GURU NANAK DEV ENGINEERING COLLEGE

LUDHIANA



DAA PRACTICAL

Submitted to: Submitted by:

Name: Sameer Sharma

Class: D3 CS B2

Roll No: 1715131

Uni. Roll No: 1805257

**EXPERIMENT NO. 1**

Write a program to find out a roll number from college database using binary search algorithm.

**PROGRAM**

1. **Iterative (Non-recursive approach)**

#include <iostream>

#include <stdio.h>

using namespace std;

int main()

{

//clrscr();

int n, i, search, first, last, middle;

int arr[] = {1715372, 1715373, 1715374, 1715375, 1715376};

int x;

cout<<”The list of roll numbers saved is as follows:”<<endl;

for (int i=0; i<5; i++) {

cout<<arr[i]<<"\t";

}

cout<<"Enter a number to find :";

cin>>search;

first = 0;

last = n-1;

middle = (first+last)/2;

while (first <= last)

{

if(arr[middle] < search)

{

first = middle + 1;

}

else if(arr[middle] == search)

{

cout<<search<<" found at location "<<middle+1<<"\n";

break;

}

else

{

last = middle - 1;

}

middle = (first + last)/2;

}

if(first > last)

{

cout<<"Not found! "<<search<<" is not present in the list.";

}

return 0;

getch();

}

1. **Recursive approach**

// C++ program to implement recursive Binary Search

#include <iostream>

using namespace std;

// A recursive binary search function. It returns

// location of x in given array arr[l..r] is present,

// otherwise -1

int binarySearch(int arr[], int l, int r, int x)

{

if (r >= l) {

int mid = l + (r - l) / 2;

// If the element is present at the middle

// itself

if (arr[mid] == x)

return mid;

// If element is smaller than mid, then

// it can only be present in left subarray

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

// Else the element can only be present

// in right subarray

return binarySearch(arr, mid + 1, r, x); }

// We reach here when element is not

// present in array

return -1;

}

int main(void){

int arr[] = {1715372, 1715373, 1715374, 1715375, 1715376};

int x;

cout<<”The list of roll numbers saved is as follows:”<<endl;

for (int i=0; i<5; i++) {

cout<<arr[i]<<"\t";

}

cout<<"Enter the element to search: "<<endl;

cin>>x;

int n = sizeof(arr) / sizeof(arr[0]);

int result = binarySearch(arr, 0, n - 1, x);

(result == -1) ? cout << "Element is not present in array"

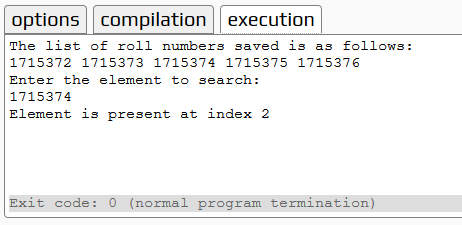
: cout << "Element is present at index " << result;

return 0;

