L A B   E X P E R I M E N T

Paper Code: **BCS 204**

Paper Title: **Design and Analysis of Algorithm**

Faculty Name: Prof. Seeja K.R.

Name: MEGHAVI(18601012020)

Branch:

Year: 2nd

S.No.	List of Experiments	Remarks
	DIVIDE & CONQUER	
	Merge Sort	
	Quick Sort	
	Simultaneous Min_Max	
	Strassen's Matrix Multiplication for N*N Matrix	
	Write a Program to implement **Disjoint Set Data Structure.**	
	GREEDY STRATEGY	
	Job Sequencing with Deadline	
	Fractional Knapsack Problem	
	Huffman Coding Problem	
	Kruskal Algorithm	
	Prim's Algorithm	
	DYNAMIC PROGRAMMING	
	Matrix Chain Multiplication Problem	
	Longest Common Subsequence Problem	
	BACKTRACKING : N – Queen Problem	
	BRANCH & BOUND : 0/1 Knapsack	
	GRAPH ALGORITHMS	
	DFS and BFS	
	Topological Sort	

Experiment 1

MERGE SORT

DATE: 01-02-2022

AIM: *To Implement The Merge Sort Algorithm*.

ALGORITHM:

The Merge Sort function repeatedly divides the array into two halves until we reach a stage where wetry to perform Merge Sort on a sub array of size 1 .

After that, the merge function combines the sorted arrays into larger arrays until the wholearray is merged.
m = l+ (r-l)/2
Call mergeSort for 1rst half:

Call mergeSort(arr, l, m)
Find the middle point to divide the array into two halves:
middleCall mergeSort for second half:Call mergeSort(arr
sorted in step 2 and 3:Call merge(arr, l, m, r)
, m+1, r)
Merge the two halves

PROGRAM:

```
using namespace std;#include <iostream>
void merge(int a[], int beg, int mid, int
end)
{
int i, j, k;
int n1 = mid - beg + 1;int n2 = end - mid;
int LeftArray[n1], RightArray[n2];

for (int i = 0; i < n1; i++)LeftArray[i] = a[beg + i];for (int j = 0; j < n2; j++)
RightArray[j] = a[mid + 1 + j];

i = 0, /* initial index of 1rst sub-array */
j = 0; /* initial index of second sub-array */
k = beg; /* initial index of merged sub-array */

while (i < n1 && j < n2)
{
if(LeftArray[i] <= RightArray[j])
{
a[k] = LeftArray[i];i++;
}
else
{
a[k] = RightArray[j];j++;
} k++;
```

```
}
while (i<n1)

{
a[k] = LeftArray[i];i++;
k++;
}



while (j<n2)
{
a[k] = RightArray[j];j++;
k++;
}
}
void MERGE_SORT(int arr[], int beg, int end) {



if ( beg < end){
int mid = (beg + end)/2 ; MERGE_SORT(arr, beg, mid) ; MERGE_SORT(arr, mid + 1, end) ;merge
 (arr, beg, mid, end) ;
 }
 }
 int main()
 {
 int n; cin>>n; int arr[n];
 for(int i=0;i<n;i++){cin>>arr[i];
 }
 MERGE_SORT( arr,0,n);
 for(int i=0;i<n;i++){
 cout<<arr[i]<<" ";
 }
 cout<<endl;
 return 0;
 }
```

OUTPUT:

```
5
3
7
8
3
4
3 3 4 7 8


...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment 2

QUICK SORT

DATE: 11-02-2022

AIM: *To Implement The Quick Sort Algorithm.*

ALGORITHM:

quickSort(array, leftmostIndex, rightmostIndex)

if (leftmostIndex < rightmostIndex)
pivotIndex <- partition(array,leftmostIndex, rightmostIndex)
quickSort(array, leftmostIndex, pivotIndex - 1)
quickSort(array, pivotIndex, rightmostIndex)

partition(array, leftmostIndex, rightmostIndex)
set rightmostIndex as pivotIndex
storeIndex <- leftmostIndex - 1
for i <- leftmostIndex + 1 to rightmostIndex
if element[i] < pivotElement
swap element[i] and element[storeIndex]
storeIndex++
swap pivotElement and element[storeIndex+1]
return storeIndex + 1

CODE:

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

// A utility function to swap two elementsvoid
swap(int* a, int* b)
{
int t = *a;
*a = *b; *b = t;
}
int partition (int arr[], int low, int high)
{
int pivot = arr[high]; // pivot
int i = (low - 1); // Index of smaller element and indicates the right position of pivot
found so far

for (int j = low; j <= high - 1; j++)
{
// If current element is smaller than the pivotif
(arr[j] < pivot)
{
i++; // increment index of smaller element
swap(&arr[i], &arr[j]);



}
```

```
}

swap(&arr[i + 1],
&arr[high]);return (i + 1);

}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index
*/
void quickSort(int arr[], int low, int high)
{
if (low < high)
{
/* pi is partitioning index, arr[p] is now
at right place */
int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1,
high);
}
}

/* Function to print an array */
void printArray(int arr[], int size)
{
int i;
for (i = 0; i < size;
i++)cout << arr[i] <<
" "; cout << endl;
}

// Driver Code
int main()
{
int arr[] = {4,7,99,2,5,1};
int n = sizeof(arr) / sizeof(arr[0]);
quickSort(arr, 0, n - 1);
cout << "Sorted array: \n";
printArray(arr, n);
return 0;
}
```

