Netaji Subhas University of Technology

(Main Campus, Dwarka, Delhi)

Department of Computer Science and Engineering

DESIGN AND ANALYSIS OF ALGORITHMS (DAA) - LAB PROJECT FILE

Submitted By:
Name: Nishant Yadav
Roll No.: 2024UCS1710
Class: CSE-3
Branch: Computer Science and Engineering (2024–2028)

Submitted To:
Test
Assistant Professor, CSE Department

Academic Session: 2025–26

INDEX

Experiment 1: Bubble, Insertion, Selection Sort

Aim: To implement and compare Bubble, Insertion and Selection Sort algorithms.

Sample Input/Output:
Input: 5
Elements: 4 1 3 9 7
Output (sorted): 1 3 4 7 9

Complexity: Best: O(n) (for Bubble/Insertion), Average/Worst: O(n^2). Space: O(1).

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

void bubbleSort(vector<int> a) {
    int n = a.size();
    for (int i = 0; i < n-1; ++i)
        for (int j = 0; j < n-i-1; ++j)
            if (a[j] > a[j+1]) swap(a[j], a[j+1]);
    for (int x : a) cout << x << " ";
    cout << "\n";
}

void insertionSort(vector<int> a) {
    int n = a.size();
    for (int i = 1; i < n; ++i) {
        int key = a[i], j = i-1;
        while (j >= 0 && a[j] > key) { a[j+1] = a[j]; --j; }
        a[j+1] = key;
    }
    for (int x : a) cout << x << " ";
    cout << "\n";
}

void selectionSort(vector<int> a) {
    int n = a.size();
    for (int i = 0; i < n-1; ++i) {
        int minIdx = i;
        for (int j = i+1; j < n; ++j) if (a[j] < a[minIdx]) minIdx = j;
```

```
        swap(a[i], a[minIdx]);
    }
    for (int x : a) cout << x << " ";
    cout << "\n";
}

int main() {
    int n; cout << "Enter number of elements: "; cin >> n;
    vector<int> arr(n);
    cout << "Enter elements: ";
    for (int i=0;i<n;++i) cin >> arr[i];
    while (true) {
        cout << "\n1. Bubble  2. Insertion  3. Selection  4. Exit\nChoice: ";
        int ch; cin >> ch; if (ch==4) break;
        if (ch==1) bubbleSort(arr);
        else if (ch==2) insertionSort(arr);
        else if (ch==3) selectionSort(arr);
        else cout << "Invalid choice\\n";
    }
    return 0;
}
```

Output :

```
D:\Anand\loc\College\Sem 3\DAA\LAB>.\a.exe
Enter number of elements: 5
Enter elements: 5 4 3 2 1