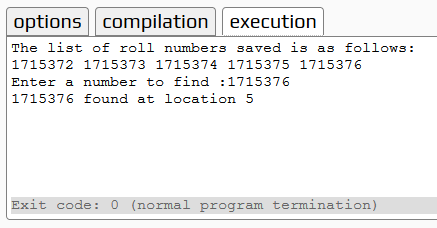
}

**Output**

1. **Iterative (Non-recursive approach)**



1. **Recursive approach**



**EXPERIMENT NO. 2**

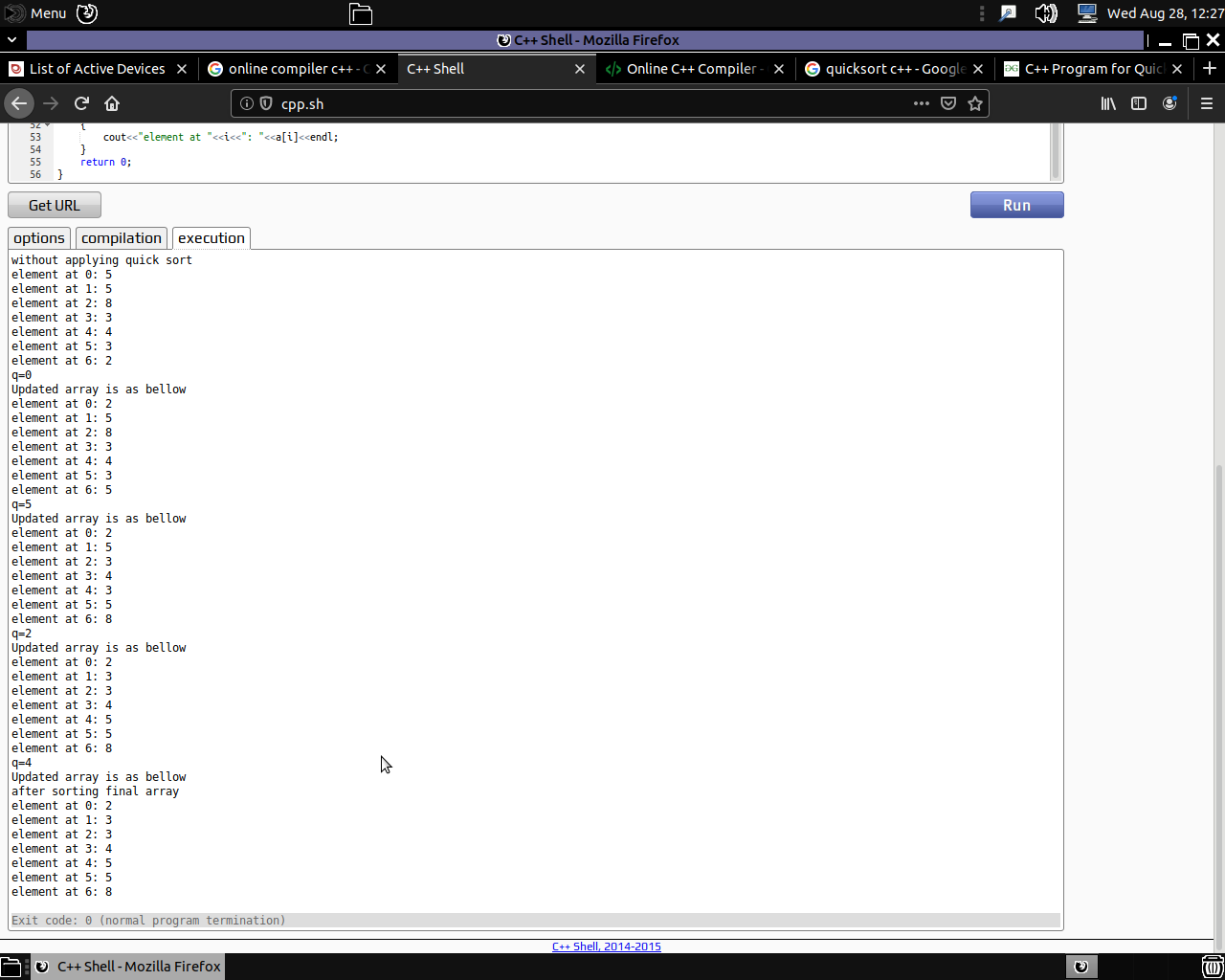
Write a program to sort the university roll numbers of your class using Quick sort method and determine the time required to sort the element.

**PROGRAM**

#include <iostream>  
using namespace std;  
int partition(int a[],int p,int r) {  
    int x = a[r];  
    int temp;  
    int i = p-1;  
    for (int j = p; j <= r-1; j++) {  
        if (a[j] <= x) {  
            i = i+1;

