Expt. No. _____ Page No. _____

Assignment

1 Sorting Techniques

1.1 Implement Quick Sort using first/last/any random element as pivot.

Ans:-

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#define MAX 100
int size[20], x=0, n, s=0;
float x2[20];
using namespace std;
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);
    for(int j=low; j<=high-1; j++)
    {
        if (arr[j] <= pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i+1], &arr[high]);
    return (i+1);
}
```

Teacher's Signature _____

Expt. No. _____ Page No. _____

```
void quicksort ( int arr[ ], int low, int high)
{
    if (low < high)
    {
        int pi = partition (arr, low, high);
        quicksort (arr, low, pi-1);
        quicksort (arr, pi+1, high);
    }
    int end_clock = clock ();
    size[s] = n;
    s++;
}
void printArray (int arr[], int size )
{   int i;
    for ( i=0; i< size; i++)
        cout << " " << arr[i];
    cout << endl;
}
int main()
{
    int a[1000],n=3;
    while (n>0)
    {   cout<<"\n Enter the number of elements in the array:";
        cin >> n;
        int a[n];
        for (int i=0; i<n; i++)
        {   int random = rand()%100;
            a[i] = random;
        }
        cout << " Randomly generated array: ";
        int start_clock = clock();
        quicksort (a,0,n-1);
        cout <<" Sorted array: ";
        printArray (a, n);
```

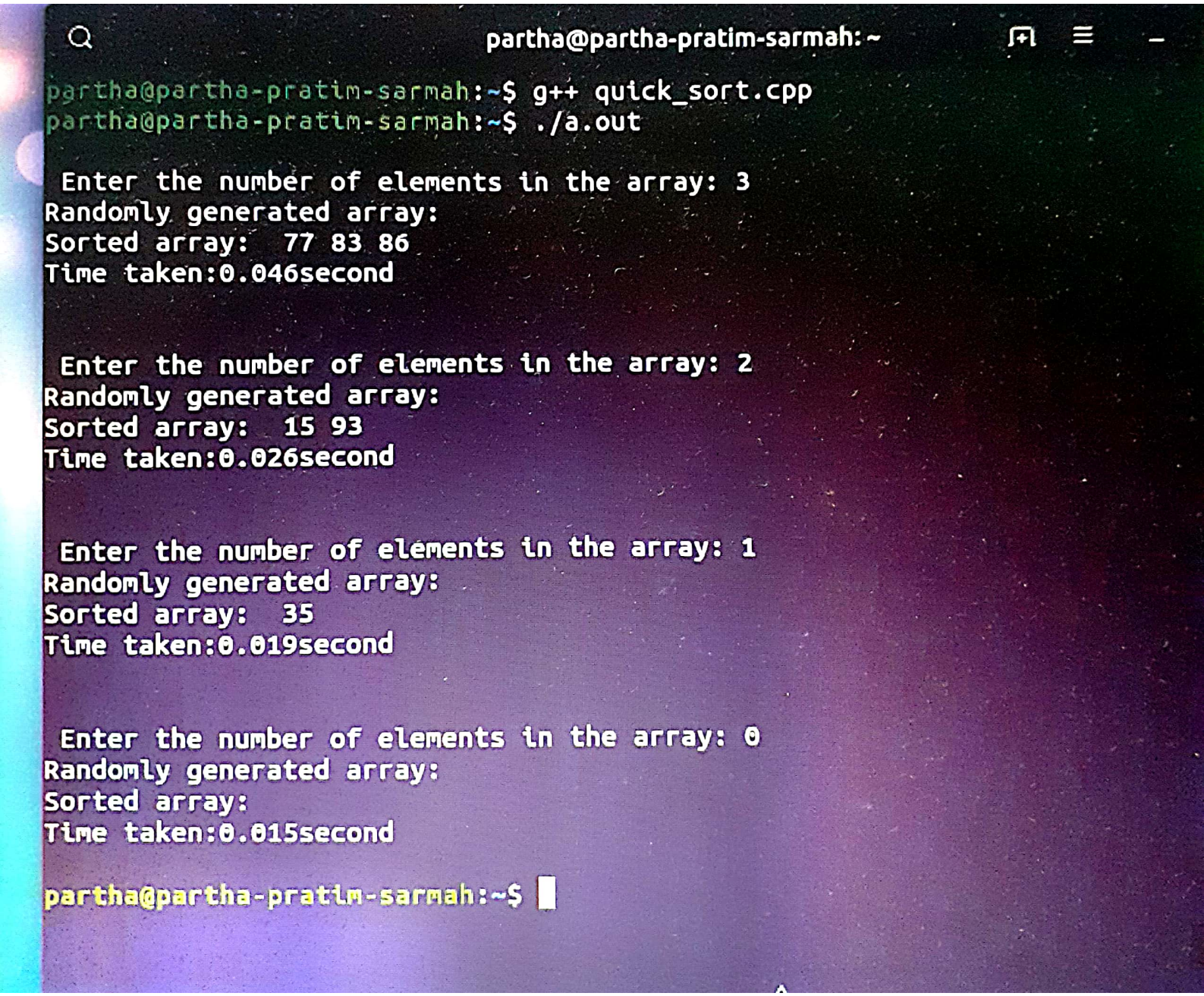
Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
    int end_clock = clock();
    cout << " Time taken: " << (end_clock - start_clock) / double
            (CLOCKS_PER_SEC) * 1000 << " second" << endl;
}
return 0;
}
```

———— o ————



1.2. Implement the ascending and desending order using Quick Sort.

Ans:-

```
#include <stdio.h>
int partition (int a[], int low, int high) // for ascending
{
    int pivot;
    int i, j, t;
    pivot = a[high];
    i = (low - 1);
    for (j = low; j <= high-1; j++)
    {
        if (a[j] < pivot)
        {
            i++;
            t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    t = a[i+1];
    a[i+1] = a[high];
    a[high] = t;
    return i+1;
}
```

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
int quicksort (int a[20], int low, int high)
{
    int pivot;
    if (low < high)
    {
        pivot = partition (a, low, high);
        quicksort (a, low, pivot-1);
        quicksort (a, pivot+1, high);
    }
    return 0;
}
void swap (int arr[], int one, int two) //for descending
{
    int temp = Array[one];
    Array[one] = Array[two];
    Array[two] = temp;
}// for descending order
int partition1 (int Array[], int left, int right)
{
    int pivot = Array[right];
    int leftPointer = left - 1;
    int rightPointer = right;
    for(;;)
    {
        while (Array[++leftPointer] > pivot)
        { }
        while (rightPointer > 0 && Array[--rightPointer] < pivot)
        { }
        if (leftPointer >= rightPointer) { break; }
        else { swap (Array, leftPointer, right);
        return leftPointer;
}
```

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
void Quicksort (int Array[ ], int left , int right)
{
    if (left < right)
    {   int Partition1 Point = partition 1 (Array, left, right);
        Quicksort (Array , left , Partition1 Point - 1);
        Quicksort (Array, Partition1 Point + 1, right);
    }
}
int main( )
{
    int a[20], i , j, low, high , n;
    printf (" Enter the size of the array: ");
    scanf ("%d",& n);
    printf (" Enter the elements in the array:");
    for (i= 0; i <n; i++)
        scanf ("%d", & a[i]);
    quicksort (a,0,n-1);
    printf (" After quicksort the arrays is in ascending order:");
    for (i=0; i<n; i++)
        printf ("%d" , a[i]);
    Quick sort (a,0,n-1);
    printf ("\nAfter quicksort the arrays is in descending order:");
    for (i=0; i<n;i++)
        printf ("%d", a[i]);
    printf ("\n");
    return 0;
```