1. Bubble  2. Insertion  3. Selection  4. Exit
Choice: 1
1 2 3 4 5

1. Bubble  2. Insertion  3. Selection  4. Exit
Choice: 2
1 2 3 4 5

1. Bubble  2. Insertion  3. Selection  4. Exit
Choice: 3
1 2 3 4 5

1. Bubble  2. Insertion  3. Selection  4. Exit
Choice: 4

D:\Anand\loc\College\Sem 3\DAA\LAB>
```

Experiment 2: Merge Sort & Quick Sort

Aim: To implement Merge Sort and Quick Sort using divide-and-conquer.

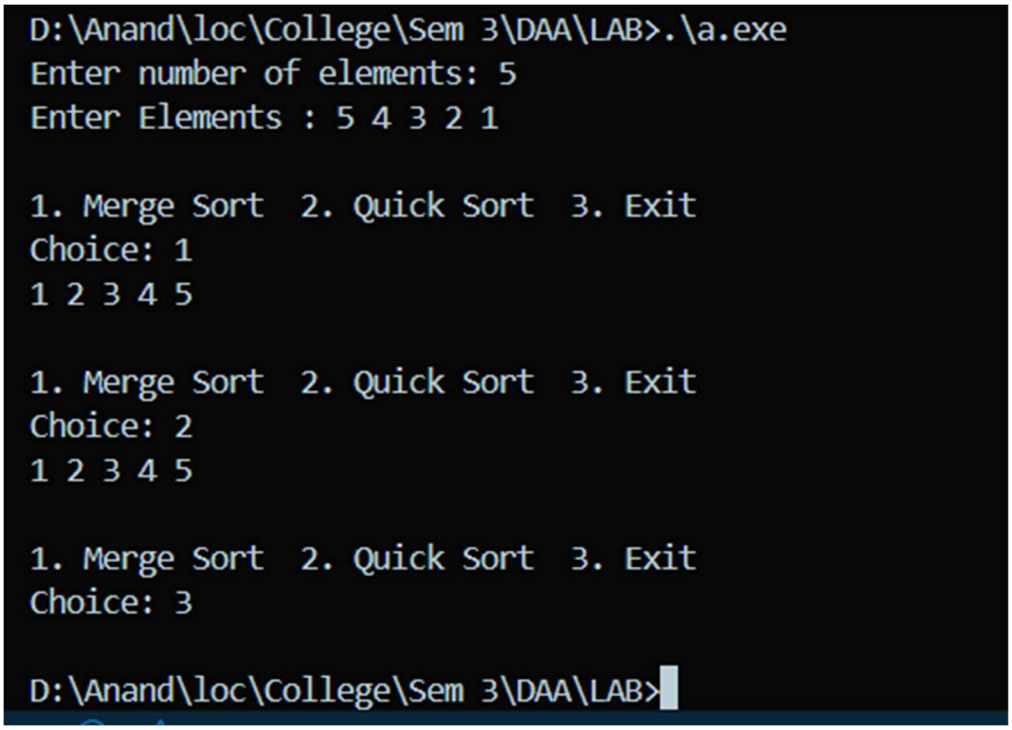
Sample Input/Output:
Input: 6
Elements: 8 4 5 3 2 7
Output (sorted): 2 3 4 5 7 8

Complexity: Merge Sort: O(n log n) always, Space: O(n). Quick Sort: Avg O(n log n), Worst O(n^2), Space: O(log n).

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

void mergeVec(vector<int>& a, int l, int m, int r) {
    int n1 = m-l+1, n2 = r-m;
    vector<int> L(n1), R(n2);
    for(int i=0;i<n1;++i) L[i]=a[l+i];
    for(int j=0;j<n2;++j) R[j]=a[m+1+j];
    int i=0,j=0,k=l;
    while(i<n1 && j<n2) a[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    while(i<n1) a[k++]=L[i++];
    while(j<n2) a[k++]=R[j++];
}

void mergeSort(vector<int>& a,int l,int r){
    if(l<r){
        int m = l + (r-l)/2;
        mergeSort(a,l,m);
        mergeSort(a,m+1,r);
        mergeVec(a,l,m,r);
    }
}

int partition(vector<int>& a,int low,int high){
    int pivot = a[high], i = low-1;
    for(int j=low;j<high;++j) if(a[j] < pivot) swap(a[++i], a[j]);
    swap(a[i+1], a[high]);
```

```
    return i+1;
}

void quickSort(vector<int>& a,int low,int high){
    if(low < high){
        int pi = partition(a, low, high);
        quickSort(a, low, pi-1);
        quickSort(a, pi+1, high);
    }
}

int main(){
    int n; cout<<"Enter number of elements: "; cin>>n;
    vector<int> arr(n); for(int i=0;i<n;++i) cin>>arr[i];
    while(true){
        cout<<"\\n1. Merge Sort  2. Quick Sort  3. Exit\\nChoice: ";
        int ch; cin>>ch; if(ch==3) break;
        vector<int> tmp = arr;
        if(ch==1) mergeSort(tmp,0,n-1);
        else if(ch==2) quickSort(tmp,0,n-1);
        else { cout<<"Invalid\\n"; continue; }
        for(int x: tmp) cout<<x<<" ";
        cout<<"\n";
    }
    return 0;
}
```

Output :


D:\Anand\loc\College\Sem 3\DAA\LAB>.\a.exe
Enter number of elements: 5
Enter Elements : 5 4 3 2 1
1. Merge Sort 2. Quick Sort 3. Exit
Choice: 1
1 2 3 4 5
1. Merge Sort 2. Quick Sort 3. Exit
Choice: 2
1 2 3 4 5
1. Merge Sort 2. Quick Sort 3. Exit
Choice: 3
D:\Anand\loc\College\Sem 3\DAA\LAB>

Experiment 3: Linear Search & Binary Search

Aim: To implement linear search and binary search on arrays.