// swapping the element less than the pivot element in the list  
            temp = a[i];   
            a[i] = a[j];  
            a[j] = temp;  
        }  
    }  
    temp = a[i+1]; //swapping pivot element  
    a[i+1] = a[r];  
    a[r] = temp;  
    return (i+1);  
}  
void quicksort(int a[],int p,int r) {  
    if (p<r) {  
        for (int i=0; i<=6; i++) {  
            cout<<"element at "<<i<<": "<<a[i]<<endl;  
        }  
        int q = partition(a,p,r);  
        cout<<"q="<<q<<endl;  
        cout<<"Updated array is as bellow"<<endl;  
        quicksort(a,p,q-1);  
        quicksort(a,q+1,r);    
    }  
}  
int main()  
{  
    int a[7] = {5,5,8,3,4,3,2};  
    int p,r;  
    p=0;  
    r=6;  
    cout<<"without applying quick sort"<<endl;  
    quicksort(a,p,r);  
    cout<<"after sorting final array"<<endl;  
    for (int i=0; i<7; i++)  
    {  
        cout<<"element at "<<i<<": "<<a[i]<<endl;  
    }  
    return 0;  
}

**Output**



**EXPERIMENT NO. 3**

Write a program to sort the roll numbers of your class using merge sort algorithm and determine the time required to sort the elements.

**PROGRAM**

#include<iostream>

using namespace std;

void swapping(int &a, int &b) { //swap the content of a and b

int temp;

temp = a;

a = b;

b = temp;

}

void display(int \*array, int size) {

for(int i = 0; i<size; i++)

cout << array[i] << " ";

cout << endl;

}

void merge(int \*array, int l, int m, int r) {

int i, j, k, nl, nr;

//size of left and right sub-arrays

nl = m-l+1; nr = r-m;

int larr[nl], rarr[nr];

//fill left and right sub-arrays

for(i = 0; i<nl; i++)

larr[i] = array[l+i];

for(j = 0; j<nr; j++)

rarr[j] = array[m+1+j];

i = 0; j = 0; k = l;

//marge temp arrays to real array

while(i < nl && j<nr) {

if(larr[i] <= rarr[j]) {

array[k] = larr[i];

i++;

}else{

array[k] = rarr[j];

j++;

}

k++;

}

while(i<nl) { //extra element in left array

array[k] = larr[i];

i++; k++;

}

while(j<nr) { //extra element in right array

array[k] = rarr[j];

j++; k++;

}

}

void mergeSort(int \*array, int l, int r) {

int m;

if(l < r) {

int m = l+(r-l)/2;

// Sort first and second arrays

mergeSort(array, l, m);

mergeSort(array, m+1, r);

merge(array, l, m, r);

}

}

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n]; //create an array with given number of elements

cout << "Enter elements:" << endl;

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

cin >> arr[i];

}

cout << "Array before Sorting: ";

display(arr, n);

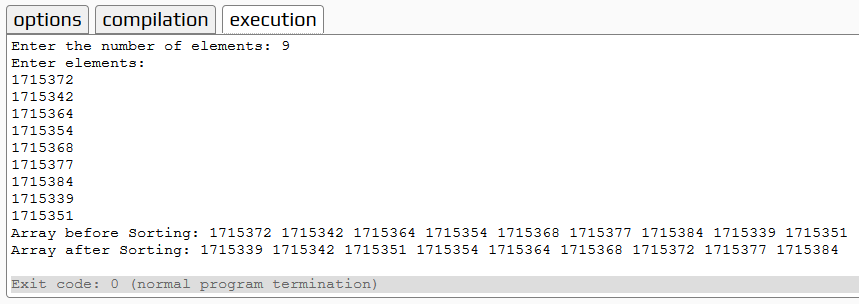
mergeSort(arr, 0, n-1); //(n-1) for last index

cout << "Array after Sorting: ";

display(arr, n);

}

**Output**



**EXPERIMENT NO. 4**

Write a program to solve 0/1 knapsack using Greedy algorithm

**PROGRAM**

1. **Iterative (Non-recursive approach)**

#include<iostream>

using namespace std;

// A utility function that returns maximum of two integers

int max(int a, int b) { return (a > b)? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity W

int knapSack(int W, int wt[], int val[], int n)

{

cout<<"inside function"<<endl;

int i, w;

int K[n+1][W+1];

cout<<"Variables intialized"<<endl;

// Build table K[][] in bottom up manner

for (i = 0; i <= n; i++)

{

for (w = 0; w <= W; w++)

{

if (i==0 || w==0)