OUTPUT:

```
for (int j = low; j <= high - 1; j
{
// If current element is smaller t
if (arr[j] < pivot)
{
i++; // increment index of smaller
swap(&arr[i], &arr[j]);
}
}
swap(&arr[i + 1], &arr[high]);
return (i + 1);

}
```

```
Sorted array:
1 2 4 5 7 99


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 3

STRASSEN'S MATRIX MULTIPLICATION

DATE: 18-02-2022

AIM: *Strassen's matrix multiplication for N*N Matrix.*

ALGORITHM:

Strassen's matrix multiplication algorithm is based on the divide and conquer principle.

- Divide the matrix into sub matrices
- Calculate the p1 to p7 values.
- Recombine the resultant values to form the answer

matrix. P1 = (a11+a22)*(b11+b22)

P2=

(a21+a22)*(b11)

P3= (a11)*(b12-

b22) P4=

(a22)*(b21-b11)

P5=

(a11+a12)*(b22)

P6= (a21-

a11)*(b11+b12) P7=

(a12+a22)*(b21+b22)

C12= p3+p5

C21= p2+p4

C11= p1+p4-p5+p7

C22= p1+p3-p2+p6

Time complexity:

O(n^{log7}) Space

complexity: O(n)

CODE:

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

std; double a[4][4];

```
double b[4][4];
void insert(double x[4][4])
{
   double val;
   for(int i=0;i<4;i++)
   {
      for(int j=0;j<4;j++)
      {
         cin>>val;
         x[i][j]=val
         ;
      }
   }
}
double cal11(double x[4][4])
{
   return (x[1][1] * x[1][2])+ (x[1][2] * x[2][1]);
}
double cal21(double x[4][4])
{
   return (x[3][1] * x[4][2])+ (x[3][2] * x[4][1]);
}
double cal12(double x[4][4])
{
   return (x[1][3] * x[2][4])+ (x[1][4] * x[2][3]);
}
double cal22(double x[4][4])
{
   return (x[2][3] * x[1][4])+ (x[2][4] * x[1][3]);
```

```
}
int main()
{
   double a11,a12,a22,a21,b11,b12,b21,b22,a[4][4],b[4][4];
   double p,q,r,s,t,u,v,c11,c12,c21,c22;
   //insert values in the matrix a
   cout<<"\n a: \n";
   insert(a);
   //insert values in the matrix a
   cout<<"\n b: \n";
   insert(b);
   //dividing single 4x4 matrix into four 2x2 matrices
   a11=cal11(a);
   a12=cal12(a
   );
   a21=cal21(a
   );
   a22=cal22(a
   );
   b11=cal11(b
   );
   b12=cal12(b
   );
   b21=cal21(b
   );
   b22=cal22(b
   );

   //assigning variables acc. to strassen's algo
   p=(a11+a22)*(b11+b22);
   q=(a21+a22)*b11;
   r=a11*(b12-b22);
```

```
s=a22*(b21-b11);
t=(a11+a12)*b22;
u=(a11-
a21)*(b11+b12);
```

v=(a12-a22)*(b21+b22);

//outputting the final matrix

cout<<"\n final matrix";

cout<<"\n"<<p+s-t+v<<"

"<<r+t; cout<<"\n"<<q+s<<"

"<<p+r-q+u; return 0;

}

OUTPUT:

```
 a:
4 5 6 8
8 7 5 1
3 7 5 9
21 4 56  9

 b:
9 8 7 6
6 7 8 3
7 8 9 1
23 6 8 9

 final matrix
8400 8968
57120 -119328
```

EXPERIMENT 4

SIMULTANEOUS MIN MAX

DATE: 23-02-2022

AIM: *Simultaneous min max*

ALGORITHM:

a. Let P = (n, a [i],……,a [j]) denote an arbitrary instance of the problem.

b. Here ‘n’ is the no. of elements in the list (a [i],….,a[j]) and we are interested in finding the maximum and minimum of the list.

c. If the list has more than 2 elements, P has to be divided into smaller instances.

d. For example, we might divide ‘P’ into the 2 instances, P1=([n/2],a[1], a[n/2]) & P2= (n-
[n/2], a[[n/2]+1],….., a[n]) After having divided ‘P’ into 2 smaller sub problems, we can solve them by recursively invoking the same divide-and-conquer algorithm

CODE:

```
#include<iostream>

using namespace std;