Sample Input/Output:
Input: 5
Elements: 1 3 5 7 9
Key: 7
Output: Found at index 3 (0-based)

Complexity: Linear Search: O(n). Binary Search: O(log n). Space: O(1).

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

int linearSearch(const vector<int>& a, int key){
    for(int i=0;i<a.size();++i) if(a[i]==key) return i;
    return -1;
}

int binarySearch(const vector<int>& a, int key){
    int l=0, r=a.size()-1;
    while(l<=r){
        int m = l + (r-l)/2;
        if(a[m]==key) return m;
        else if(a[m]<key) l = m+1;
        else r = m-1;
    }
    return -1;
}

int main(){
    int n; cout<<"Enter number of elements: "; cin>>n;
    vector<int> arr(n); for(int i=0;i<n;++i) cin>>arr[i];
    while(true){
        cout<<"\\n1. Linear Search  2. Binary Search  3. Exit\\nChoice: ";
        int ch; cin>>ch; if(ch==3) break;
        cout<<"Enter key: "; int key; cin>>key;
```

```
        int idx = (ch==1)? linearSearch(arr,key) : (ch==2)? binarySearch(arr,key)
: -2;
        if(idx==-2) cout<<"Invalid choice\\n";
        else if(idx==-1) cout<<"Not found\\n";
        else cout<<"Found at index: "<<idx<<"\\n";
    }
    return 0;
}
```

Output :

```
D:\Anand\loc\College\Sem 3\DAA\LAB>.\a.exe
Enter number of elements: 5
Enter Elements : 5 4 3 2 1

1. Linear Search  2. Binary Search  3. Exit
Choice: 1
Enter key: 3
Found at index: 2

1. Linear Search  2. Binary Search  3. Exit
Choice: 2
Enter key: 3
Found at index: 2

1. Linear Search  2. Binary Search  3. Exit
Choice: 3

D:\Anand\loc\College\Sem 3\DAA\LAB>
```

Experiment 4: Binary Search Tree (Insert/Delete/Search)

Aim: To implement BST insertion, deletion, searching and inorder traversal.

Sample Input/Output:
Operations: Insert 5, Insert 2, Insert 8, Inorder -> 2 5 8

Complexity: Average: O(log n). Worst: O(n) (skewed). Space: O(n).

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node
{
    int val;
    Node *left;
    Node *right;
    Node(int v) : val(v), left(nullptr), right(nullptr) {}
};

Node *insertNode(Node *root, int key)
{
    if (!root)
        return new Node(key);
    if (key < root->val)
        root->left = insertNode(root->left, key);
    else if (key > root->val)
        root->right = insertNode(root->right, key);
    return root;
}

Node *minValueNode(Node *node)
{
    Node *cur = node;
    while (cur && cur->left)
        cur = cur->left;
    return cur;
}

Node *deleteNode(Node *root, int key)
```

```
{
    if (!root)
        return root;
    if (key < root->val)
        root->left = deleteNode(root->left, key);
    else if (key > root->val)
        root->right = deleteNode(root->right, key);
    else
    {
        if (!root->left)
        {
            Node *t = root->right;
            delete root;
            return t;
        }
        else if (!root->right)
        {
            Node *t = root->left;
            delete root;
            return t;
        }
        Node *temp = minValueNode(root->right);
        root->val = temp->val;
        root->right = deleteNode(root->right, temp->val);
    }
    return root;
}

Node *searchNode(Node *root, int key)
{
    if (!root || root->val == key)
        return root;
    if (key < root->val)
        return searchNode(root->left, key);
    return searchNode(root->right, key);
}

void inorder(Node *root)
{
    if (!root)
        return;
    inorder(root->left);
    cout << root->val << " ";
    inorder(root->right);
}
```

```
int main()
{
    Node *root = nullptr;
    while (true)
    {
        cout << "\\n1.Insert 2.Delete 3.Search 4.Inorder 5.Exit\\nChoice: ";
        int ch;
        cin >> ch;
        if (ch == 5)
            break;
        int x;
        if (ch == 1)
        {
            cout << "Value: ";
            cin >> x;
            root = insertNode(root, x);
        }
        else if (ch == 2)
        {
            cout << "Value to delete: ";
            cin >> x;
            root = deleteNode(root, x);
        }
        else if (ch == 3)
        {
            cout << "Value to search: ";
            cin >> x;
            cout << (searchNode(root, x) ? "Found\\n" : "Not found\\n");
        }
        else if (ch == 4)
        {
            cout << "Inorder: ";
            inorder(root);
            cout << "\\n";
        }
        else
            cout << "Invalid\\n";
    }
    return 0;
}
```