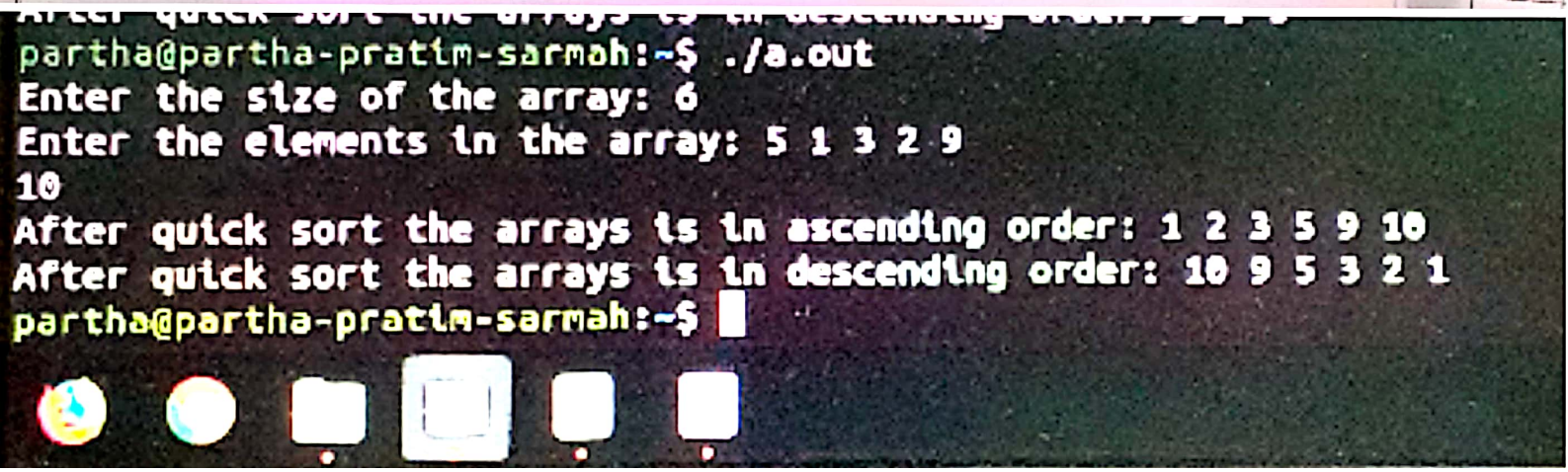
Output →

Enter the size of the array: 6

Enter the elements in the array: 5 1 3 2 9 10

After quick sort the arrays is in ascending order: 1 2 3 5 9 10

After quick sort the arrays is in descending order: 10 9 5 3 2 1



Expt. No. _____ Page No. _____

1.3. Implement Quick Sort with duplicate numbers in a given array/elements.

Answer :-

```
#include <bits/stdc++.h>
using namespace std;
void partition ( int a[], int l, int r, int &i, int &j)
{
    i = l-1, j = r;
    int p = l-1, q = r;
    int v = a[r];
    while (true)
    {   while (a[++l] < v);
        while (v < a[--j])
            if (j == l)
                break;
        if (i >= j) break;
        swap(a[i], a[j]);
        if (a[i] == v)
        {   p++;
            swap(a[p], a[i]);
        }
        if (a[j] == v)
        {   q--;
            swap(a[j], a[q]);
        }
    }
    swap(a[i], a[r]);
    j = i-1;
    for (int k = l; k < p; k++, j--)
        swap (a[k], a[j]);
```

Teacher's Signature _____

```
    i = i + 1;
    for (int k = r-1; k > q; k--, i++)
        swap (a[i], a[k]);
}
void quicksort (int a[], int l, int r)
{
    if (r <= l) return;
    int i, j;
    partition (a, l, r, i, j);
    quicksort (a, l, j);
    quicksort (a, i, r);
}
void printarr (int a[], int n)
{
    for (int i = 0; i < n; ++i)
        printf ("%d", a[i]);
    printf ("\n");
}
int main()
{
    printf (" Given Arrays elements are : \n");
    int a[] = {4, 9, 4, 4, 1, 9, 4, 4, 9, 4, 4, 1, 4};
    int size = sizeof (a) / sizeof (int);
    printarr (a, size);
    quicksort (a, 0, size-1);
    printf ("\n Sorted Arrays elements are : \n");
    printarr (a, size);
    return 0;
}
```

Teacher's Signature _____

Output →

Given Arrays elements are:

4 9 4 4 1 9 4 4 9 4 4 1 4

Sorted arrays elements are:

1 1 4 4 4 4 4 4 4 4 9 9 9

```
partha@partha-pratim-sarmah:~$ g++ 1.3.3.3
partha@partha-pratim-sarmah:~$ ./a.out
Given Arrays elements are:
4 9 4 4 1 9 4 4 9 4 4 1 4

Sorted arrays elements are:
1 1 4 4 4 4 4 4 4 4 4 9 9 9
partha@partha-pratim-sarmah:~$
```

Date _____

Expt. No. _____ Page No. _____

1.4 Finding Kth minimum and maximum element in Heap.

Ans

```
#include <iostream>
using namespace std;
void swap (int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
class MinHeap
{
    int size;
    int* arr;
    public:
        MinHeap (int size, int input[]);
        void heapify (int i);
        void buildHeap();

};
MinHeap :: MinHeap (int size, int input[])
{
    this -> size = size;
    this -> arr = input;
    buildHeap();
}
void MinHeap :: heapify (int i)
{
    if (i >= size/2)
        return;
    int smallest;
    int left = 2*i + 1;
    int right = 2*i + 2;
    smallest = arr[left] < arr[i] ? left : i;
```

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
    if(smallest != i)
    {   swap(arr[i], arr[smallest]);
        heapify(smallest);
    }
}
void MinHeap::buildHeap()
{
    for(int i = size/2 - 1; i >= 0; i--)
        heapify(i);
}
void FirstKelements(int arr[], int size, int K)
{
    MinHeap* m = new MinHeap(K, arr);
    for(int i=K; i<size; i++)
    {   if(arr[0] > arr[i])
            continue;
        else
        {   arr[0] = arr[i];
            m->heapify(0);
        }
    }
    for(int i=0; i<K; i++)
        cout << arr[i] << " ";
}
int main()
{   int arr[] = {11, 3, 2, 1, 15, 5, 4, 45, 88, 96, 50, 45};
    int size = sizeof(arr) / sizeof(arr[0]);
    int K = 3;
    FirstKelements(arr, size, k);
    cout << "\n";
    return 0;
}
```

Teacher's Signature _____

Output :

50 88 96

```
partha@partha-pratim-sarmah:
50 88 96
partha@partha-pratim-sarmah:
```

Expt. No. _____ Page No. _____

1.5 Build Min heap, Max heap and sort the given elements

Ans

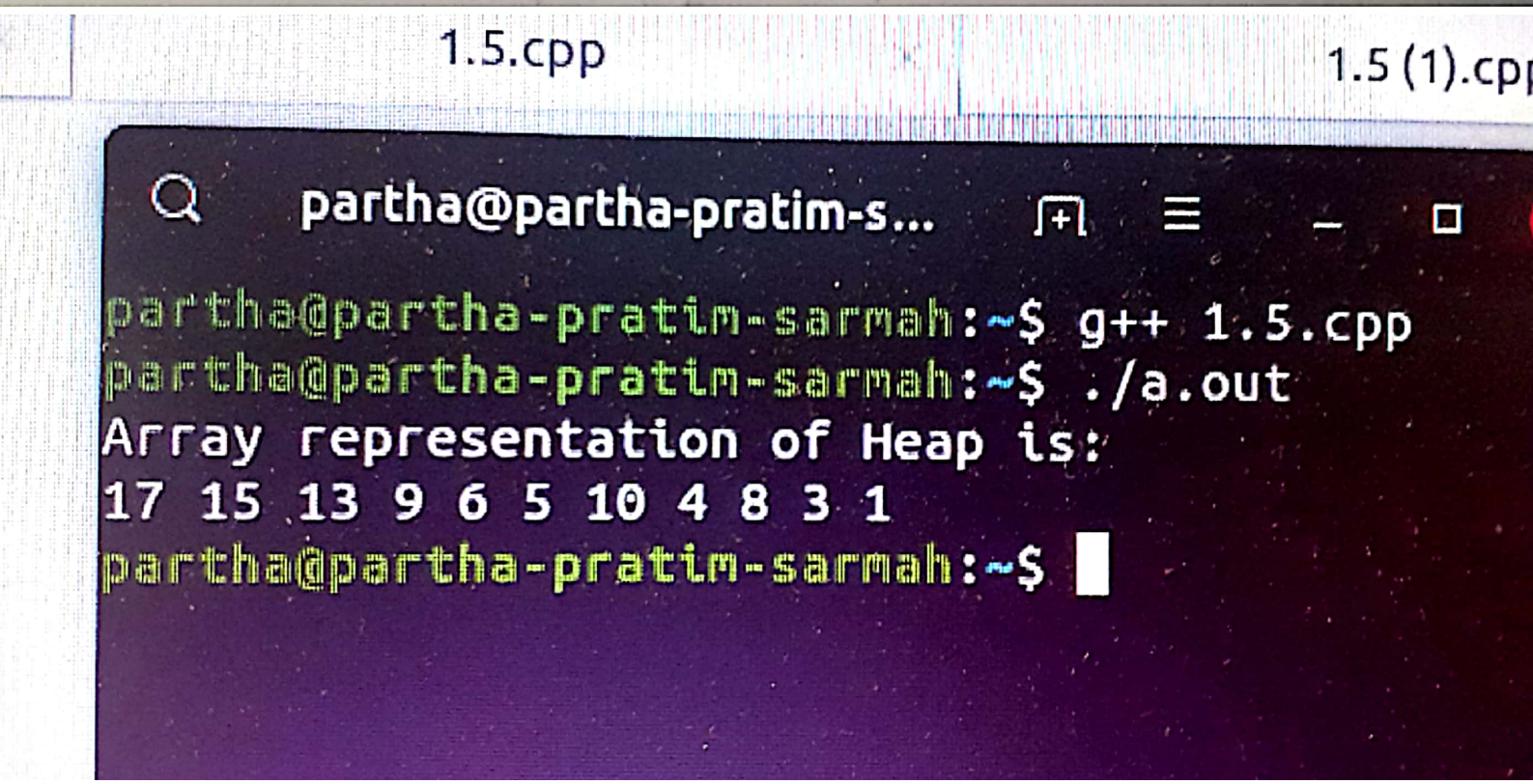
```
#include <iostream>
using namespace std;
void heapify(int arr[], int n, int i)
{   int largest = i;
    int l = 2*i+1;
    int r = 2*i+2;
    if (l<n && arr[i] > arr[largest])
        largest = l;
    if (r<n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {   swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
void buildHeap(int arr[], int n)
{   int startIdx = (n/2) - 1;
    for(int i= startIdx; i>=0; i--)
        heapify(arr, n, i);
}
void printHeap(int arr[], int n)
{   cout << "Array representation of Heap is: \n";
    for(int i=0; i<n; ++i)
        cout << arr[i] << " ";
}
int main()
{   int arr[] = {1, 3, 5, 4, 6, 13, 10, 9, 8, 15, 17};
    int n= sizeof(arr)/sizeof(arr[0]);
    buildHeap(arr,n); printHeap(arr, n);
}   return 0;
```

Teacher's Signature _____

Output →

Array representation of Heap is :

17 15 13 9 6 5 10 4 8 3 1



Date _____

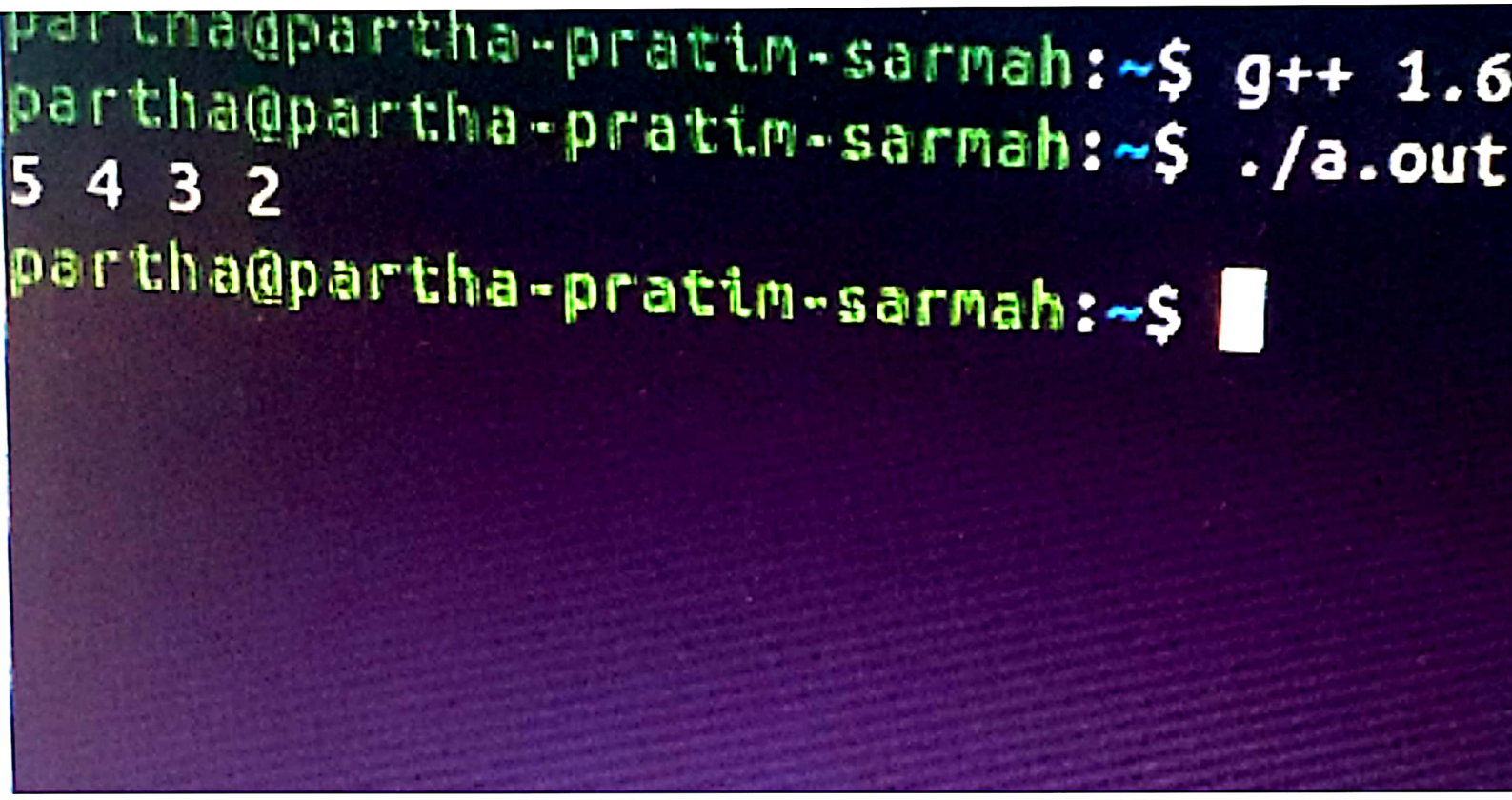
Expt. No. _____ Page No. _____

1.6 Delete Kth indexed element in Min Heap and Max Heap.

Ans:

```
#include <iostream>
using namespace std;
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    if(l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
void deleteRoot(int arr[], int& n)
{
    int lastElement = arr[n-1];
    arr[0] = lastElement;
    n = n-1;
    heapify(arr, n, 0);
}
void printArray(int arr[], int n)
{
    for(int i=0; i<n; ++i)
        cout << arr[i] << " ";
    cout << endl;
}
int main(){
    int arr[] = {10, 5, 3, 2, 4};
    int n=sizeof(arr) / sizeof(arr[0]);
    deleteRoot(arr,n); printArray(arr,n);
    return 0;}
```

Teacher's Signature _____



Output → 5 4 3 2

Expt. ____ Page No. ____

2 Greedy Algorithms

2.1 Implement the job sequencing with deadlines problem using the fixed tuple size formulation.

Ans

```
#include <iostream>
#include <algorithm>
using namespace std;
struct Job
{
    char id;
    int dead;
    int profit;
};
bool compare (Job a, Job b)
{
    return (a.profit > b.profit);
}
void jobschedule (Job arr[], int n)
{
    sort (arr, arr+n, compare);
    int result [n];
    bool slot [n];
    for (int i=0; i<n; i++)
        slot [i] = false;
    for (int i=0; i<n; i++)
    {
        for (int j = min (n, arr [i]. dead)-1 ; j>=0; j--)
        {
            if (slot [j] == false)
            {
                result [j] = i;
                slot [j] = true;
                break;
            }
        }
    }
```

Teacher's Signature ____

Date _____

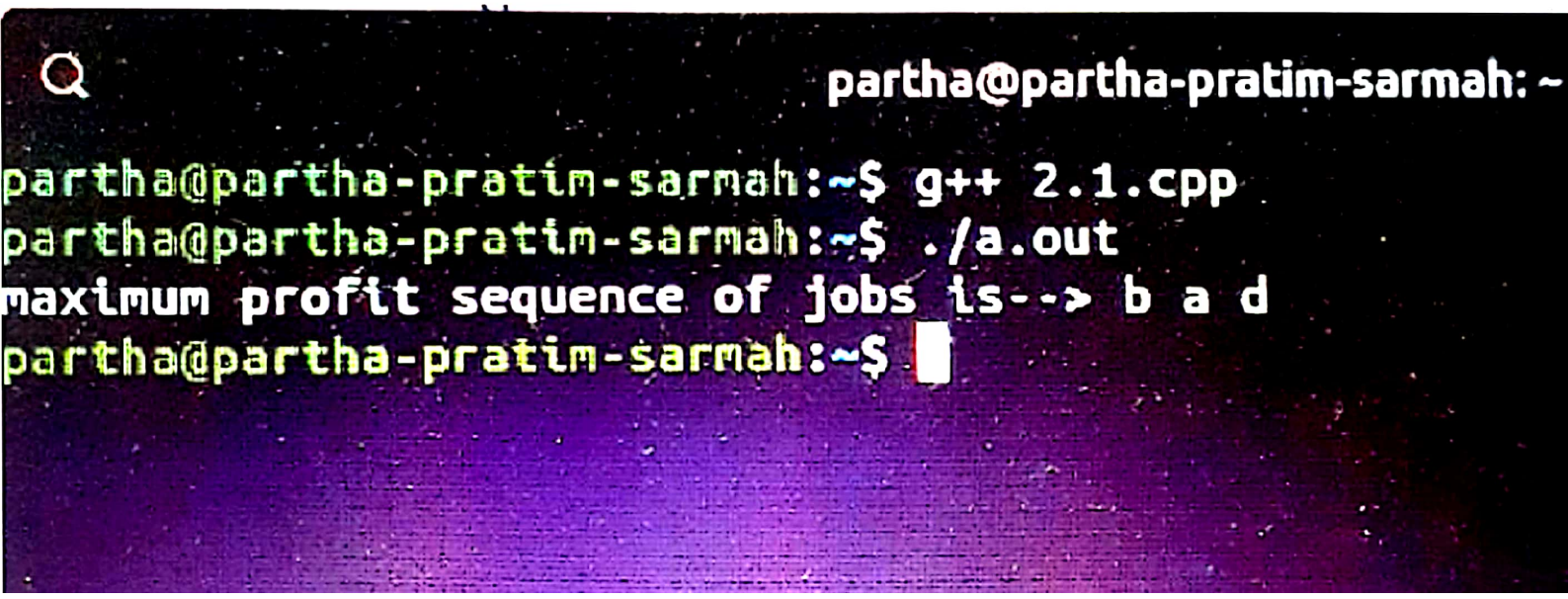
Expt. No. _____ Page No. _____

```
    for (int i=0 ; i<n; i++)
        if (slot[i])
            cout << arr[result[i]].id << " " ;
}

int main()
{
    Job arr[] = { {'a', 2, 20}, {'b', 2, 15}, {'c', 1, 10},
                  {'d', 3, 5}, {'e', 3, 1}};
    int n = 5;
    cout << "Maximum profit sequence of jobs is --> ";
    jobschedule(arr, n);
    cout << endl;
}
```

Output →

maximum profit sequence of jobs is --> b a d



2.2 Implement Knapsack problem.

Ans

