Logo, company name

Description automatically generated**Charutar Vidya Mandal University**

**VV Nagar, Anand**

**A.D. Patel Institute of Technology New V.V.Nagar, Anand**

**Design and Analysis of Algorithms**

**Subject code:-102045601**

**Computer Engineering (Semester-5)**

**Submitted by:**

**Name: Vikram Mali**

**Enrolment No: 12002040601070**

**Academic Year (2022-2023)**

**Internal Faculty Head of the Department External Faculty**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No.** | **Aim** | **Date** | **Sign** |
| 1 | Write a program to sort given elements of an array in ascending order using bubble sort. Analyze the time complexity for best, average and worst case. |  |  |
| 2 | Write a program to sort given elements of an array in ascending order using selection sort. Analyze the time complexity for best, average and worst case |  |  |
| 3 | Write a program to implement heap sort. |  |  |
| 4 | Write a program to search given element from an array using sequential search and binary search. Analyze the time complexity for best, average and worst case. |  |  |
| 5 | Write a program to sort given elements of an array in ascending order using merge sort. Analyze the time complexity for best, average and worst case. |  |  |
| 6 | Write a program to sort given elements of an array in ascending order using quick sort. Analyze the time complexity for best, average and worst case. |  |  |
| 7 | Write a program to implement making change problem using greedy algorithm. |  |  |
| 8 | Write a program to implement the knapsack problem using greedy algorithm. |  |  |
| 9 | Write a program to implement making change problem using dynamic programming. |  |  |
| 10 | Write a program to implement the knapsack problem using dynamic programming. |  |  |
| 11 | Write a program to implement Floyd’s algorithm for finding shortest path using dynamic programming. |  |  |
| 12 | Write a program to implement chained matrix multiplication using dynamic programming. |  |  |
| 13 | Write a program to implement longest common subsequence using dynamic programming. |  |  |

**1.Write a program to sort given elements of an array in ascending order using bubble sort. Analyze the time complexity for best, average and worst case.**

#include <bits/stdc++.h>

using namespace std;

void Sort(int arr[], int n)

{

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

{

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

{

if (arr[j] > arr[j + 1])

{

swap(arr[j], arr[j + 1]);

}

}

}

}

int main()

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

Sort(arr, n);

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

{

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

}

return 0;

}

Output:



Best Case Time Complexity:(N)

Average Case Time Complexity: θ(N2)

Worst Case Time Complexity: O(N2)

**2.Write a program to sort given elements of an array in ascending order using selection sort. Analyze the time complexity for best, average and worst case.**

Code in C++:

#include <bits/stdc++.h>

using namespace std;

void Sort(int arr[], int n)

{

int i, j, min\_idx;

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

{

min\_idx = i;

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

if (arr[j] < arr[min\_idx])

min\_idx = j;

if(min\_idx!=i)

swap(arr[min\_idx],arr[i]);

}

}

int main()

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

Sort(arr, n);

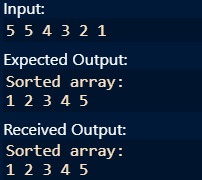
cout << "Sorted array: \n";

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

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

return 0;

}



Best Case Time Complexity:(N2)

Average Case Time Complexity: θ(N2)

Worst Case Time Complexity: O(N2)

**3. Write a program to implement heap sort.**

Code in C++:

#include <bits/stdc++.h>

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 heapSort(int arr[], int N)

{

for (int i = N / 2 - 1; i >= 0; i--)

heapify(arr, N, i);

for (int i = N - 1; i > 0; i--) {

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

int main()

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

heapSort(arr, n);

cout << "Sorted array is \n";

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

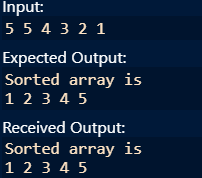
{

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

}

}

Output:



**4. Write a program to search given element from an array using sequential search and binary search. Analyze the time complexity for best, average and worst case.**

**Sequential Search:**

#include <iostream>

using namespace std;

int search(int arr[], int N, int x)

{

int i;

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

if (arr[i] == x)

return i;

return -1;

}

int main(void)

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

int result = search(arr, n, 5);

if(result==-1)

{

cout << "Element is not present in array";

}

else

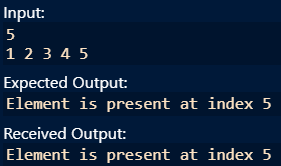
{

cout << "Element is present at index " << result+1;

}

return 0;

}



Best Case Time Complexity:(1)

Average Case Time Complexity: θ(N)

Worst Case Time Complexity: O(N)

**Binary Search:**

Code in C++:

#include <bits/stdc++.h>

using namespace std;

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

{

if (r >= l) {

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

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

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

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

}

return -1;

}

int main()

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

int result = binarySearch(arr, 0, n - 1, 5);

if(result == -1)

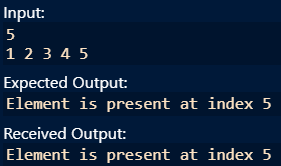
cout << "Element is not present in array";

else

cout << "Element is present at index " << result+1;

return 0;