Output :

```
D:\Anand\loc\College\Sem 3\DAA\LAB>.\a.exe

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 1
Value: 5

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 1
Value: 6

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 1
Value: 3

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 2
Value to delete: 5

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 4
Inorder: 3 6

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 5
```

Experiment 5: AVL Tree (Insert/Delete/Search)

Aim: To implement AVL tree with rotations to maintain balance.

Sample Input/Output:
Insert: 30 20 40 10 25 -> Inorder: 10 20 25 30 40

Complexity: Search/Insert/Delete: O(log n). Space: O(n).

```
#include <bits/stdc++.h>
using namespace std;
struct Node
{
    int key, height;
    Node *left;
    Node *right;
    Node(int k) : key(k), height(1), left(nullptr), right(nullptr) {}
};
int height(Node *n) { return n ? n->height : 0; }
int getBalance(Node *n) { return n ? height(n->left) - height(n->right) : 0; }
Node *rightRotate(Node *y)
{
    Node *x = y->left;
    Node *T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = 1 + max(height(y->left), height(y->right));
    x->height = 1 + max(height(x->left), height(x->right));
    return x;
}
Node *leftRotate(Node *x)
{
    Node *y = x->right;
    Node *T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = 1 + max(height(x->left), height(x->right));
```

```
    y->height = 1 + max(height(y->left), height(y->right));
    return y;
}
Node *insertNode(Node *node, int key)
{
    if (!node)
        return new Node(key);
    if (key < node->key)
        node->left = insertNode(node->left, key);
    else if (key > node->key)
        node->right = insertNode(node->right, key);
    else
        return node;
    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);
    if (balance < -1 && key > node->right->key)
        return leftRotate(node);
    if (balance > 1 && key > node->left->key)
    {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->key)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
void inorder(Node *r)
{
    if (!r)
        return;
    inorder(r->left);
    cout << r->key << " ";
    inorder(r->right);
}
int main()
{
    Node *root = nullptr;
    int ch, x;
    while (true)
    {
```

```
        cout << "\\n1.Insert 2.Delete 3.Search 4.Inorder 5.Exit\\nChoice: ";
        cin >> ch;
        if (ch == 5)
            break;
        if (ch == 1)
        {
            cout << "Value: ";
            cin >> x;
            root = insertNode(root, x);
        }
        else if (ch == 4)
        {
            cout << "Inorder: ";
            inorder(root);
            cout << "\\n";
        }
        else
            cout << "(For brevity, full delete/search options not shown in this
sample)\\n";
    }
    return 0;
}
```

Output :

```
D:\Anand\loc\College\Sem 3\DAA\LAB>.\a.exe

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 1
Value: 2

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 1
Value: 5

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 1
Value: 3

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 2
(For brevity, full delete/search options not shown in this sample)

1.Insert 2.Delete 3.Search 4.Inorder 5.Exit
Choice: 5

D:\Anand\loc\College\Sem 3\DAA\LAB>
```

Experiment 6: Red-Black Tree (Insertion & Search)

Aim: To implement Red-Black Tree insertion and basic properties demonstration.