```
#include <bits/stdc++.h>
using namespace std;
struct Item
{
    int value, weight;
    Item(int value, int weight) : value(value), weight(weight)
    {}
};
bool cmp(struct Item a, struct Item b)
{
    double r1 = (double)a.value / a.weight;
    double r2 = (double)b.value / b.weight;
    return r1 > r2;
}
```

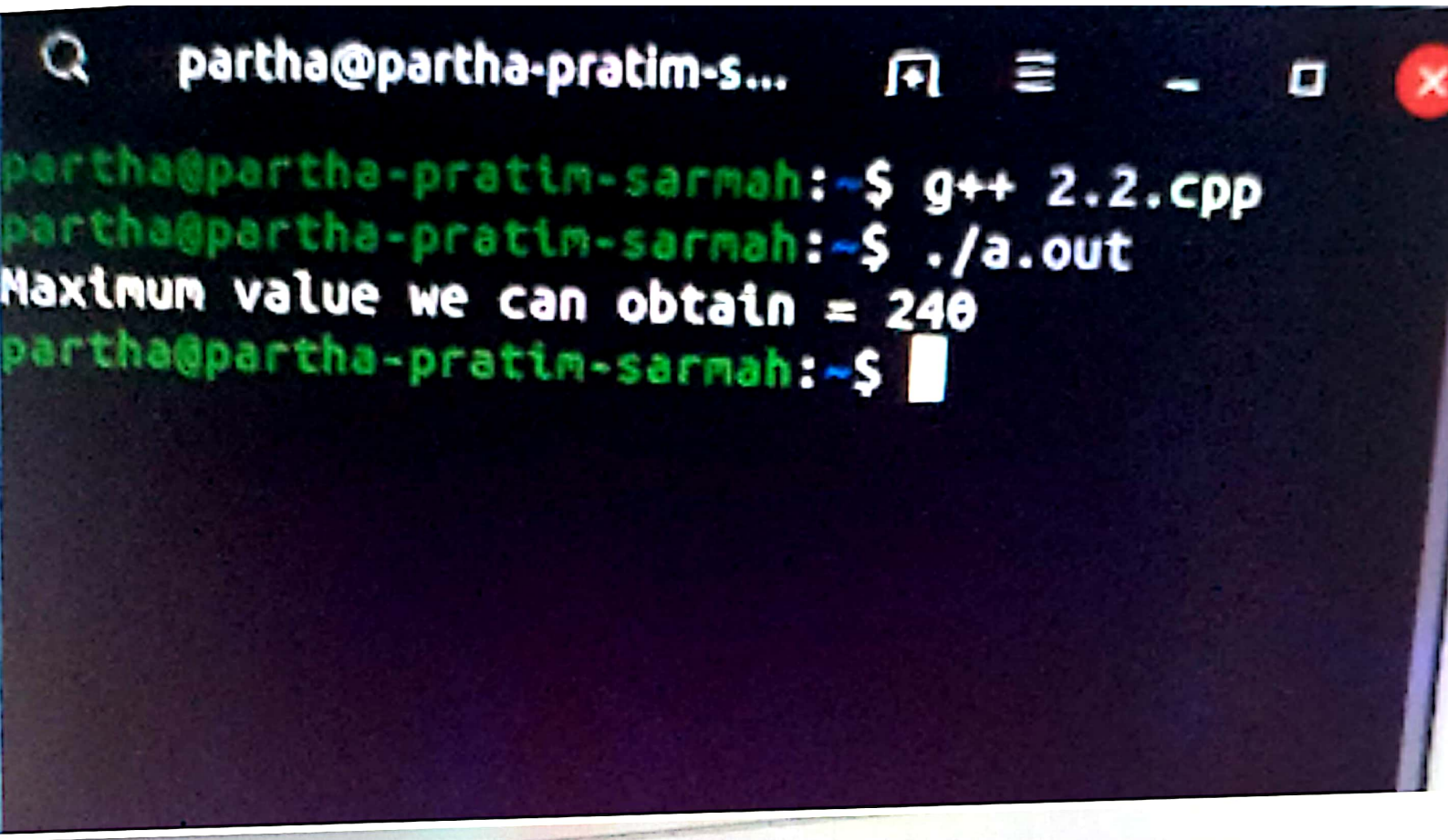
Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
double fractionalKnapsack(int W, struct Item arr[], int n)
{
    sort(arr, arr + n, cmp);
    int curWeight = 0;
    double finalvalue = 0.0;
    for (int i=0; i<n; i++)
    {
        if (curWeight + arr[i].weight <= W)
        {
            curWeight += arr[i].weight;
            finalvalue += arr[i].value;
        }
        else
        {
            int remain = W - curWeight;
            finalvalue += arr[i].value * ((double)remain / arr[i].
                                          weight);
            break;
        }
    }
    return finalvalue;
}
int main()
{   int W = 50;
    Item arr[] = {{60,10}, {100, 20}, {120, 30}};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum value we can obtain = "
         << fractionalKnapsack(W, arr, n) << endl;
    return 0;
}
```

Teacher's Signature _____



Output →

Maximum value we can obtain = 240

Expt. No. _____ Date _____

Page No. _____

2.3 Implement the file or code compression using Huffman's algorithm.

Ans

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

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
    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);
    }
}
```

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
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 );
}
```

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
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 -> arr[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);
```

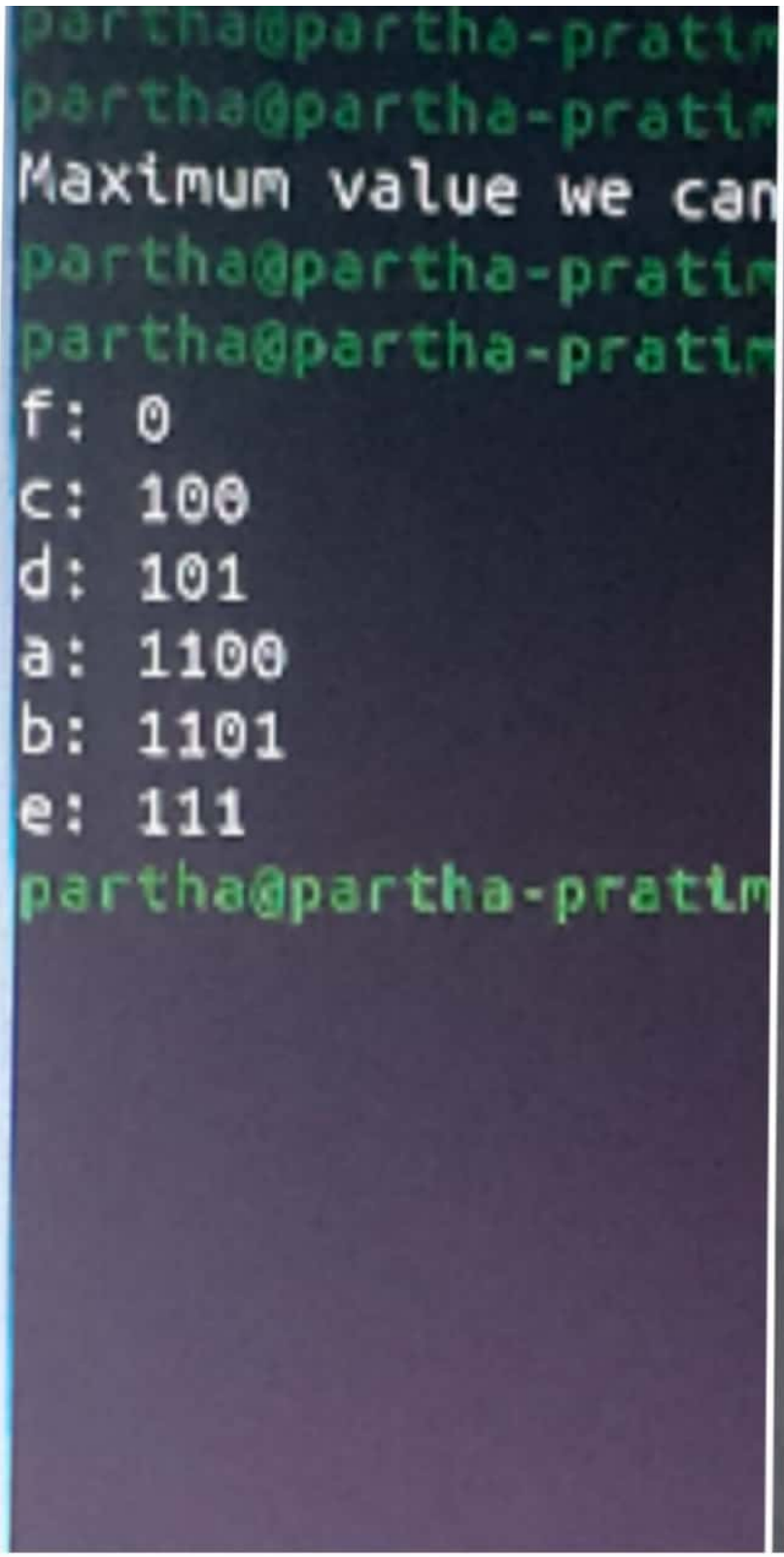
Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
    top -> left = left;
    top -> right = right;
    insertMinHeap (minHeap, top);
}
return extractMin (minHeap);
void printCodes ( struct MinHeapNode* root, int arr[], int top)
{
    if (root -> left)
    {
        arr[top] = 0;
        printCodes (root -> left, arr, top + 1);
    }
    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)
{
    struct MinHeapNode* root = buildHuffmanTree (data, freq, size);
    int arr[MAX_TREE_HT], top = 0;
    printCodes (root, arr, top);
}
int main()
{
    char arr[] = {'a', 'b', 'c', 'd', 'e', 'f'};
    int freq[] = {5, 9, 12, 13, 16, 45};
    int size = sizeof (arr) / sizeof (arr[0]);
    HuffmanCoder (arr, freq, size);
    return 0;
}
```