}



Best Case Time Complexity:(1)

Average Case Time Complexity: θ(logN)

Worst Case Time Complexity: O(logN)

**5. Write a program to sort given elements of an array in ascending order using merge sort. Analyze the time complexity for best, average and worst case.**

#include <bits/stdc++.h>

using namespace std;

void merge(int arr[], int p, int q, int r)

{

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

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

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

M[j] = arr[q + 1 + j];

int i=0, j=0, k=p;

while (i < n1 && j < n2)

{

if (L[i] <= M[j])

arr[k] = L[i],i++;

else

arr[k] = M[j],j++;

k++;

}

while (i < n1)

{

arr[k] = L[i],i++,k++;

}

while (j < n2)

{

arr[k] = M[j],j++,k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

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

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

int main()

{

int n;

cin >> n;

int arr[n];

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

cin >> arr[i];

mergeSort(arr, 0, n - 1);

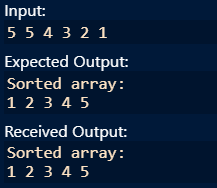
cout << "Sorted array: \n";

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

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

return 0;

}



Best Case Time Complexity:(NlogN)

Average Case Time Complexity: θ(NlogN)

Worst Case Time Complexity: O(NlogN)

**6. Write a program to sort given elements of an array in ascending order using quick sort. Analyze the time complexity for best, average and worst case.**

#include <bits/stdc++.h>

using namespace std;

int partition(int arr[], int start, int end)

{

int pivot = arr[start];

int count = 0;

for (int i = start + 1; i <= end; i++) {

if (arr[i] <= pivot)

count++;

}

int pivotIndex = start + count;

swap(arr[pivotIndex], arr[start]);

int i = start, j = end;

while (i < pivotIndex && j > pivotIndex) {

while (arr[i] <= pivot)

i++;

while (arr[j] > pivot)

j--;

if (i < pivotIndex && j > pivotIndex)

swap(arr[i++], arr[j--]);

}

return pivotIndex;

}

void quickSort(int arr[], int start, int end)

{

if (start >= end)

return;

int p = partition(arr, start, end);

quickSort(arr, start, p - 1);

quickSort(arr, p + 1, end);

}

int main()

{

int n;

cin >> n;

int arr[n];

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

cin >> arr[i];

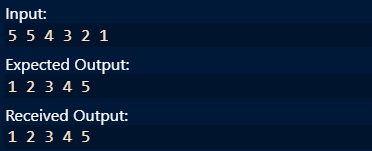
quickSort(arr, 0, n - 1);

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

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

return 0;

}



Best Case Time Complexity:(NlogN)

Average Case Time Complexity: θ(NlogN)

Worst Case Time Complexity: O(N2)

**7. Write a program to implement making change problem using greedy algorithm.**

#include <bits/stdc++.h>

using namespace std;

int denomination[] = { 1, 2, 5, 10, 20,50, 100, 500, 1000 };

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

void findMin(int V)

{

sort(denomination, denomination + n,greater<int>());

vector<int> ans;

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

while (V >= denomination[i]) {

V -= denomination[i];

ans.push\_back(denomination[i]);

}

}

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

cout << ans[i] << " ";

}

int main()

{

int n;

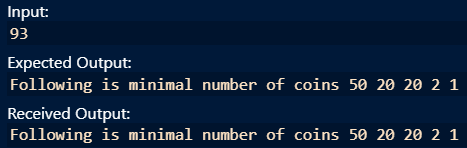
cin>>n;

cout << "Following is minimal number of coins ";

findMin(n);

return 0;

}



**8.Write a program to implement the knapsack problem using greedy algorithm.**

#include <bits/stdc++.h>

using namespace std;

typedef struct {

int value;

int weight;

float density;

}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 << " Value ";

cin >> items[i].value;

cout << "Enter "<< i+1 << " Weight ";

cin >> items[i].weight;

}

}

void display(Item items[],int sizeOfItems){

cout << "values: ";

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

cout << items[i].value << "\t";

}

cout << endl << "weight: ";

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

cout << items[i].weight << "\t";

}

cout << endl;

}

bool compare(Item i1, Item i2){

return (i1.density > i2.density);

}

float knapsack(Item items[],int sizeOfItems, int W){

float totalValue=0, totalWeight=0;

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

items[i].density = items[i].value/items[i].weight;

}

sort(items, items+sizeOfItems,compare);

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

if(totalWeight + items[i].weight<= W){

totalValue += items[i].value ;

totalWeight += items[i].weight;

} else {

int wt = W-totalWeight;

totalValue += (wt \* items[i].density);

totalWeight += wt;

break;

}

}

cout << "total weight in bag " << totalWeight<<endl;

return totalValue;

}

int main()

{

int W;

Item items[3];

input(items,3);

cout << "Entered data \n";

display(items,3);

cout<< "Enter Knapsack weight \n";