// Pair struct is used to return

// two values from getMinMax()

struct Pair

{

	int min;

	int max;

};

Pair getMinMax(int arr[], int n)

{
```

```
struct Pair
minmax; int i;
// If there is only one element
// then return it as min and max
both if (n == 1)
{
        minmax.max      =
        arr[0];   minmax.min
        =     arr[0];     return
        minmax;
}
// If there are more than one elements,
// then initialize min and
max if (arr[0] > arr[1])
{minmax.max = arr[0];
minmax.min = arr[1];
}
else
{
minmax.max = arr[1];
minmax.min = arr[0];
}
for(i = 2; i < n; i++)
{
        if (arr[i] > minmax.max)
```

```
                minmax.max = arr[i];

                else if (arr[i] <

                minmax.min)

                minmax.min = arr[i];

        }

        return minmax;

}

// Driver code

int main()

{

        int arr[] = { 1000, 11, 445,

                        1, 330, 3000 };

int arr_size = 6;

        struct Pair minmax = getMinMax(arr, arr_size);

        cout << "Minimum element is "

                << minmax.min <<

        endl; cout << "Maximum

        element is "

                <<

        minmax.max; return

        0;

}
```

OUTPUT:

```
Minimum element is 6
Maximum element is 9000
```

Experiment-6

Date: 07-02-2022

Aim:

To implement the Shell sort algorithm.

Algorithm:

1. ShellSort(a, n) // 'a' is the given array, 'n' is the size of array
2. for (interval = n/2; interval > 0; interval /= 2)
3. for (i = interval; i < n; i += 1)
4. temp = a[i];
5. for (j = i; j >= interval && a[j - interval] > temp; j -= interval)
6. a[j] = a[j - interval];
7. a[j] = temp;
8. End ShellSort

Efficiency of Program

Time Complexity:

Best Case : O(n*logn)

Average Case : O(n*log(n)^2)

Worst Case : O(n^2).

Space Complexity: The space complexity of heap sort is O(1).

Program:

```
#include<iostream>
using namespace std;
// A function implementing Shell sort.
void ShellSort(int a[], int n)
{
	int i, j, k, temp;
// Gap 'i' between index of the element to be compared, initially n/2.
	for(i = n/2; i > 0; i = i/2)
	{
```

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

			for(k = j-i; k >= 0; k = k-i)
			{
			// If value at higher index is greater, then break the loop.
				if(a[k+i] >= a[k])
				break;
				// Switch the values otherwise.
				else
				{
					temp = a[k];
					a[k] = a[k+i];
					a[k+i] = temp;
				}
			}
		}
	}
}
int main()
{
	int n, i;
	cout<<"\nEnter the number of data element to be sorted: ";
	cin>>n;

	int arr[n];
	for(i = 0; i < n; i++)
	{
		cout<<"Enter element "<<i+1<<": ";
```

```
cin>>arr[i];
	}
	ShellSort(arr, n);

	// Printing the sorted data.
	cout<<"\nSorted Data ";
	for (i = 0; i < n; i++)
		cout<<"->"<<arr[i];

	return 0;
}
```

<u>Output:</u>

```
Enter the number of data element to be sorted: 6
Enter element 1: 4
Enter element 2: 5
Enter element 3: 2
Enter element 4: 1
Enter element 5: 7
Enter element 6: 9

Sorted Data ->1->2->4->5->7->9

...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment-5

Date: 28-02-2022

Aim:

To implement the heap sort algorithm.

Algorithm:

HeapSort(arr)

Step 1: for i = n to 2
Step 2: swap arr[1] with arr[i]
Step 3: heap_size[arr] = heap_size[arr] ? 1
Step 4: MaxHeapify(arr,1)
Step 5: End

BuildMaxHeap(arr)

Step 1: BuildMaxHeap(arr)
Step 2: heap_size(arr) = n
Step 3: for i = n/2 to 1
Step 4: MaxHeapify(arr,i)
Step 5: End

MaxHeapify

```
Step 1: temp = 2*i
      j=2*i
Step 2: while ( j<=n)
    { if ( j<n && a[j+1] > a[j]
       j=j+1
    If(temp > a[j])
      Break
    Else if ( temp <= a[j])
   { a[j/2] = a[j])
        j= 2*j
   }
   }
Step 3: a[j/2] = temp
Step 4: End
```

Efficiency of Program

Time Complexity: The time complexity of heap sort is O(n log n).

Space Complexity: The space complexity of heap sort is O(1).

Program:

```
#include <iostream>
using namespace std;
// A function to heapify the array.
void MaxHeapify(int a[], int i, int n)
{
	int j, temp;
	temp = a[i];
	j = 2*i;

	while (j <= n)
	{
		if (j < n && a[j+1] > a[j])
		j = j+1;
		// Break if parent value is already greater than child value.
		if (temp > a[j])
			break;
		// Switching value with the parent node if temp < a[j].
		else if (temp <= a[j])
		{
			a[j/2] = a[j];
			j = 2*j;
		}
	}
	a[j/2] = temp;
	return;
}
void HeapSort(int a[], int n)
{
```

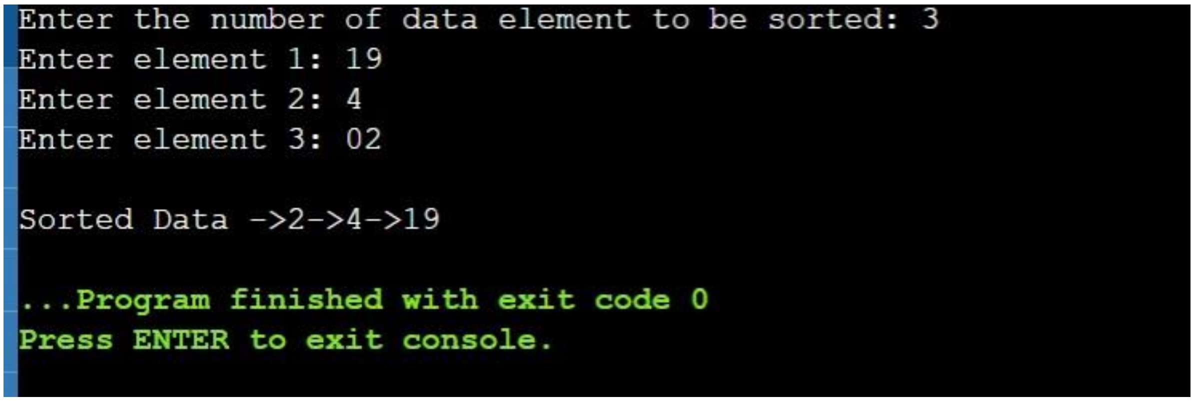
```
		int i, temp;
		for (i = n; i >= 2; i--)
		{
			// Storing maximum value at the end.
			temp = a[i];
			a[i] = a[1];
			a[1] = temp;
			// Building max heap of remaining element.
			MaxHeapify(a, 1, i - 1);
		}
	}
	void Build_MaxHeap(int a[], int n)
	{
		int i;
		for(i = n/2; i >= 1; i--)
			MaxHeapify(a, i, n);
	}
	int main()
	{
		int n, i;
		cout<<"\nEnter the number of data element to be sorted: ";
		cin>>n;
		n++;
		int arr[n];
		for(i = 1; i < n; i++)
		{
			cout<<"Enter element "<<i<<": ";
			cin>>arr[i];
		}
		// Building max heap.
		Build_MaxHeap(arr, n-1);
		HeapSort(arr, n-1);
```

```
        // Printing the sorted data.
        cout<<"\nSorted Data ";

        for (i = 1; i < n; i++)
                cout<<"->"<<arr[i];

        return 0;
}
```

Output:



```
Enter the number of data element to be sorted: 3
Enter element 1: 19
Enter element 2: 4
Enter element 3: 02

Sorted Data ->2->4->19

...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment-7

Date: 14-03-2022

Aim:

To implement the Job Sequencing with Deadline.

Algorithm:

Step 1: We have to sort the jobs in descending order of profit.

Step 2: Next, we have to iterate through the job and select slots.

Step 3: Slot i is selected if,

a. Slot i is not previously selected.

b. i < deadline

c. i should be just less than the deadline if possible (so that other slots can be optimized).

Step 4: If no such slot is possible, ignore the job.

Step 5: Exit.

Efficiency of Program

Time complexity**:** O(nlog(n))
Space complexity**:** O(n)

Program:

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

//structure for holding values
typedef struct Job
{
   int jobNum;
   int deadline;
   int profit;
}Job;

bool compare(Job a, Job b);
```

```
void jobSequencing(Job input[], int num);

int main()
{
    int num;
    cin >> num;
    Job input[num];
    // inputing the values
    for (int i = 0; i < num; i++)
    {
        cin >> input[i].jobNum;
        cin >> input[i].deadline;
        cin >> input[i].profit;
    }

    jobSequencing(input, num);
}

// a custom comparison function for arrenging jobs in decreasing order of profit
bool compare(Job a, Job b)
{
    return (a.profit > b.profit);
}

// main part of code where job sequencing happens
void jobSequencing(Job input[], int num)
{
    sort(input, input + num, compare);

    int result[num];
     bool slot[num];
    // setting all values in slot as false
```

```
    memset(slot, 0, sizeof(slot));

    for (int i = 0; i < num; i++)
    {
        for (int j = min(num, input[i].deadline)-1; j >= 0; j--)
        {
            if(slot[j] == false)
            {
                result[j] = i;
                slot[j] = true;
                break;
            }
        }
    }

    cout << "Job sequenced in order: ";
    for (int i=0; i<num; i++)
    {
      if (slot[i] == true)
       cout << input[result[i]].jobNum << " ";
    }
}
```

Output:

```
4
1 5 9
2 1 7
3 1 8
6 1 8
Job sequenced in order: 3 1

...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment-8

Date: 14-03-2022

Aim:

To implement the Fractional Knapsack Problem.