Teacher's Signature _____



output →

f: 0

c: 100

d: 101

a: 1100

b: 1101

e: 111

Date _____

Expt. No. _____ Page No. _____

2.4 Implement the minimisation of records movement using Optimal Merge Pattern Algorithm.

Ans

```
#include <iostream>
using namespace std;
int main()
{
    int i, k, a[10], c[10], n, l;
    cout<<" Enter the no. of elements: ";
    cin >> n;
    cout<<" \n Enter the sorted elements for optimal merge
          pattern: ";
    for (i=0; i<n; i++)
        cin>>a[i];
    i=0;
    k=0;
    c[k] = a[i] + a[i+1];
    i = 2;
    while (i<n)
    {
        k++;
        if ((c[k-1]+a[i]) <= (a[i] + a[i+1]))
            c[k] = c[k-1] + a[i];
        else
        {
            c[k] = a[i] + a[i+1];
            i = i+2;
            while (i<n)
            {
                k++;
                if ((c[k-1] + a[i]) <= (c[k-2] + a[i]))
                    c[k] = c[k-1] + a[i];
                else
                    c[k] = c[k-2] + a[i];
                i++;
            }
        }
    i++;
```

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
    }
    k++
    c[k] = c[k-1] + c[k-2];
    cout << "\n The optimal sum are as follows.....\n";
    for ( k=0; k<n-1; k++)
    {
        cout << c[k] << "\t";
    }
    l=0;
    for ( k=0; k<n-1; k++)
        l = l + c[k];
    cout << "\n The external path length is...." << l;
    return 0;
}
```

```
 The external path length is ......33
partha@partha-pratim-sarmah:~$ ./a.out
Enter the no. of elements: 5

Enter the sorted elments for optimal merge pattern: 5 6 7 8 9 10


The optimal sum are as follows......

11      15      20      35

 The external path length is ......81
partha@partha-pratim-sarmah:~$
```

```
l=0;
for(k=0;k<n-1;k++)
{
```

Output →

Enter the no of elements : 5

Enter the sorted elements for optimal merge pattern : 5 6 7 8 9 10

The optimal sum are as follows . . .

11 15 20 35

The external path length is 81

3. Graph Algorithm

3.1 Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

Ans

```
#include <stdlib.h>
#include <cstring>
#include <iostream>
using namespace std;
struct Edge
{
    int src, dest, weight;
};
struct Graph
{
    int V, E;
    struct Edge* edge;
};
```

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*)malloc(sizeof
                                        (struct Graph));
    graph->V = V;
    graph->E = E;
    graph->edge = (struct Edge*)malloc(graph->E *
                                        sizeof(struct Edge));
    return graph;
}
struct subset
{
    int parent;
    int rank;
};
int find(struct subset subsets[], int i)
{
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);
    return subsets[i].parent;
}
void Union(struct subset subsets[], int x, int y)
{
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);
        if (subsets[xroot].rank < subsets[yroot].rank)
            subsets[xroot].parent = yroot;
        else if (subsets[xroot].rank > subsets[yroot].rank)
            subsets[yroot].parent = xroot;
        else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
}
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
int myComp ( const void* a, const void* b)
{
    struct Edge* a1 = (struct Edge*) a;
    struct Edge* b1 = ( struct Edge*) b;
    return a1->weight > b1->weight;
}
void KruskalMST (struct Graph* graph)
{   int V= graph->V;
    Struct Edge result[V];
    int e=0;
    int i=0;
    qsort (graph-> edge, graph ->E, sizeof( graph ->Edge[0])
                                        myComp);
    struct subset *subsets = ( struct subset*) malloc
                                (V * sizeof(struct subset));

    for ( int v=0; v<V; ++v)
    {
        subsets[V]. parent =V;
        subsets[v]. rank =0;
    }
    while (e<V-1)
    {
        Struct Edge next_edge = graph-> edge [i++];
        int x = find (subsets, next_edge. src);
        int y = find (subsets, next_edge . dest);
        if (x!=y)
        {   result [e++] = next_edge;
            union ( subsets, x, y);
        }
    }
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
    cout << " Following are the edges in the constructed MST\n";
    for(i=0; i<e; ++i)
    {
        cout << result[i].src << " -- " << result[i].dest << "=="
                                        << result[i].weight << endl;
    }
    return;
}

int main()
{   int V=4;
    int E=5;
    struct Graph* graph = createGraph(V, E);

    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 10;

    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 6;
    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;
    graph->edge[2].weight = 5;
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 15;
    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
    graph->edge[4].weight = 4;
    KruskalMST(graph);
    return 0;
}
```

Teacher's Signature _____

```
partha@partha-pratim-sarmah:~$ g++ 3.1.cpp
partha@partha-pratim-sarmah:~$ ./a.out
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
partha@partha-pratim-sarmah:~$
```

Output →

Following are the edges in the constructed MST

2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Date _____

Expt. No. _____ Page No. _____

3.2 Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

Ans

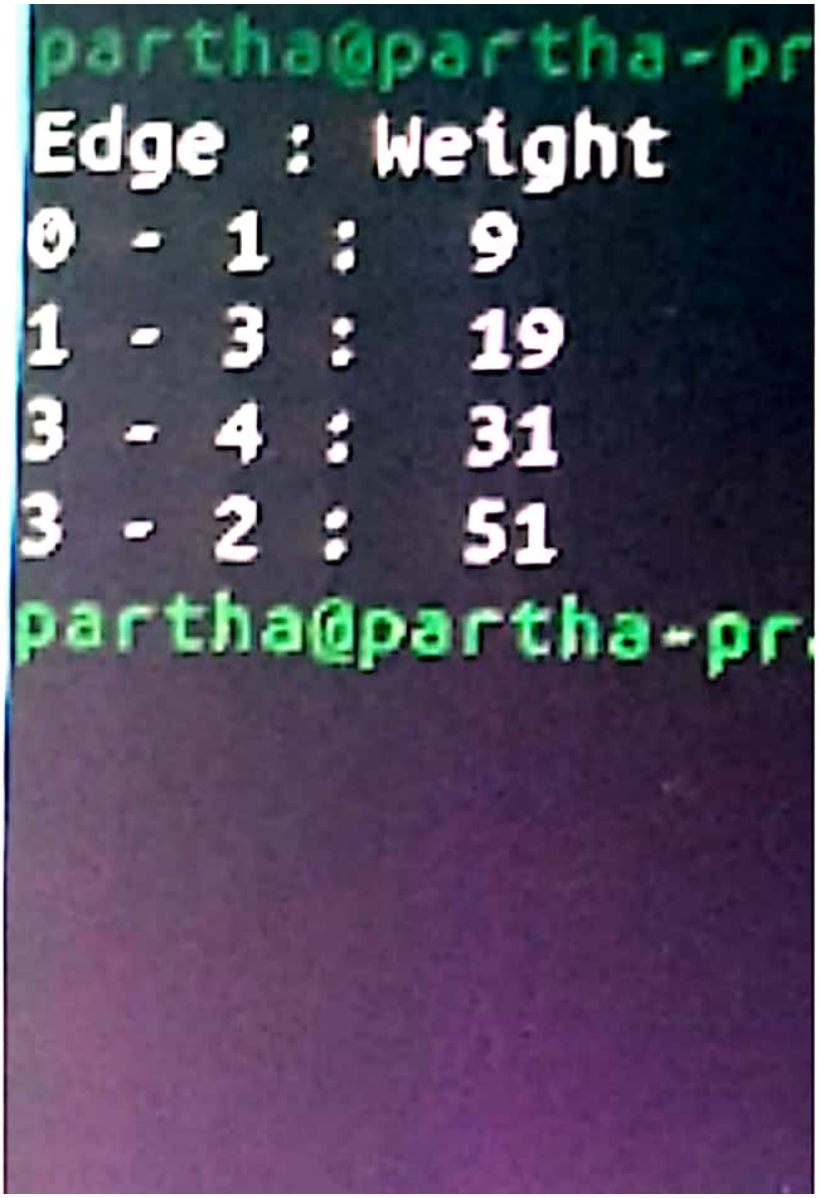
```
#include <iostream>
#include <cstring>
using namespace std;
# define INF 9999999
# define V 5
int G[V][V] = {
                { 0,9, 75, 0, 0},
                { 9,0, 95, 19, 42},
                { 75, 95, 0, 51, 66},
                { 0, 19, 51, 0, 31},
                { 0, 42, 66, 31, 0}
              };
int main()
{   int no_edge;
    int selected[V];
    memset(selected, false, sizeof(selected));
    no_edge = 0;
    selected[0] = true;
    int x, y;
    cout << "Edge" << " :" << "Weight";
    cout << endl;
    while (no_edge < V-1)
    {
        int min = INF;
        x = 0;
        y = 0;
        for (int i=0; i<V; i++)
        {   if (selected[i])
```