K[i][w] = 0;

else if (wt[i-1] <= w)

K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);

else

K[i][w] = K[i-1][w];

}

}

cout<<"for loop checking whether the elements to be considered or not"<<K[n][W]<<endl;

return K[n][W];

}

int main()

{

cout<<"Iterative Method"<<endl;

int val[] = {60, 100, 120};

int wt[] = {10, 20, 30};

int W = 50;

int n = sizeof(val)/sizeof(val[0]);

cout<<"Initial variables are as folows: \n"<<endl;

cout<<"Size of array is: "<<n<<endl;

cout<<"Weight of Knapsack is: "<<W<<endl;

cout<<"Values are: "<<endl;

for(int i=0; i<n;i++)

{

cout<<val[i]<<endl;

}

cout<<"Weights are: "<<endl;

for(int i=0; i<n;i++)

{

cout<<wt[i]<<endl;

}

cout<<"calling function with values \nW- weight of knapsack,\nwt- array of weight of items,\nval- value of various items in an array form,\nn- total number of elements in the array"<<endl;

cout<< "Value returned from the function is: "<<knapSack(W, wt, val, n);

return 0;

}

1. **Recursive approach**

/\* A Naive recursive implementation of 0-1 Knapsack problem \*/

#include <iostream>

using namespace std;

// A utility function that returns maximum of two integers

int max(int a, int b) { return (a > b)? a : b; }

// Returns the maximum value that

// can be put in a knapsack of capacity W

int knapSack(int W, int wt[], int val[], int n)

{

// Base Case

if (n == 0 || W == 0)

return 0;

// If weight of the nth item is more

// than Knapsack capacity W, then

// this item cannot be included

// in the optimal solution

if (wt[n-1] > W)

{

cout<<"if weight is greather then the next value is considered"<<endl;

return knapSack(W, wt, val, n-1);

}

// Return the maximum of two cases:

// (1) nth item included

// (2) not included

else

{

cout<<"Max function called to check the max value between 2 knapsack return values"<<endl;

return max( val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),

knapSack(W, wt, val, n-1) );

}

}

// Driver code

int main()

{

int val[] = {60, 100, 120};

int wt[] = {10, 20, 30};

int W = 50;

int n = sizeof(val)/sizeof(val[0]);

cout<<"Initial variables are as folows: \n"<<endl;

cout<<"Size of array is: "<<n<<endl;

cout<<"Weight of Knapsack is: "<<W<<endl;

cout<<"Values are: "<<endl;

for(int i=0; i<n;i++)

{

cout<<val[i]<<endl;

}

cout<<"Weights are: "<<endl;

for(int i=0; i<n;i++)

{

cout<<wt[i]<<endl;

}

cout<<"calling function with values \nW- weight of knapsack,\nwt- array of weight of items,\nval- value of various items in an array form,\nn- total number of elements in the array"<<endl;

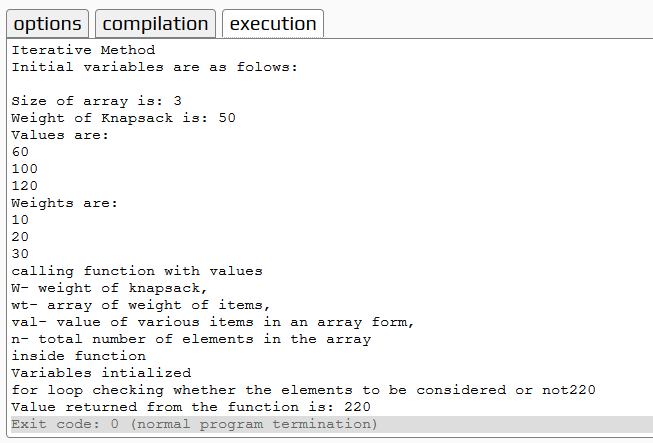
cout<< "Value returned from the function is: "<<knapSack(W, wt, val, n);

return 0;