Algorithm:

Step 1: Start

Step 2: Take an array of structure Item

Step 3: Declare value, weight, knapsack weight and density

Step 4: Calculate density=value/weight for each item

Step 5: Sorting the items array on the order of decreasing density

Step 6: We add values from the top of the array to total value until the bag is full, i.e; total value <= W

Step 7: End

Efficiency of Program

Time complexity**:** O(nlog(n))
Space complexity**:** O(1)

Program:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
typedef struct {
   int    v;
   int   w;
   float d;
} Item;
void input(Item items[],int sizeOfItems) {
   cout << "Enter total "<< sizeOfItems <<" item's values and weight"<<
   endl;
   for(int i = 0; i < sizeOfItems; i++) {
```

```
        cout << "Enter "<< i+1 << " V ";
        cin >> items[i].v;
        cout << "Enter "<< i+1 << " W ";
        cin >> items[i].w;
    }
}
void display(Item items[], int sizeOfItems) {
    int i;
    cout << "values: ";
    for(i = 0; i < sizeOfItems; i++) {
        cout << items[i].v << "\t";
    }
    cout << endl << "weight: ";
    for (i = 0; i < sizeOfItems; i++) {
        cout << items[i].w << "\t";
    }
    cout << endl;
}
bool compare(Item i1, Item i2) {
    return (i1.d > i2.d);
}
float knapsack(Item items[], int sizeOfItems, int W) {
    int i, j;
    float totalValue = 0, totalWeight = 0;
    for (i = 0; i < sizeOfItems; i++) {
        items[i].d = (float)items[i].v / items[i].w;
    }
    sort(items, items+sizeOfItems, compare);

    cout << "values : ";
```

```
  for(i = 0; i < sizeOfItems; i++) {
    cout << items[i].v << "\t";
  }
  cout << endl << "weights: ";
  for (i = 0; i < sizeOfItems; i++) {
    cout << items[i].w << "\t";
  }
  cout << endl << "ratio : ";
  for (i = 0; i < sizeOfItems; i++) {
    cout << items[i].d << "\t";
  }
  cout << endl;

  for(i=0; i<sizeOfItems; i++) {
    if(totalWeight + items[i].w<= W) {
      totalValue += items[i].v ;
      totalWeight += items[i].w;
    } else {
      int wt = W-totalWeight;
      totalValue += (wt * items[i].d);
      totalWeight += wt;
      break;
    }
  }
  cout << "Total weight in bag " << totalWeight<<endl;
  return totalValue;
}
int main() {
  int W;
  Item items[4];
```

```
    input(items, 4);
    cout << "Entered data \n";
    display(items,4);
    cout<< "Enter Knapsack weight \n";
    cin >> W;
    float mxVal = knapsack(items, 4, W);
    cout << "Max value for "<< W <<" weight is "<< mxVal;
}
```

Output:

```
Enter total 4 item's values and weight
Enter 1 V 7
Enter 1 W 6
Enter 2 V 5
Enter 2 W 8
Enter 3 V 9
Enter 3 W 3
Enter 4 V 9
Enter 4 W 1
Entered data
values: 7       5       9       9
weight: 6       8       3       1
Enter Knapsack weight
9
values : 9      9       7       5
weights: 1      3       6       8
ratio : 9       3       1.16667 0.625
Total weight in bag 9
Max value for 9 weight is 23.8333

...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment-9

Date: 04-04-2022

Aim:

To implement the Huffman coding algorithm.

Algorithm:

Step 1: Create a node for each alphabet.

Step 2: Sort them in ascending order of their frequencies.

Step 3: Merge two nodes with the least frequency.

Step 4: The parent node's value will be the sum of values from both the Nodes

Step 5: We keep repeating the third and fourth step until we obtain the binary tree.

Step 6: The tree obtained after merging all the nodes.

Step 7: Let us now obtain the encoding for all the alphabets

- Add a 0 to the representation every time you turn left
- Add a 1 to the representation every time you turn right

Step 8: Exit

Efficiency of Program

Time complexity: O(nlog(n))
Space complexity: O(1)