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
    {
        for (int j=0; j<V; j++)
        {
            if (!selected[j] && G[i][j])
            {
                if (min > G[i][j])
                {
                    min = G[i][j];
                    x = i;
                    y = j;
                }
            }
        }
    }
    cout << x << " -" << y << " : " << G[x][y];
    cout << endl;
    selected[y] = true;
    no_edge ++;
}
return 0;
}
```



Output →

Edge : Weight

0 - 1 : 9

1 - 3 : 19

3 - 4 : 31

3 - 2 : 51

4. Shortest Path Finding in Graph

4.1 From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

Ans

```
#include <iostream>
#include <limits.h>
#include <stdio.h>
#define V 9
int minDistance (int dist[], bool sptSet[])
{
    int min = INT_MAX, min_index;
```

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
    for ( int v= 0; v<V; v++)
    {       if ( sptSet [v] == false && dist [v] <= min)
            {
                    min = dist [v], min_index = v;
            }
    }
    return min_index;
}

void printSolution (int dist [])
{
    printf ("Vertex \t\t Distance from Source\n");
    for ( int i=0; i<V; i++)
        printf (" %d \t\t %d\n", i, dist [i]);
}

void dijkstra ( int graph [V][V], int src)
{
    int     dist [V];
    bool    sptSet [V];
    for (int i=0; i<V; i++)
        dist [i] = INT_MAX , sptSet [i] = false;
    dist [src] =0;
    for ( int count =0; count <V-1 ; count ++)
    {
        int  u = minDistance (dist , sptSet);
        sptSet [u] = true;
        for (int v=0; v<V; v++ )
        {       if ( ! sptSet[v] && graph [u] [v] && dist[u] !=
                INT_MAX && dist[u] + graph [u] [v] <dist [v])
                {       dist [v] = dist [u] + graph [u] [v];
                }
        }
        printSolution (dist);
}
```

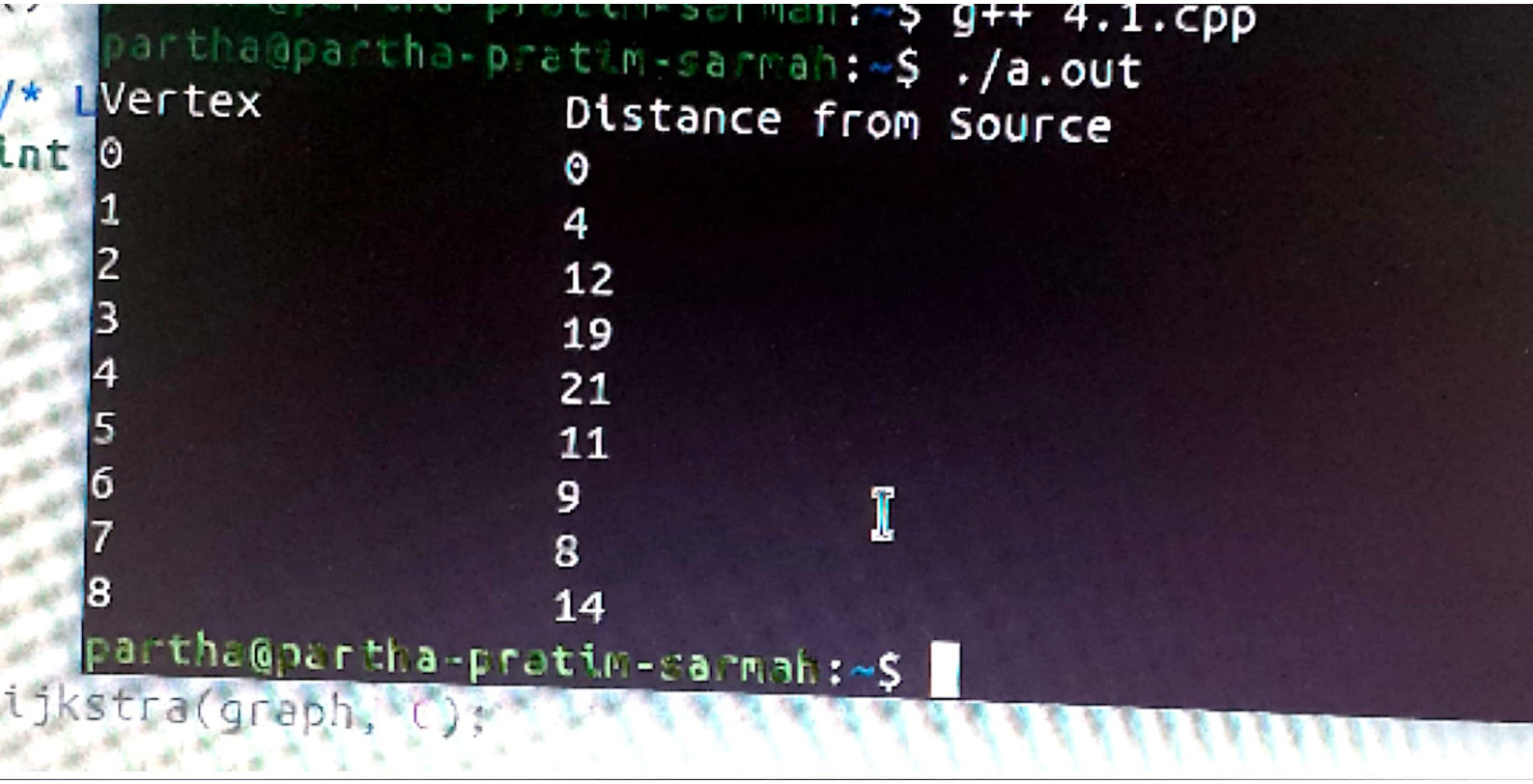
Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
int main()
{
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0},
                        {4, 0, 8, 0, 0, 0, 0, 11, 0},
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
    dijkstra(graph, 0);
    return 0;
}
```

—— o ——



Date _____

Expt. No. _____ Page No. _____

4.2 Draw simple, connected weighted graph with 8 vertices and 16 edges, each with unique edge weight. Identify one vertex as a start vertex and obtain shortest path using Dijkstra's algorithm.

Ans

```
#include <bits/stdc++.h>
using namespace std;
#define INF 0x3f3f3f3f
typedef pair<int, int> iPair;
class Graph{
        int V;
        list <pair <int, int>>* adj
        public:
            Graph (int V);
            void addEdgeRev(int u, int v, int w);
            void shortestPath (int s);

};
Graph:: Graph (int V)
{   this->V = V;
    adj = new list <iPair>[V];
}
void Graph:: addEdgeRev(int u, int v, int w)
{
    adj[v].push_back(make_pair(u,w));
}
void Graph:: shortestPath (int dest)
{
    priority_queue< iPair, vector<iPair>, greater<iPair>> pq;
    vector <int> dist (V, INF);
    pq.push(make_pair(0, dest));
    dist[dest]=0;
```

Teacher's Signature _____

Expt. No. _____ Page No. _____

```
while (!pq.empty())
{
    int u = pq.top().second;
    pq.pop();
    list<pair<int, int>>::iterator i;
    for (i = adj[u].begin(); i != adj[u].end(); ++i)
    {
        int v = (*i).first;
        int weight = (*i).second;
        if (dist[v] > dist[u] + weight)
        {
            dist[v] = dist[u] + weight;
            pq.push(make_pair(dist[v], v));
        }
    }
}
printf("Destination Vertex Distance"
       "from at vertex \n");
for(int i=0; i<V; ++i)
    printf("%d \t\t %d\n", i, dist[i]);
}
int main()
{
    int V=5;
    Graph g(V);
    g.addEdgeRev(0,2,1);
    g.addEdgeRev(0,4,5);
    g.addEdgeRev(1,4,1);
    g.addEdgeRev(2,0,10);
    g.addEdgeRev(2,3,5);
    g.addEdgeRev(3,1,1);
```

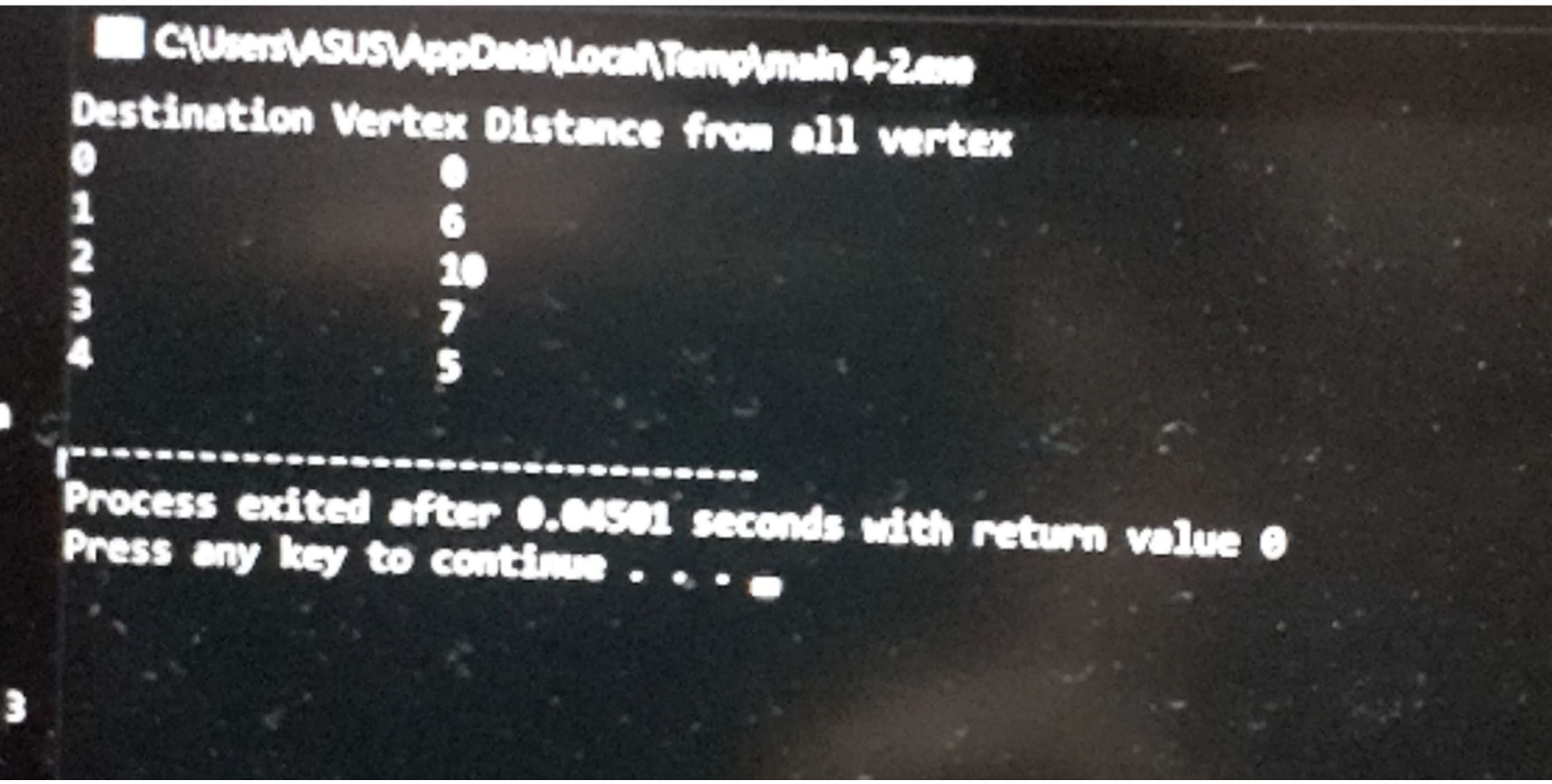
Teacher's Signature _____

Expt. No. _____