Sample Input/Output:
Insert: 10 20 30 -> Inorder: 10(R/B) 20(R/B) 30(R/B)

Complexity: Search/Insert/Delete: O(log n). Space: O(n).

```
#include <bits/stdc++.h>
using namespace std;
enum Color
{
    RED,
    BLACK
};
struct Node
{
    int key;
    Color color;
    Node *left, *right, *parent;
    Node(int k) : key(k), color(RED), left(nullptr), right(nullptr),
parent(nullptr) {}
};
class RBTree
{
    Node *root;
    Node *TNULL;

public:
    RBTree()
    {
        TNULL = new Node(0);
        TNULL->color = BLACK;
        TNULL->left = TNULL->right = TNULL->parent = nullptr;
        root = TNULL;
    }
    void leftRotate(Node *x)
    {
```

```
    Node *y = x->right;
    x->right = y->left;
    if (y->left != TNULL)
        y->left->parent = x;
    y->parent = x->parent;
    if (x->parent == nullptr)
        root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;
}
void rightRotate(Node *x)
{
    Node *y = x->left;
    x->left = y->right;
    if (y->right != TNULL)
        y->right->parent = x;
    y->parent = x->parent;
    if (x->parent == nullptr)
        root = y;
    else if (x == x->parent->right)
        x->parent->right = y;
    else
        x->parent->left = y;
    y->right = x;
    x->parent = y;
}
void insertFix(Node *k)
{
    while (k->parent && k->parent->color == RED)
    {
        if (k->parent == k->parent->parent->left)
        {
            Node *u = k->parent->parent->right;
            if (u->color == RED)
            {
                k->parent->color = BLACK;
                u->color = BLACK;
                k->parent->parent->color = RED;
                k = k->parent->parent;
            }
            else
```

```
            {
                if (k == k->parent->right)
                {
                    k = k->parent;
                    leftRotate(k);
                }
                k->parent->color = BLACK;
                k->parent->parent->color = RED;
                rightRotate(k->parent->parent);
            }
        }
        else
        {
            Node *u = k->parent->parent->left;
            if (u->color == RED)
            {
                k->parent->color = BLACK;
                u->color = BLACK;
                k->parent->parent->color = RED;
                k = k->parent->parent;
            }
            else
            {
                if (k == k->parent->left)
                {
                    k = k->parent;
                    rightRotate(k);
                }
                k->parent->color = BLACK;
                k->parent->parent->color = RED;
                leftRotate(k->parent->parent);
            }
        }
        if (k == root)
            break;
    }
    root->color = BLACK;
}
void insert(int key)
{
    Node *node = new Node(key);
    node->left = node->right = node->parent = TNULL;
    Node *y = nullptr;
    Node *x = this->root;
    while (x != TNULL)
```

```
        {
            y = x;
            if (node->key < x->key)
                x = x->left;
            else
                x = x->right;
        }
        node->parent = y;
        if (y == nullptr)
            root = node;
        else if (node->key < y->key)
            y->left = node;
        else
            y->right = node;
        if (node->parent == nullptr)
        {
            node->color = BLACK;
            return;
        }
        if (node->parent->parent == nullptr)
            return;
        insertFix(node);
    }
    Node *search(Node *node, int key)
    {
        if (node == TNULL || key == node->key)
            return node;
        if (key < node->key)
            return search(node->left, key);
        return search(node->right, key);
    }
    void inorder(Node *node)
    {
        if (node != TNULL)
        {
            inorder(node->left);
            cout << node->key <<\"(\"<<(node->color==RED?\"R\":\"B\")<<\") \";
inorder(node->right); } }
                                    Node *
                                    getRoot()
            {
                return root;
            }
            Node *getTNULL() { return TNULL; }
        };
```

```
        int main()
        {
            RBTree tree;
            while (true)
            {
                cout << "\\n1.Insert 2.Search 3.Inorder 4.Exit\\nChoice: ";
                int ch;
                cin >> ch;
                if (ch == 4)
                    break;
                if (ch == 1)
                {
                    cout << "Value: ";
                    int v;
                    cin >> v;
                    tree.insert(v);
                }
                else if (ch == 2)
                {
                    cout << "Value: ";
                    int v;
                    cin >> v;
                    auto n = tree.search(tree.getRoot(), v);
cout<<(n!=tree.getTNULL()?\"Found\\n\":\"Not found\\n\"); }
        else if(ch==3){
                        cout << "Inorder: ";
                        tree.inorder(tree.getRoot());
                        cout << "\\n"; }
        else cout<<"Invalid\\n";
                }
                return 0;
            }
```