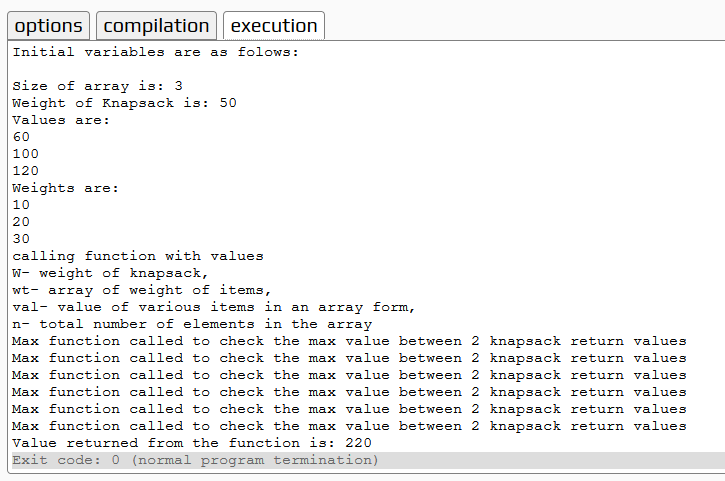
}

**Output**

1. **Iterative (Non-recursive approach)**

****

1. **Recursive approach**

****

**EXPERIMENT NO. 5**

Write a program to find minimum cost to set the phone lines to connect all the cities of your state using Prim’s algorithm.

**PROGRAM**

// C++ code to find out minimum cost

// path to connect all the cities

#include <iostream>

#include <limits>

#include <vector>

using namespace std;

// Function to find out minimum valued node

// among the nodes which are not yet included in MST

int minnode(int n, int keyval[], bool mstset[]) {

int mini = numeric\_limits<int>::max();

int mini\_index;

// Loop through all the values of the nodes

// which are not yet included in MST and find

// the minimum valued one.

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

if (mstset[i] == false && keyval[i] < mini) {

mini = keyval[i], mini\_index = i;

}

}

return mini\_index;

}

// Function to find out the MST and

// the cost of the MST.

void findcost(int n, vector<vector<int>> city) {

// Array to store the parent node of a

// particular node.

int parent[n];

// Array to store key value of each node.

int keyval[n];

// Boolean Array to hold bool values whether

// a node is included in MST or not.

bool mstset[n];

// Set all the key values to infinite and

// none of the nodes is included in MST.

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

keyval[i] = numeric\_limits<int>::max();

mstset[i] = false;

}

// Start to find the MST from node 0.

// Parent of node 0 is none so set -1.

// key value or minimum cost to reach

// 0th node from 0th node is 0.

parent[0] = -1;

keyval[0] = 0;

// Find the rest n-1 nodes of MST.

for (int i = 0; i < n - 1; i++) {

// First find out the minimum node

// among the nodes which are not yet

// included in MST.

int u = minnode(n, keyval, mstset);

// Now the uth node is included in MST.

mstset[u] = true;

// Update the values of neighbor

// nodes of u which are not yet

// included in MST.

cout<<u;

for (int v = 0; v < n; v++) {

if (city[u][v] && mstset[v] == false &&

city[u][v] < keyval[v]) {

keyval[v] = city[u][v];

parent[v] = u;

}

}

}

// Find out the cost by adding

// the edge values of MST.

int cost = 0;

for (int i = 1; i < n; i++)

cost += city[parent[i]][i];

cout <<"The minimum cost is: \t"<< cost << endl;

}

// Utility Program:

int main() {

int n1,temp1;

cout<<"Enter the number of cities you want:";

cin>>n1;

vector<vector<int> > city1;

vector<int> data;

for(int i = 0; i < n1; i++){

vector<int> temp;

cout<<"Enter the cost of "<<i+1<<" city with other citites:"<<endl;

for(int j = 0; j < n1; j++){

cin >> temp1;

temp.push\_back(temp1);

}

city1.push\_back(temp);

temp.clear();

}

for (int i = 0; i < city1.size(); i++) {

for (int j = 0; j < city1[i].size(); j++)

cout << city1[i][j] << " ";

cout << endl;

}

cout<<"Finding the minimum cost"<<endl;

findcost(n1, city1);

return 0;

}

**Output**