```
    g.addEdgeRev (4, 0, 5);
    g.addEdgeRev (4,2,100);
    g.addEdgeRev (4, 3, 5);
    g.shortestPath (0);
    return 0;
}
```

Output →

Distination Vertex	Distance from all vertex
0	0
1	6
2	10
3	7
4	5



4.3 Find the Kth shortest path between two given nodes of a graph.

Answer :—

```
#include <iostream>
#include <stdio.h>
using namespace std;
#define INFINITY 999
#define max 5
void dijkstra (int G[max][max], int n, int startnode);
int main()
{
    int G[max][max] = {{0,1,0,3,10}, {1,0,5,0,0}, {0,5,0,2,1},
                       {3,0,2,0,6}, {10,0,1,6,0}};
    int n=5, u=0;
    dijkstra (G,n,u);
}   return 0;
```

Teacher's Signature _____

```
void dijkstra ( int G[max][max], int n, int startnode )
{
    int cost[max][max], distance[max], pred[max];
    int visited[max], count, mindistance, nextnode, i, j;
    for (i = 0; i < n; i++)
        for ( j=0; j<n; j++)
    if ( G[i][j] == 0)
        cost [i][j] = INFINITY;
    else
        cost[i][j] = G[i][j];
    for( i = 0; i<n; i++)
    {
        distance [i] = cost [startnode][i];
        pred [i] = startnode;
        visited [i] = 0;
    }
    distance [startnode] = 0;
    visited [startnode] = 1;
    count = 1;
    while ( count < n-1)
    {
        mindistance = INFINITY;
        for (i=0; i<n; i++)
            if ( distance [i] < mindistance && !visited[i])
            {
                mindistance = distance [i];
                nextnode = i;
            }
        visited [nextnode] = 1;
        for (i=0; i<n; i++)
            if ( !visited [i]
```

Teacher's Signature _____

Expt. No. _____ Page No. _____

```
if (mindistance + cost [nextnode][i] < distance[i])
{
    distance[i] = mindistance + cost[nextnode][i];
    pred[i] = nextnode;
}
count ++;
}
for(i=0; i<n; i++)
if(i != startnode)
{
    cout << "\n Distance of node" << i << "=" << distance[i];
    cout << "\n Path =" << i;
    j = i;
    do
    {
      j = pred[j];
      cout << "<-" << j;
    }
    while(j != startnode);
  }
}
```

Output →

Distance of node 1 = 1

Path = 1 <- 0

Distance of node 2 = 5

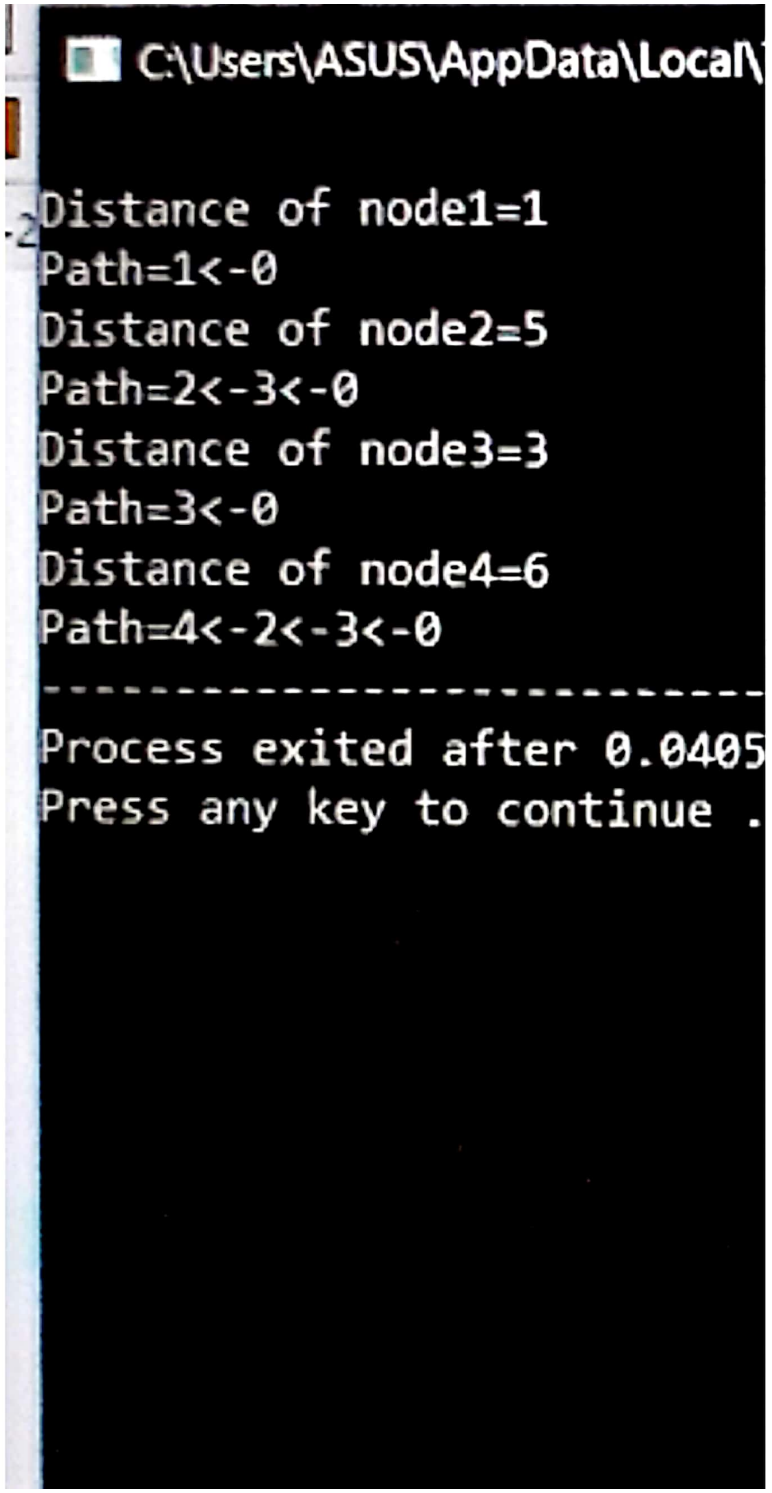
Path = 2 <- 3 <- 0

Distance of node 3 = 3

Path = 3 <- 0

Distance of node 4 = 6

Path = 4 <- 2 <- 3 <- 0



5 Dynamic Programming

5.1 Implement the Bottom Up Dynamic Programming Approach for matrix chain multiplication.

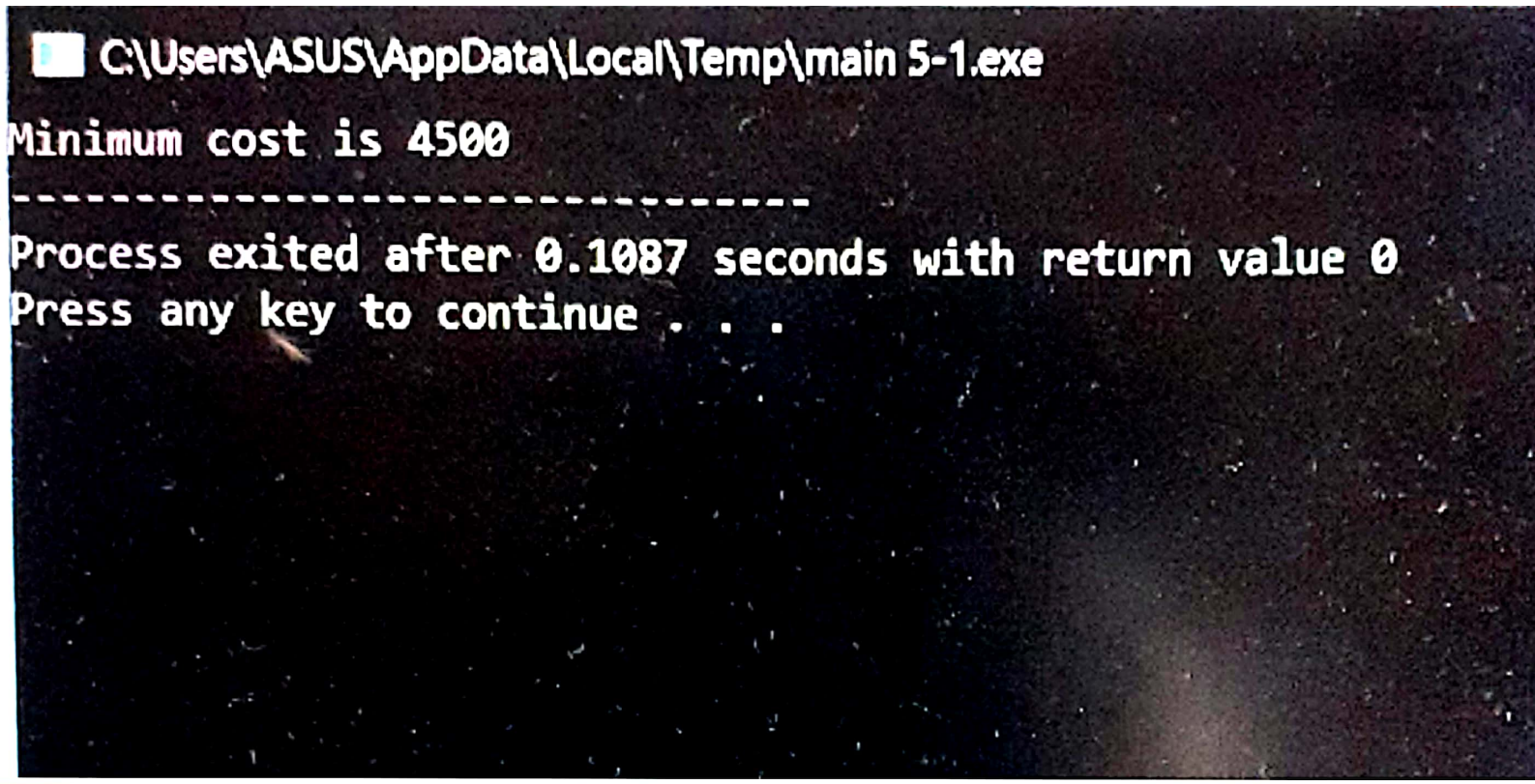
Ans:-