Program:

```
#include <iostream>
using namespace std;
#define MAX_TREE_HT 100
struct MinHeapNode
{
    char data;
    int freq;
```

```
    struct MinHeapNode *left, *right;
};

struct MinHeap
{
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
};

struct MinHeapNode* newNode(char data, unsigned freq)
{
    struct MinHeapNode* temp = (struct MinHeapNode*)malloc(sizeof(struct
MinHeapNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}
struct MinHeap* createMinHeap(unsigned capacity)
{
    struct MinHeap* minHeap = (struct MinHeap*)malloc(sizeof(struct
MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
minHeap->array=(struct MinHeapNode**)malloc(minHeap-> capacity *
sizeof(struct MinHeapNode*));
    return minHeap;
}
void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b)
```

```
{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}
void minHeapify(struct MinHeap* minHeap, int idx)
{
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
    if (left < minHeap->size && minHeap->array[left]-> freq<minHeap-
>array[smallest]->freq)
        smallest = left;
    if (right < minHeap->size && minHeap->array[right]->freq<minHeap-
>array[smallest]->freq)
        smallest = right;

    if (smallest != idx)
    {
        swapMinHeapNode(&minHeap->array[smallest],
                        &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}
int isSizeOne(struct MinHeap* minHeap)
{
    return (minHeap->size == 1);
}
struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
```

```
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}
void insertMinHeap(struct MinHeap* minHeap, struct MinHeapNode*
minHeapNode)
{

    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && minHeapNode->freq<minHeap->array[(i - 1) / 2]->freq)
    {
        minHeap->array[i] = minHeap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }
    minHeap->array[i] = minHeapNode;
}

void buildMinHeap(struct MinHeap* minHeap)
{
    int n = minHeap->size - 1;
    int i;
    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

void printArr(int arr[], int n)
{
```

```
    int i;
    for (i = 0; i < n; ++i)
        cout<< arr[i];

    cout<<"\n";
}
int isLeaf(struct MinHeapNode* root)

{
    return !(root->left) && !(root->right);
}



struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size)
{
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);

    minHeap->size = size;
    buildMinHeap(minHeap);

    return minHeap;
}
struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size)

{
    struct MinHeapNode *left, *right, *top;
    struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);
    while (!isSizeOne(minHeap))
```

```
    {
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = newNode('$', left->freq + right->freq);

        top->left = left;
        top->right = right;

        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);
}
void printCodes(struct MinHeapNode* root, int arr[], int top)
{

    // Assign 0 to left edge and recur
    if (root->left) {

        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }

    // Assign 1 to right edge and recur
    if (root->right) {

        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }
    if (isLeaf(root)) {
```

```
        cout<< root->data <<": ";
        printArr(arr, top);
    }
}
void HuffmanCodes(char data[], int freq[], int size)

{
    // Construct Huffman Tree
    struct MinHeapNode* root
        = buildHuffmanTree(data, freq, size);
    int arr[MAX_TREE_HT], top = 0;
    printCodes(root, arr, top);
}
int main()
{ int n;
cout<<"Enter the no. of elements";
cin>>n;
  cout<<"Enter characters";
    char arr[n] ;

    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
     cout<<"Enter frequencies";
    int freq[n];

     for(int i=0;i<n;i++){
        cin>>freq[i];
    }
```

```
    int size = sizeof(arr) / sizeof(arr[0]);

    HuffmanCodes(arr, freq, size);

    return 0;
}
```