Experiment 7: Graph Traversal (BFS & DFS)

Aim: To implement BFS and DFS on a directed/undirected graph using adjacency list.

Sample Input/Output:
Vertices: 5, Edges: (0,1),(0,2),(1,3)
BFS from 0: 0 1 2 3

Complexity: BFS/DFS: O(V+E). Space: O(V).

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

void BFS(int s, vector<vector<int>>& adj){
    int n = adj.size();
    vector<bool> vis(n,false);
    queue<int> q; q.push(s); vis[s]=true;
    while(!q.empty()){
        int u=q.front(); q.pop(); cout<<u<<" ";
        for(int v: adj[u]) if(!vis[v]){ vis[v]=true; q.push(v); }
    }
    cout<<"\n";
}

void DFSUtil(int u, vector<bool>& vis, vector<vector<int>>& adj){
    vis[u]=true; cout<<u<<" ";
    for(int v: adj[u]) if(!vis[v]) DFSUtil(v, vis, adj);
}

void DFS(int s, vector<vector<int>>& adj){
    vector<bool> vis(adj.size(), false);
    DFSUtil(s, vis, adj);
    cout<<"\n";
}

int main(){
    int n,m; cout<<"Enter number of vertices and edges: "; cin>>n>>m;
    vector<vector<int>> adj(n);
    cout<<"Enter edges (u v):\n";
```

```
    for(int i=0;i<m;++i){ int u,v; cin>>u>>v; adj[u].push_back(v); }
    while(true){
        cout<<"\\n1.BFS 2.DFS 3.Exit\\nChoice: ";
        int ch; cin>>ch; if(ch==3) break;
        cout<<"Start vertex: "; int s; cin>>s;
        if(ch==1) BFS(s,adj); else if(ch==2) DFS(s,adj); else cout<<"Invalid\\n";
    }
    return 0;
}
```

Experiment 8: Bucket Sort & Radix Sort

Aim: To implement Bucket Sort and Radix Sort for integers.

Sample Input/Output:
Input: 6
Elements: 170 45 75 90 802 24
Output: 24 45 75 90 170 802

Complexity: Bucket Sort: O(n + k). Radix Sort: O(d*(n+k)). Space: O(n+k).

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

// ---------- Bucket Sort ----------
void bucketSort(vector<int> &a)
{
    if (a.empty()) return;

    int n = a.size();
    int maxv = *max_element(a.begin(), a.end());
    int minv = *min_element(a.begin(), a.end());
    int range = maxv - minv + 1;

    // Number of buckets (choose based on size)
    int bucketCount = max(1, n / 5);
    vector<vector<int>> buckets(bucketCount);

    // Distribute elements into buckets
    for (int x : a)
    {
        int idx = (int)((long long)(x - minv) * bucketCount / range);
        if (idx == bucketCount) idx--; // Edge case for max element
        buckets[idx].push_back(x);
    }

    // Sort individual buckets
    for (auto &b : buckets)
        sort(b.begin(), b.end());
```

```
    // Concatenate buckets
    a.clear();
    for (auto &b : buckets)
        for (int x : b)
            a.push_back(x);
}

// ---------- Radix Sort (for non-negative integers) ----------
void radixSort(vector<int> &a)
{
    if (a.empty()) return;

    int maxv = *max_element(a.begin(), a.end());
    for (int exp = 1; maxv / exp > 0; exp *= 10)
    {
        vector<int> output(a.size());
        vector<int> cnt(10, 0);

        for (int x : a)
            cnt[(x / exp) % 10]++;

        for (int i = 1; i < 10; ++i)
            cnt[i] += cnt[i - 1];

        for (int i = (int)a.size() - 1; i >= 0; --i)
        {
            int digit = (a[i] / exp) % 10;
            output[--cnt[digit]] = a[i];
        }
        a = output;
    }
}

// ---------- Main Menu ----------
int main()
{
    int n;
    cout << "Enter number of non-negative integers: ";
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; ++i)
        cin >> arr[i];

    while (true)
```

```
    {
        cout << "\n1. Bucket Sort  2. Radix Sort  3. Exit\nChoice: ";
        int ch;
        cin >> ch;
        if (ch == 3)
            break;

        vector<int> tmp = arr;
        if (ch == 1)
            bucketSort(tmp);
        else if (ch == 2)
            radixSort(tmp);
        else
        {
            cout << "Invalid\n";
            continue;
        }

        cout << "Sorted: ";
        for (int x : tmp)
            cout << x << " ";
        cout << "\n";
    }
    return 0;
}
```