```
#include <iostream>
#include <climits>
using namespace std;
int MatrixChainMultiplication ( int dims [ ], int i, int j )
{
    if ( j <= i + 1)
        return 0;
    int min = INT_MAX;
    for (int K = i + 1; K <= j - 1; K++)
    {
        int cost = MatrixChainMultiplication(dims, i, k);
        cost += MatrixChainMultiplication(dims, K, j);
        cost += dims [i] * dims[K] * dims [j];
        if ( cost < min )
            min = cost;
    }
    return min;
}
int main ( )
{
    int dims[ ] = {10, 30, 5, 60};
```

Teacher's Signature _____

Expt. No. _____ Page No. _____

```
int n = sizeof(dims) / sizeof(dims[0]);
cout << "Minimum cost is " << MatrixChainMultiplication(dims, 0, n-1);
return 0;
}
```



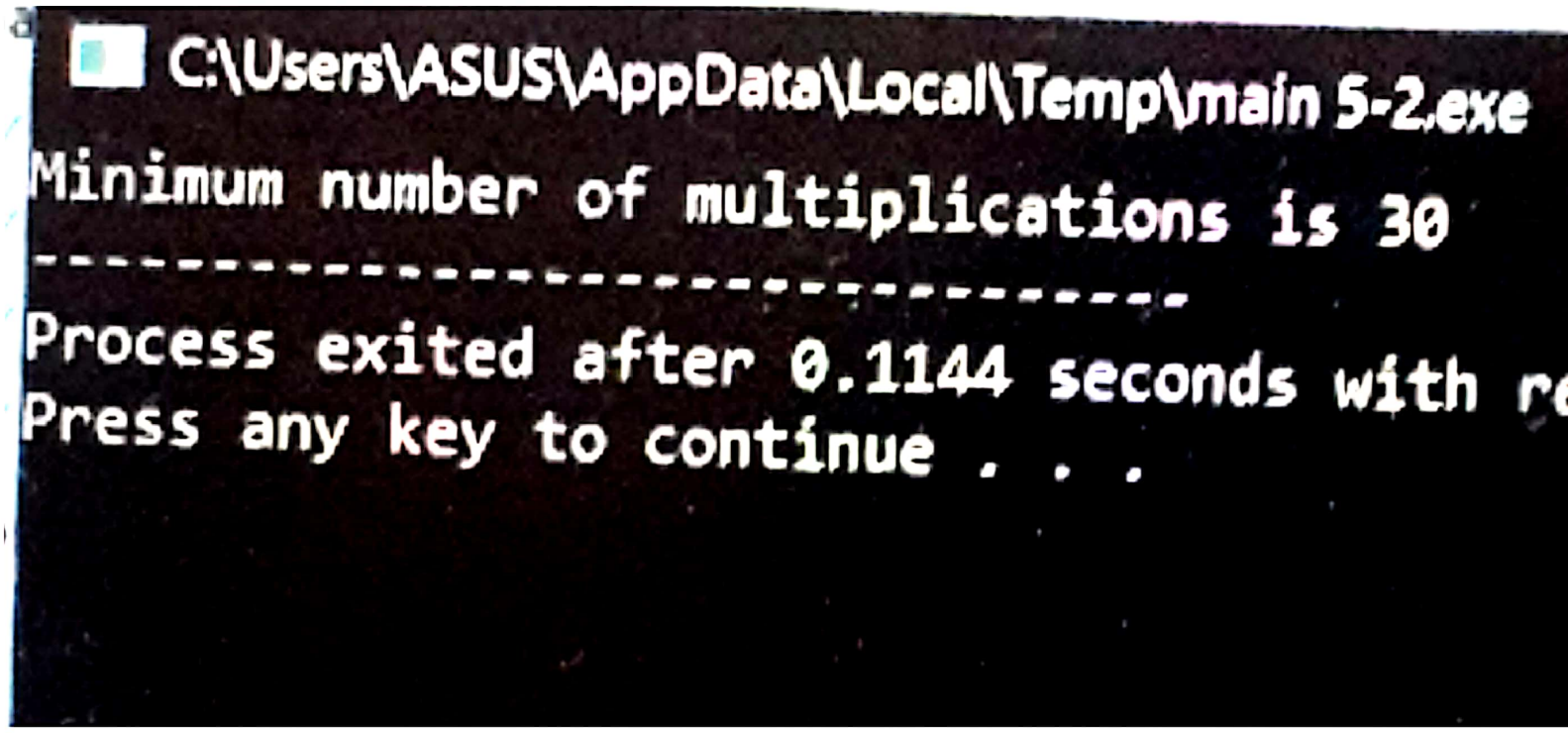
Output - Minimum cost is 4500

5.3 Implement the Top Down Dynamic Programming Memoization Approach for matrix chain multiplication.

Ans:

```
#include <bits/stdc++.h>
using namespace std;
int MatrixChainOrder (int P[], int i, int j)
{
    if (i == j)
        return 0;
    int k;
    int min = INT_MAX;
    int count;
    for (k = i; k < j ; k++
    {
        count = MatrixChainOrder (p, i, k)
              + MatrixChainOrder ( p, k+1, j)
              + P[i-1] * P[k] * P[j];
        if (count < min)
            min = count;
    }
    return min;
}
int main()
{
    int arr[] = {1, 2, 3, 4, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Minimum number of multiplication is "
         << MatrixChainOrder(arr, 1, n-1);
}
```

Teacher's Signature _____



5.2 Output →

Minimum number of multiplication is 30

Date _____

Expt. No. _____ Page No. _____

5.3 Implement the Bottom Up Dynamic Programming Approach for longest common subsequence.

Ans

```
#include <iostream>
#include <string>
using namespace std;
int LCSLength (string X, string Y)
{
    int m = X.length(), n = Y.length();
    int lookup [m+1] [n+1];
    for (int i=0; i<=m; i++)
        lookup [i] [0] = 0;
    for (int j=0; j<=n; j++)
        lookup [0] [j] = 0;
    for (int i=1; i<=m; i++)
    {
        for (int j=1; j<=n; j++)
        {
            if (X[i-1] == Y[j-1])
                lookup [i] [j] = lookup [i-1] [j-1] + 1;
            else
                lookup [i] [j] = max (lookup [i-1] [j],
                                      lookup [i] [j-1]);
        }
    }
    return lookup [m] [n];
}
int main()
{
    string X = "XMJYAUZ", Y = "MZJAWXU";
    cout << "The length of LCS is " << LCSLength (X, Y);
    return 0;
}
```

Teacher's Signature _____

Output → The Length of LCS is 4