Output:

```
Enter the no. of elements4
Enter charactersq r s t
Enter frequencies 2 5 6 7
t: 0
s: 10
q: 110
r: 111


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 10

Date: 04/04/2022

AIM: To implement the Kruskal's minimum spanning tree algorithm.

ALGORITHM:
MST- KRUSKAL (G, w)

1. A ← ∅
2. for each vertex v ∈ V [G]
3. do MAKE - SET (v)
4. sort the edges of E into non decreasing order by weight w
5. for each edge (u, v) ∈ E, taken in non decreasing order byweight
6. do if FIND-SET (μ) ≠ if FIND-SET (v)
7. then A ← A ∪ {(u, v)}
8. UNION (u, v)
9. return A

PROGRAM:

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

class Graph
{
  char vertices[10][10];
int cost[10][10],no;
public:
  Graph();
  void creat_graph();
  void display();
  int Position(char[]);
  void kruskal_algo();
};

/* Initialzing adj matrix with 999 */
/* 999 denotes infinite distance */
Graph::Graph()
{
  no=0;
  for(int i=0;i<10;i++)
  for(int j=0;j<10;j++)
  {
     cost[i][j]=999;
   }
}
```

```
/* Taking inputs for creating graph */
void Graph::creat_graph()
{
  char ans,Start[10],End[10];
  int wt,i,j;
  cout<<"Enter the number of vertices: ";
  cin>>no;
  cout<<"\nEnter the vertices: ";
  for(i=0;i<no;i++)
        cin>>vertices[i];
  do
  {
    cout<<"\nEnter Start and End vertex of the edge: ";
    cin>>Start>>End;
    cout<<"Enter weight: ";
    cin>>wt;
    i=Position(Start);
    j=Position(End);
    cost[i][j]=cost[j][i]=wt;
    cout<<"\nDo you want to add more edges (Y=YES/N=NO)? : "; /* Type 'Y' or 'y' for YES
and 'N' or 'n' for NO */
    cin>>ans;
  }while(ans=='y' || ans=='Y');
}

/* Displaying Cost matrix */
void Graph::display()
{
  int i,j;
  cout<<"\n\nCost matrix: ";
  for(i=0;i<no;i++)
  {
     cout<<"\n";
     for(j=0;j<no;j++)
     cout<<"\t"<<cost[i][j];
  }
}

/* Retrieving position of vertices in 'vertices' array */
int Graph::Position(char key[10])
{
  int i;
  for(i=0;i<10;i++)
  if(strcmp(vertices[i],key)==0)
    return i;
return -1;
}

void Graph::kruskal_algo()
```

```
{
 int i,j,v[10]={0},x,y,Total_cost=0,min,gr=1,flag=0,temp,d;

 while(flag==0)
 {
   min=999;
    for(i=0;i<no;i++)
   {
     for(j=0;j<no;j++)
        {
       if(cost[i][j]<min)
           {
         min=cost[i][j];
         x=i;
         y=j;
           }
        }
    }

   if(v[x]==0 && v[y]==0)
   {
     v[x]=v[y]=gr;
     gr++;
   }
   else if(v[x]!=0 && v[y]==0)
     v[y]=v[x];
   else if(v[x]==0 && v[y]!=0)
     v[x]=v[y];
   else
   {
     if(v[x]!=v[y])
     {
       d=v[x];
       for(i=0;i<no;i++)
       {
         if(v[i]==d)
         v[i]=v[y];
       }//end for
     }
   }

   cost[x][y]=cost[y][x]=999;
   Total_cost=Total_cost+min;      /* calculating cost of minimum spanning tree */
   cout<<"\n\t"<<vertices[x]<<"\t\t"<<vertices[y]<<"\t\t"<<min;

      temp=v[0]; flag=1;
      for(i=0;i<no;i++)
      {
        if(temp!=v[i])
        {
```

```
          flag=0;
          break;
          }
        }
   }
 cout<<"\nTotal cost of the tree= "<<Total_cost;
 }

 int main()
 {
   Graph g;
   g.creat_graph();
   g.display();

   cout<<"\n\n\nMinimum Spanning tree using kruskal algo=>";
   cout<<"\nSource vertex\tDestination vertex\tWeight\n";
   g.kruskal_algo();

 return 0;
 }
```

OUTPUT:

```
Enter the number of vertices: 4

Enter the vertices: 0 2 5 7

Enter Start and End vertex of the edge: 3 4
Enter weight: 6

Do you want to add more edges (Y=YES/N=NO)? : y

Enter Start and End vertex of the edge: 7 8
Enter weight: 7

Do you want to add more edges (Y=YES/N=NO)? : y

Enter Start and End vertex of the edge: 7 4
Enter weight: 8

Do you want to add more edges (Y=YES/N=NO)? : n


Cost matrix:
        999     999     999     999
        999     999     999     999
        999     999     999     999
        999     999     999     999



Minimum Spanning tree using kruskal algo=>
Source vertex   Destination vertex      Weight


...Program finished with exit code 0
Press ENTER to exit console.
```

Complexity

- **Time complexity: O(E log V)**
- **Space complexity: O(E + V)**

EXPERIMENT 11

Date: 11/04/2022

AIM: To implement the Prim's Minimum spanning Tree algorithm.

ALGORITHM:

i. Choose the edge with the smallest weight among all the active edges of any source.
ii. We need to select the vertex in MST.
iii. Add the edges starting with the previous vertex in the active edge list.
iv. Then we repeat the second step again and again till we have all the vertices in our graph.

PROGRAM:

```
#include <bits/stdc++.h>
using namespace std;
class Graph{
    vector<pair<int,int>> *l;
    int V;
    public:
    Graph(int n){
    V = n;
    l = new vector<pair<int,int>> [n];
    }
    void addEdge(int x,int y,int w){
    l[x].push_back({y,w});
    l[y].push_back({x,w});
    }
    int prim_mst(){
    priority_queue<pair<int,int>, vector<pair<int,int> > ,greater<pair<int,int>>>
    Q;
```

```
        bool *visited = new bool[V]{0};
        int ans = 0;
        Q.push({0,0});
        while(!Q.empty()){
                auto best = Q.top();
                Q.pop();
                int to = best.second;
                int weight = best.first;
                if(visited[to]){
                continue;
                }
                ans += weight;
                visited[to] = 1;
                for(auto x:l[to]){
                if(visited[x.first] == 0){
                        Q.push({x.second,x.first});
                }
                }
        }
        return ans;
        }
};
int main()
{
        int n,m;
        cin>>n>>m;
        Graph g(n);
        for(int i = 0;i<m;i++){
        int x,y,w;
```

```
        cin>>x>>y>>w;
        g.addEdge(x-1,y-1,w);
        }
        cout<<g.prim_mst()<<"\n";
        return 0;
}
```

OUTPUT:

```
7 8
4 6 1
5 7 1
2 7 1
8 2 1


...Program finished with exit code 0
Press ENTER to exit console.
```

Efficiency:

For Adjacency List:

Time Complexity: O(ElogV), where E is the number of edges and V is the number ofvertices.