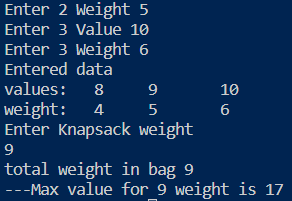
cin >> W;

float mxVal = knapsack(items,3,W);

cout << "---Max value for "<< W <<" weight is "<< mxVal;

return 0;

}



**9.Write a program to implement making change problem using dynamic programming**.

#include<bits/stdc++.h>

using namespace std;

int minCoins(int coins[], int m, int V)

{

if (V == 0) return 0;

int res = INT\_MAX;

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

{

if (coins[i] <= V)

{

int sub\_res = minCoins(coins, m, V-coins[i]);

if (sub\_res != INT\_MAX && sub\_res + 1 < res)

res = sub\_res + 1;

}

}

return res;

}

int main()

{

int coins[] = {9, 6, 5, 1};

int m = sizeof(coins)/sizeof(coins[0]);

int V;

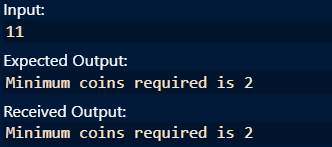
cin>>V;

cout << "Minimum coins required is "

<< minCoins(coins, m, V);

return 0;

}



**10.Write a program to implement the knapsack problem using dynamic programming.**

#include <bits/stdc++.h>

using namespace std;

int knapsackUtil(vector<int>& wt, vector<int>& val, int ind, int W, vector<vector<int>>& dp){

if(ind == 0){

if(wt[0] <=W) return val[0];

else return 0;

}

if(dp[ind][W]!=-1)

return dp[ind][W];

int notTaken = 0 + knapsackUtil(wt,val,ind-1,W,dp);

int taken = INT\_MIN;

if(wt[ind] <= W)

taken = val[ind] + knapsackUtil(wt,val,ind-1,W-wt[ind],dp);

return dp[ind][W] = max(notTaken,taken);

}

int knapsack(vector<int>& wt, vector<int>& val, int n, int W){

vector<vector<int>> dp(n,vector<int>(W+1,-1));

return knapsackUtil(wt, val, n-1, W, dp);

}

int main() {

vector<int> wt = {1,2,4,5};

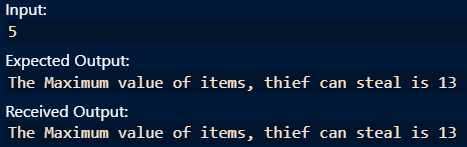
vector<int> val = {5,4,8,6};

int W=5;

int n = wt.size();

cout<<"The Maximum value of items, thief can steal is " <<knapsack(wt,val,n,W);

}



**11. Write a program to implement Floyd’s algorithm for finding shortest path using dynamic programming.**

#include <bits/stdc++.h>

using namespace std;

#define V 4

#define INF 99999

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

void floydWarshall(int graph[][V])

{

int dist[V][V], i, j, k;

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

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

dist[i][j] = graph[i][j];

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))

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, 5, INF, 10 },

{ INF, 0, 3, INF },

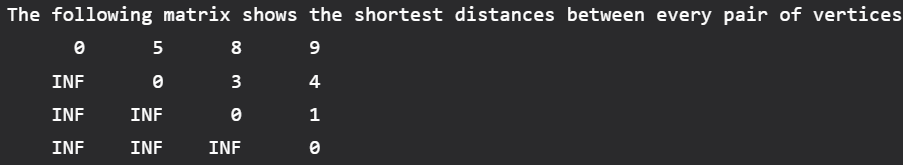
{ INF, INF, 0, 1 },

{ INF, INF, INF, 0 } };

floydWarshall(graph);

return 0;

}



**12.Write a program to implement chained matrix multiplication using dynamic programming.**

#include <bits/stdc++.h>

using namespace std;

int MatrixChainOrder(int p[], int i, int j)

{

if (i == j)

return 0;

int k;

int mini = 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];

mini = min(count, mini);

}

return mini;

}

int main()

{

int arr[] = { 1, 2, 3, 4, 3 };

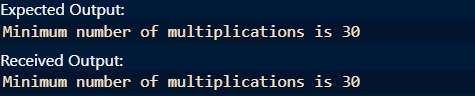
int N = sizeof(arr) / sizeof(arr[0]);

cout << "Minimum number of multiplications is "

<< MatrixChainOrder(arr, 1, N - 1);

return 0;

}



**13. Write a program to implement longest common subsequence using dynamic programming.**

#include <bits/stdc++.h>

using namespace std;

int main()

{

string s1, s2;

cin >> s1 >> s2;

int dp[s1.length() + 1][s2.length() + 1];

int n = s1.length(), m = s2.length();

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

{

for (int j = 0; j <= m; j++)

if (i == 0 || j == 0)

dp[i][j] = 0;

else

{

if (s1[i - 1] == s2[j - 1])

dp[i][j] = dp[i - 1][j - 1] + 1;

else

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);

}

}

cout << dp[n][m] << "\n";

return 0;

}

