>& adj, int s, int t) {
    adj[s].push_back(t);
    adj[t].push_back(s);
}

void DFSRec(vector<vector<int>>& adj, vector<bool>& visited, int s) {
    visited[s] = true;
    cout << s << " ";

    for (int i : adj[s])
        if (visited[i] == false)
            DFSRec(adj, visited, i);
}

void DFS(vector<vector<int>>& adj) {
    vector<bool> visited(adj.size(), false);

    for (int i = 0; i < adj.size(); i++) {
        if (visited[i] == false) {
            DFSRec(adj, visited, i);
        }
    }
}

int main() {
    int V = 6;

    vector<vector<int>> adj(V);
    vector<vector<int>> edges = { {1, 2}, {2, 0}, {0, 3}, {4, 5} };

    for (auto& e : edges)
        addEdge(adj, e[0], e[1]);

    cout << "Complete DFS of the graph:" << endl;
    DFS(adj);

    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 Graph Traversal Approach	
Experiment No: 12	**Date:**	**Enrollment No: 92200133030**

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12> cd "d:\Aryan Data\Usefull Dat
a\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12\" ; if ($?) { g++ Depth_First_Search.cpp -o Depth_First_Se
arch } ; if ($?) { .\Depth_First_Search }
Complete DFS of the graph:
0 2 1 3 4 5
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12>
```

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Floyd Warshall All Pair Shortest Path Algorithm	
Experiment No: 13	**Date:**	**Enrollment No: 92200133030**

Aim: Implementing Floyd Warshall All Pair Shortest Path Algorithm

IDE: Visual Studio Code

Implementing Floyd Warshall All Pair Shortest Path Algorithm

Theory: -

- The Floyd-Warshall algorithm, named after its creators Robert Floyd and Stephen Warshall, is a fundamental algorithm in computer science and graph theory. It is used to find the shortest paths between all pairs of nodes in a weighted graph. This algorithm is highly efficient and can handle graphs with both positive and negative edge weights, making it a versatile tool for solving a wide range of network and connectivity problems.

Floyd Warshall Algorithm:

- The Floyd Warshall Algorithm is an all pair shortest path algorithm unlike Dijkstra and Bellman Ford which are single source shortest path algorithms. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative). It follows Dynamic Programming approach to check every possible path going via every possible node in order to calculate shortest distance between every pair of nodes.

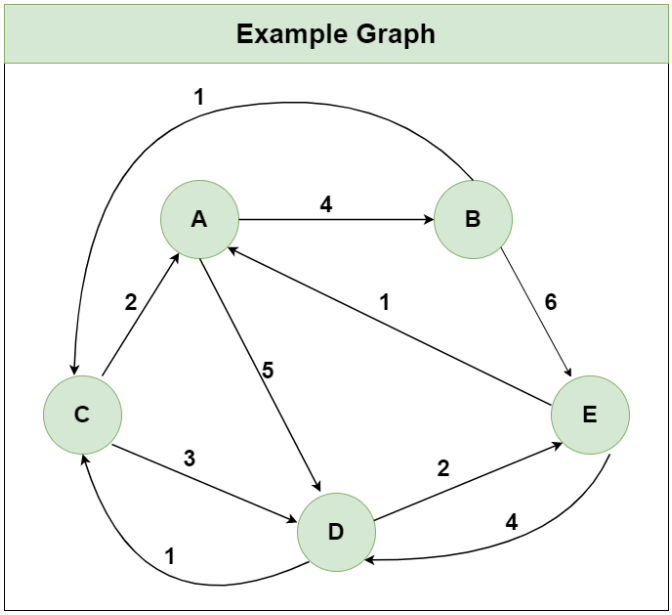
Idea Behind Floyd Warshall Algorithm:

- Initialize the solution matrix same as the input graph matrix as a first step.
- Then update the solution matrix by considering all vertices as an intermediate vertex.
- The idea is to pick all vertices one by one and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.
- When we pick vertex number **k** as an intermediate vertex, we already have considered vertices **{0, 1, 2, .. k-1}** as intermediate vertices.
- For every pair **(i, j)** of the source and destination vertices respectively, there are two possible cases.
    - **k** is not an intermediate vertex in shortest path from **i** to **j**. We keep the value of **dist[i][j]** as it is.
    - **k** is an intermediate vertex in shortest path from **i** to **j**. We update the value of **dist[i][j]** as **dist[i][k] + dist[k][j], if dist[i][j] > dist[i][k] + dist[k][j]**

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Floyd Warshall All Pair Shortest Path Algorithm	
Experiment No: 13	**Date:**	**Enrollment No: 92200133030**