For Adjacency Matrix:

Time Complexity: O(V^2), where V is the number of vertices.

Experiment-12

Date: 18-04-2022

AIM: To implement the Floyd Warshall's All pair shortest path algorithm.

ALGORITHM:

let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)

for each vertex v

dist[v][v] ← 0

for each edge (u,v)

dist[u][v] ← w(u,v) // the weight of the edge (u,v)

for k from 1 to |V|

for i from 1 to |V|

for j from 1 to |V|

if dist[i][j] > dist[i][k] + dist[k][j]

dist[i][j] ← dist[i][k] + dist[k][j]

end if

Efficiency of program:

Time Complexity: O(V^3)

Space Complexity: O(V^2)

PROGRAM:

```
#include   <iostream>
using namespace std;
void floyds(int b[][7])
{
 int i, j, k;
 for (k = 0; k < 7; k++)
 {
 for (i = 0; i < 7; i++)
 {
 for (j = 0; j < 7; j++)
```

```
{
if ((b[i][k] * b[k][j] != 0) && (i != j))
{
if ((b[i][k] + b[k][j] < b[i][j]) || (b[i][j] == 0))
{
b[i][j] = b[i][k] + b[k][j];
}
}
}
}
}
for (i = 0; i < 7; i++)
{
cout<<"\nMinimum Cost With Respect to Node:"<<i<<endl;
for (j = 0; j < 7; j++)
{
cout<<b[i][j]<<" ";
}
}
}
int main()
{
int b[7][7];
cout<<"ENTER VALUES OF ADJACENCY MATRIX\n";
for (int i = 0; i < 7; i++)
{
cout<<"enter values for "<<(i+1)<<" row"<<endl;
for (int j = 0; j < 7; j++)
cin>>b[i][j];
}
```

```
floyds(b);
return 0;
}
```

OUTPUT:

```
ENTER VALUES OF ADJACENCY MATRIX
enter values for 1 row
1 2 3 4 5 6 7
enter values for 2 row
2 3 4 5 6 7 8
enter values for 3 row
4 5 6 7 8 9 0
enter values for 4 row
1 3 5 7 9 0 1
enter values for 5 row
1 3 2 4 8 9 5
enter values for 6 row
4 6 2 7 7 9 4
enter values for 7 row
2 3 4 5 6 7 8

Minimum Cost With Respect to Node:0
1 2 3 4 5 6 5
Minimum Cost With Respect to Node:1
2 3 4 5 6 7 6
Minimum Cost With Respect to Node:2
4 5 6 7 8 9 8
Minimum Cost With Respect to Node:3
1 3 4 7 6 7 1
Minimum Cost With Respect to Node:4
1 3 2 4 8 7 5
Minimum Cost With Respect to Node:5
4 6 2 7 7 9 4
Minimum Cost With Respect to Node:6
2 3 4 5 6 7 8

...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 13

Date: 18/04/2022

AIM: To implement the Matrix chain Multiplication algorithm.

ALGORITHM:

Step:1 Create a dp matrix and set all values with a big value(INFINITY).
Step:2 for i in range 1 to N-1:
dp[i][i]=0.
Step:3 for i in range 2 to N-1:
for j in range 1 to N-i+1:
ran=i+j-1.
for k in range i to j:
dp[j][ran]=min(dp[j][ran],dp[j][k]+dp[k+1][ran]+v[j-1]*v[k]*v[ran]).
Step:4 Print dp[1][N-1].

PROGRAM:

```
#include<bits/stdc++.h>
using namespace std;
#define INF 1000000009
int min_operation(vector<int> &v, int n)
{
int dp[n+1][n+1];
memset(dp,INF,sizeof(dp));

for(int i=1;i<n;i++)
{
dp[i][i]=0;
}
/*Find M[i,j] using the formula.*/
int ran;
for(int i=2;i<n;i++)
{
for(int j=1;j<n-i+1;j++)
{
ran=i+j-1;
for(int k=j;k<=ran-1;k++)
{
/*formula used here.*/
dp[j][ran]=min(dp[j][ran],dp[j][k]+dp[k+1][ran]+v[j-1]*v[k]*v[ran]);
```

```
}
}
}

return dp[1][n-1];
}
int main()
{
int n;

cin>>n;
/*sequence/chain of the matrices if there are n matrices then chain contain n+1
numbers.*/
vector<int> chain;
for(int i=0;i<n+1;i++)
{
int x;
cin>>x;
chain.push_back(x);
}
/*store the min operation needed to multiply all the given matrices in ans.*/
int ans=min_operation(chain,n+1);

cout<<ans<<endl;
return 0;
}
```

OUTPUT:

```
5
20 15 30 45 24 10
31800
```

Efficiency:

Time Complexity: O(N^3)
Space Complexity: O(N^2)
(where N is the number present in the chain of the matrices)