Experiment 9: Topological Sort & 0/1 Knapsack (DP)

Aim: To implement Kahn's topological sort and DP solution for 0/1 Knapsack.

Sample Input/Output:
Topo input: DAG vertices 4 edges: (0,1),(0,2),(1,3)
Knapsack sample: n=3 wt:1 3 4 val:15 50 60 W=5 -> Max value 65

Complexity: Topological : O(V+E). Knapsack DP: O(n*W). Space: O(n*W).

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

// ---------- Topological Sort using DFS ----------
void dfsTopo(int u, vector<vector<int>>& adj, vector<int>& vis, stack<int>& st) {
    vis[u] = 1;
    for (int v : adj[u]) {
        if (!vis[v]) dfsTopo(v, adj, vis, st);
    }
    st.push(u);
}

void topoDFS(int n, vector<vector<int>>& adj) {
    vector<int> vis(n, 0);
    stack<int> st;

    for (int i = 0; i < n; ++i) {
        if (!vis[i]) dfsTopo(i, adj, vis, st);
    }

    vector<int> topo;
    while (!st.empty()) {
        topo.push_back(st.top());
        st.pop();
    }

    cout << "Topological order (DFS-based): ";
    for (int x : topo) cout << x << " ";
    cout << "\n";
```

```
}

// ---------- 0/1 Knapsack ----------
int knapSack(int W, vector<int>& wt, vector<int>& val) {
    int n = wt.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= n; ++i)
        for (int w = 0; w <= W; ++w)
            if (wt[i - 1] <= w)
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - wt[i - 1]] + val[i -
1]);
            else
                dp[i][w] = dp[i - 1][w];

    return dp[n][W];
}

// ---------- Main Menu ----------
int main() {
    while (true) {
        cout << "\n1. Topological Sort (DFS)  2. Knapsack  3. Exit\nChoice: ";
        int ch;
        cin >> ch;
        if (ch == 3) break;

        if (ch == 1) {
            int n, m;
            cout << "Vertices and edges: ";
            cin >> n >> m;
            vector<vector<int>> adj(n);
            cout << "Enter edges (u v):\n";
            for (int i = 0; i < m; ++i) {
                int u, v;
                cin >> u >> v;
                adj[u].push_back(v);
            }
            topoDFS(n, adj);
        }
        else if (ch == 2) {
            int n, W;
            cout << "Number of items: ";
            cin >> n;
            vector<int> wt(n), val(n);
            cout << "Enter weights:\n";
```

```
            for (int i = 0; i < n; ++i) cin >> wt[i];
            cout << "Enter values:\n";
            for (int i = 0; i < n; ++i) cin >> val[i];
            cout << "Enter capacity: ";
            cin >> W;
            cout << "Max value: " << knapSack(W, wt, val) << "\n";
        }
        else cout << "Invalid choice\n";
    }
    return 0;
}
```

Conclusion

This lab report includes implementations and analysis of major algorithms covered under DAA. The experiments demonstrate algorithmic techniques and efficiency trade-offs.

Acknowledgment

I sincerely thank my faculty for their guidance and continuous support throughout this lab work.