Illustration of Floyd Warshall Algorithm :

Suppose we have a graph as shown in the image:



Step 1: Initialize the Distance[][] matrix using the input graph such that Distance[i][j]= weight of edge from i to j, also Distance[i][j] = Infinity if there is no edge from i to j.

Step1: Initializing Distance[ ][ ] using the Input Graph

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	∞	0	3	∞
D	∞	∞	1	0	2
E	1	∞	∞	4	0

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Floyd Warshall All Pair Shortest Path Algorithm	
Experiment No: 13	**Date:**	**Enrollment No: 92200133030**

Step 2: Treat node A as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:

= Distance[i][j] = minimum (Distance[i][j], (Distance from i to A) + (Distance from A to j))

= Distance[i][j] = minimum (Distance[i][j], Distance[i][A] + Distance[A][j])

Step 2: Using Node A as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][A] + Distance[A][j])

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	?	?	?	?
C	2	?	?	?	?
D	∞	?	?	?	?
E	1	?	?	?	?

→

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	6	0	3	12
D	∞	∞	1	0	2
E	1	5	∞	4	0

Step 3: Treat node B as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:

= Distance[i][j] = minimum (Distance[i][j], (Distance from i to B) + (Distance from B to j))

= Distance[i][j] = minimum (Distance[i][j], Distance[i][B] + Distance[B][j])

Step 3: Using Node B as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][B] + Distance[B][j])

	A	B	C	D	E
A	?	4	?	?	?
B	∞	0	1	∞	6
C	?	6	?	?	?
D	?	∞	?	?	?
E	?	5	?	?	?

	A	B	C	D	E
A	0	4	5	5	10
B	∞	0	1	∞	6
C	2	6	0	3	12
D	∞	∞	1	0	2
E	1	5	6	4	0

Step 4: Treat node C as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:

= Distance[i][j] = minimum (Distance[i][j], (Distance from i to C) + (Distance from C to j))

= Distance[i][j] = minimum (Distance[i][j], Distance[i][C] + Distance[C][j])

Step 4: Using Node C as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][C] + Distance[C][j])

	A	B	C	D	E
A	?	?	5	?	?
B	?	?	1	?	?
C	2	6	0	3	12
D	?	?	1	?	?
E	?	?	6	?	?

	A	B	C	D	E
A	0	4	5	5	10
B	3	0	1	4	6
C	2	6	0	3	12
D	3	7	1	0	2
E	1	5	6	4	0

Step 5: Treat node D as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:

= Distance[i][j] = minimum (Distance[i][j], (Distance from i to D) + (Distance from D to j))

= Distance[i][j] = minimum (Distance[i][j], Distance[i][D] + Distance[D][j])

Step 5: Using Node D as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][D] + Distance[D][j])

	A	B	C	D	E
A	?	?	?	5	?
B	?	?	?	4	?
C	?	?	?	3	?
D	3	7	1	0	2
E	?	?	?	4	?

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Step 6: Treat node E as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:

= Distance[i][j] = minimum (Distance[i][j], (Distance from i to E) + (Distance from E to j))
= Distance[i][j] = minimum (Distance[i][j], Distance[i][E] + Distance[E][j])

Step 6: Using Node E as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][E] + Distance[E][j])

	A	B	C	D	E
A	?	?	?	?	7
B	?	?	?	?	6
C	?	?	?	?	5
D	?	?	?	?	2
E	1	5	5	4	0

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Step 7: Since all the nodes have been treated as an intermediate node, we can now return the updated Distance[][] matrix as our answer matrix.

Step 7: Return Distance[ ][ ] matrix as the result

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Floyd Warshall All Pair Shortest Path Algorithm	
Experiment No: 13	**Date:**	**Enrollment No: 92200133030**

Algorithm: -

Programming Language: - C++

Code :-

```
 #include <bits/stdc++.h>
 using namespace std;
 #define V 5
 #define INF 99999

 void printSolution(int dist[][V]);

 void floydWarshall(int dist[][V]){
     int i, j, k;
     for (k = 0; k < V; k++) {
         for (i = 0; i < V; i++) {
             for (j = 0; j < V; j++) {
                 if (dist[i][j] > (dist[i][k] + dist[k][j]) && (dist[k][j] != INF &&
dist[i][k] != INF))
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Floyd Warshall All Pair Shortest Path Algorithm	
Experiment No: 13	**Date:**	**Enrollment No: 92200133030**

```
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }

        printSolution(dist);
    }

    void printSolution(int dist[][V]) {
        cout << "The following matrix shows the shortest distances between every pair of
vertices \n";
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][j] == INF)
                    cout << "INF" << " ";
                else
                    cout << dist[i][j] << "   ";
            }
            cout << endl;
        }
    }

    int main(){
        int graph[V][V] = { { 0, 4, INF, 5, INF },
                            { INF, 0, 1, INF, 6 },
                            { 2, INF, 0, 3, INF },
                            { INF,INF, 1,  0, 2 },
                            { 1, INF, INF, 4, 0 } };

        floydWarshall(graph);
        return 0;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 13> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design
-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 13\" ; if ($?) { g++ Flyod_Warshall_Algorithm.cpp -o Flyod_Warshall_Algorithm } ; if ($?) { .\Flyod_War
shall_Algorithm }
The following matrix shows the shortest distances between every pair of vertices
0   4   5   5   7
3   0   1   4   6
2   6   0   3   5
3   7   1   0   2
1   5   5   4   0
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 13>
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Floyd Warshall All Pair Shortest Path Algorithm	
Experiment No: 13	**Date:**	**Enrollment No: 92200133030**

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution

IDE: Visual Studio Code

Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution

- Given a partially filled 9×9 2D array 'grid[9][9]', the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

1. **Naive Approach:**

- The naive approach is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found, i.e. for every unassigned position fill the position with a number from 1 to 9. After filling all the unassigned positions check if the matrix is safe or not. If safe print else recurs for other cases.

- Follow the steps below to solve the problem:
  - Create a function that checks if the given matrix is valid sudoku or not. Keep Hashmap for the row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true;

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

- Create a recursive function that takes a grid and the current row and column index.
- Check some base cases.
  - If the index is at the end of the matrix, i.e. i=N-1 and j=N then check if the grid is safe or not, if safe print the grid and return true else return false.
  - The other base case is when the value of column is N, i.e j = N, then move to next row, i.e. i++ and j = 0.
- If the current index is not assigned then fill the element from 1 to 9 and recur for all 9 cases with the index of next element, i.e. i, j+1. if the recursive call returns true then break the loop and return true.
- If the current index is assigned then call the recursive function with the index of the next element, i.e. i, j+1

Algorithm: -

Programming Language: - C++

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

Code :-

```
#include <iostream>
using namespace std;
#define N 9

void print(int arr[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            cout << arr[i][j] << " ";
        cout << endl;
    }
}

bool isSafe(int grid[N][N], int row, int col, int num) {

    for (int x = 0; x <= 8; x++)
        if (grid[row][x] == num)
            return false;

    for (int x = 0; x <= 8; x++)
        if (grid[x][col] == num)
            return false;

    int startRow = row - row % 3, startCol = col - col % 3;

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (grid[i + startRow][j + startCol] == num)
                return false;

    return true;
}

bool solveSudoku(int grid[N][N], int row, int col) {
    if (row == N - 1 && col == N)
        return true;

    if (col == N) {
        row++;
        col = 0;
    }

    if (grid[row][col] > 0)
        return solveSudoku(grid, row, col + 1);

    for (int num = 1; num <= N; num++) {
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

```
            if (isSafe(grid, row, col, num)) {
                grid[row][col] = num;

                if (solveSudoku(grid, row, col + 1))
                    return true;
            }
            grid[row][col] = 0;
        }
        return false;
    }

    int main() {
        int grid[N][N] = { { 3, 0, 6, 5, 0, 8, 4, 0, 0 },
                           { 5, 2, 0, 0, 0, 0, 0, 0, 0 },
                           { 0, 8, 7, 0, 0, 0, 0, 3, 1 },
                           { 0, 0, 3, 0, 1, 0, 0, 8, 0 },
                           { 9, 0, 0, 8, 6, 3, 0, 0, 5 },
                           { 0, 5, 0, 0, 9, 0, 6, 0, 0 },
                           { 1, 3, 0, 0, 0, 0, 2, 5, 0 },
                           { 0, 0, 0, 0, 0, 0, 0, 7, 4 },
                           { 0, 0, 5, 2, 0, 6, 3, 0, 0 } };

        if (solveSudoku(grid, 0, 0))
            print(grid);
        else
            cout << "no solution  exists " << endl;
        return 0;
    }
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-A
nalysis-of-Algorithms\Lab - Manual\Experiment - 14\" ; if ($?) { g++ Sudoku_Solver_Naive.cpp -o Sudoku_Solver_Naive } ; if ($?) { .\Sudoku_Solver_Naive }
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14>
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

2. Sudoku Using Backtracking :-

- Like all other Backtracking problems, Sudoku can be solved by assigning numbers one by one to empty cells. Before assigning a number, check whether it is safe to assign.

- Check that the same number is not present in the current row, current column and current 3X3 subgrid. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn't lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.

- Follow the steps below to solve the problem:

- Create a function that checks after assigning the current index the grid becomes unsafe or not. Keep Hashmap for a row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true; hashMap can be avoided by using loops.

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

- Create a recursive function that takes a grid.
- Check for any unassigned location.
    - If present then assigns a number from 1 to 9.
    - Check if assigning the number to current index makes the grid unsafe or not.
    - If safe then recursively call the function for all safe cases from 0 to 9.
    - If any recursive call returns true, end the loop and return true. If no recursive call returns true then return false.
- If there is no unassigned location then return true.

Algorithm: -

Programming Language: - C++

Code :-

```
#include <bits/stdc++.h>
using namespace std;
#define UNASSIGNED 0
#define N 9
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

```
bool FindUnassignedLocation(int grid[N][N],int& row, int& col);
bool isSafe(int grid[N][N], int row,int col, int num);

bool SolveSudoku(int grid[N][N]) {
    int row, col;

    if (!FindUnassignedLocation(grid, row, col))
        return true;

    for (int num = 1; num <= 9; num++)
    {
        if (isSafe(grid, row, col, num))
        {
            grid[row][col] = num;
            if (SolveSudoku(grid))
                return true;
            grid[row][col] = UNASSIGNED;
        }
    }
    return false;
}

bool FindUnassignedLocation(int grid[N][N],int& row, int& col) {
    for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
            if (grid[row][col] == UNASSIGNED)
                return true;
    return false;
}

bool UsedInRow(int grid[N][N], int row, int num) {
    for (int col = 0; col < N; col++)
        if (grid[row][col] == num)
            return true;
    return false;
}

bool UsedInCol(int grid[N][N], int col, int num) {
    for (int row = 0; row < N; row++)
        if (grid[row][col] == num)
            return true;
    return false;
}

bool UsedInBox(int grid[N][N], int boxStartRow, int boxStartCol, int num) {
    for (int row = 0; row < 3; row++)
        for (int col = 0; col < 3; col++)
            if (grid[row + boxStartRow][col + boxStartCol] == num)
                return true;
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

```
    return false;
}

bool isSafe(int grid[N][N], int row, int col, int num) {
    return !UsedInRow(grid, row, num) && !UsedInCol(grid, col, num) && !UsedInBox(grid,
row - row % 3, col - col % 3, num) && grid[row][col] == UNASSIGNED;
}

void printGrid(int grid[N][N]) {
    for (int row = 0; row < N; row++)
    {
        for (int col = 0; col < N; col++)
            cout << grid[row][col] << " ";
        cout << endl;
    }
}

int main()
{
    int grid[N][N] = { { 3, 0, 6, 5, 0, 8, 4, 0, 0 },
                       { 5, 2, 0, 0, 0, 0, 0, 0, 0 },
                       { 0, 8, 7, 0, 0, 0, 0, 3, 1 },
                       { 0, 0, 3, 0, 1, 0, 0, 8, 0 },
                       { 9, 0, 0, 8, 6, 3, 0, 0, 5 },
                       { 0, 5, 0, 0, 9, 0, 6, 0, 0 },
                       { 1, 3, 0, 0, 0, 0, 2, 5, 0 },
                       { 0, 0, 0, 0, 0, 0, 0, 7, 4 },
                       { 0, 0, 5, 2, 0, 6, 3, 0, 0 } };
    if (SolveSudoku(grid) == true)
        printGrid(grid);
    else
        cout << "No solution exists";

    return 0;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14\" ; if ($?) { g++ Sudoku_Solver_Naive.cpp -o Sudoku_Solver_Naive } ; if ($?) { .\Sudoku_Solver_Naive }
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14>
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

Space Complexity:- _____
Justification: -

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

3. **Sudoku Using Bit Masks :-**

- This method is a slight optimization to the above 2 methods. For each row/column/box create a bitmask and for each element in the grid set the bit at position 'value' to 1 in the corresponding bitmasks, for O(1) checks.

- **Follow the steps below to solve the problem:**

- Create 3 arrays of size N (one for rows, columns, and boxes).
- The boxes are indexed from 0 to 8. (in order to find the box index of an element we use the following formula: row / 3 * 3 + column / 3).
- Map the initial values of the grid first.
- Each time we add/remove an element to/from the grid set the bit to 1/0 to the corresponding bitmasks.

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

Algorithm: -

Programming Language: - C++

Code :-

```
#include <bits/stdc++.h>
using namespace std;
#define N 9

int row[N], col[N], box[N];
bool seted = false;

int getBox(int i, int j) {
    return i / 3 * 3 + j / 3;
}

bool isSafe(int i, int j, int number) {
    return !((row[i] >> number) & 1) && !((col[j] >> number) & 1) && !((box[getBox(i, j)]
>> number) & 1);
}
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

```
void setInitialValues(int grid[N][N]) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            row[i] |= 1 << grid[i][j],
            col[j] |= 1 << grid[i][j],
            box[getBox(i, j)] |= 1 << grid[i][j];
}

bool SolveSudoku(int grid[N][N], int i, int j) {
    if (!seted)
        seted = true, setInitialValues(grid);

    if (i == N - 1 && j == N)
        return true;
    if (j == N)
        j = 0, i++;

    if (grid[i][j])
        return SolveSudoku(grid, i, j + 1);

    for (int nr = 1; nr <= N; nr++) {
        if (isSafe(i, j, nr)) {
            grid[i][j] = nr;
            row[i] |= 1 << nr;
            col[j] |= 1 << nr;
            box[getBox(i, j)] |= 1 << nr;

            if (SolveSudoku(grid, i, j + 1))
                return true;

            row[i] &= ~(1 << nr);
            col[j] &= ~(1 << nr);
            box[getBox(i, j)] &= ~(1 << nr);
        }

        grid[i][j] = 0;
    }

    return false;
}

void print(int grid[N][N])
{
    for (int i = 0; i < N; i++, cout << '\n')
        for (int j = 0; j < N; j++)
            cout << grid[i][j] << ' ';
}
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

```
int main()
{
    int grid[N][N] = { { 3, 0, 6, 5, 0, 8, 4, 0, 0 },
                       { 5, 2, 0, 0, 0, 0, 0, 0, 0 },
                       { 0, 8, 7, 0, 0, 0, 0, 3, 1 },
                       { 0, 0, 3, 0, 1, 0, 0, 8, 0 },
                       { 9, 0, 0, 8, 6, 3, 0, 0, 5 },
                       { 0, 5, 0, 0, 9, 0, 6, 0, 0 },
                       { 1, 3, 0, 0, 0, 0, 2, 5, 0 },
                       { 0, 0, 0, 0, 0, 0, 0, 7, 4 },
                       { 0, 0, 5, 2, 0, 6, 3, 0, 0 } };

    if (SolveSudoku(grid, 0, 0))
        print(grid);
    else
        cout << "No solution exists\n";

    return 0;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-A
nalysis-of-Algorithms\Lab - Manual\Experiment - 14\" ; if ($?) { g++ Sudoku_Solver_Bit_Masks.cpp -o Sudoku_Solver_Bit_Masks } ; if ($?) { .\Sudoku_Solver_Bit_Masks
 }
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14>
```

Space Complexity:- _____

Justification: -

_____

_____

_____

_____

_____

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	**Date:**	**Enrollment No: 92200133030**

Time Complexity:

Best Case Time Complexity: _____
Justification: -

Worst Case Time Complexity:- _____
Justification: -

Conclusion:-