Linear and binary

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int linearSearch(const vector<int>& arr, int target) {

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

if (arr[i] == target) {

return i;

}

}

return -1;

}

int binarySearch(const vector<int>& arr, int target) {

int left = 0;

int right = arr.size() - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target) {

return mid;

} else if (arr[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

int main() {

int n, target;

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

cin >> n;

vector<int> arr(n);

cout << "Enter the elements of the array:\n";

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

cin >> arr[i];

}

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

cin >> target;

int linearResult = linearSearch(arr, target);

if (linearResult != -1) {

cout << "Element found at index " << linearResult << " using Linear Search.\n";

} else {

cout << "Element not found using Linear Search.\n";

}

sort(arr.begin(), arr.end());

int binaryResult = binarySearch(arr, target);

if (binaryResult != -1) {

cout << "Element found at index " << binaryResult << " using Binary Search (after sorting).\n";

} else {

cout << "Element not found using Binary Search.\n";

}

return 0;

}

Pseudocode Selection Sort

function selectionSort(array):

n = length(array)

for i from 0 to n - 1:

minIndex = i

for j from i + 1 to n - 1:

if array[j] < array[minIndex]:

minIndex = j

swap(array[i], array[minIndex])

return array

Postfix

#include <iostream>

#include <stack>

#include <string>

using namespace std;

bool isOperator(char c) {

return c == '+' || c == '-' || c == '\*' || c == '/';

}

int precedence(char op) {

if (op == '+' || op == '-') {

return 1;

} else if (op == '\*' || op == '/') {

return 2;

}

return 0;

}

string infixToPostfix(const string& infix) {

stack<char> s;

string postfix;

for (char c : infix) {

if (isalnum(c)) {

postfix += c;

} else if (c == '(') {

s.push(c);

} else if (c == ')') {

while (!s.empty() && s.top() != '(') {

postfix += s.top();

s.pop();

}

s.pop();

} else if (isOperator(c)) {

while (!s.empty() && precedence(s.top()) >= precedence(c)) {

postfix += s.top();

s.pop();

}

s.push(c);

}

}

while (!s.empty()) {

postfix += s.top();

s.pop();

}

return postfix;

}

int main() {

string infix;

cout << "Enter an infix expression: ";

cin >> infix;

string postfix = infixToPostfix(infix);

cout << "Postfix expression: " << postfix << endl;

return 0;

}

Postorder

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

Node\* insert(Node\* root, int data) {

if (root == nullptr) {

return new Node(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

}

else {

root->right = insert(root->right, data);

}

return root;

}

void postorderTraversal(Node\* root) {

if (root == nullptr) {

return;

}

postorderTraversal(root->left);

postorderTraversal(root->right);

cout << root->data << " ";

}

int main() {

Node\* root = nullptr;

int n, value;

cout << "Enter the number of elements you want to insert in the BST: ";

cin >> n;

cout << "Enter the elements to insert into the BST:\n";

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

cin >> value;

root = insert(root, value);

}

cout << "Postorder traversal of the BST: ";

postorderTraversal(root);

cout << endl;

return 0;

}

Prefix

#include <iostream>

#include <stack>

#include <string>

#include <algorithm>

using namespace std;

bool isOperator(char c) {

return c == '+' || c == '-' || c == '\*' || c == '/';

}

int precedence(char op) {

if (op == '+' || op == '-') {

return 1;

} else if (op == '\*' || op == '/') {

return 2;

}

return 0;

}

string infixToPostfix(const string& infix) {

stack<char> s;

string postfix;

for (char c : infix) {

if (isalnum(c)) {

postfix += c;

} else if (c == '(') {

s.push(c);

} else if (c == ')') {

while (!s.empty() && s.top() != '(') {

postfix += s.top();

s.pop();

}

s.pop();

} else if (isOperator(c)) {

while (!s.empty() && precedence(s.top()) >= precedence(c)) {

postfix += s.top();

s.pop();

}

s.push(c);

}

}

while (!s.empty()) {

postfix += s.top();

s.pop();

}

return postfix;

}

string infixToPrefix(string infix) {

reverse(infix.begin(), infix.end());

for (char& c : infix) {

if (c == '(') c = ')';

else if (c == ')') c = '(';

}

string postfix = infixToPostfix(infix);

reverse(postfix.begin(), postfix.end());

return postfix;

}

int main() {

string infix;

cout << "Enter an infix expression: ";

cin >> infix;

string prefix = infixToPrefix(infix);

cout << "Prefix expression: " << prefix << endl;

return 0;

}

Push, pop stack

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

class Stack {

Node\* top;

public:

Stack() : top(nullptr) {}

void push(int data) {

Node\* newNode = new Node(data);

newNode->next = top;

top = newNode;

cout << data << " pushed onto the stack.\n";

}

void pop() {

if (top == nullptr) {

cout << "Stack underflow. Nothing to pop.\n";

return;

}

Node\* temp = top;

top = top->next;

cout << temp->data << " popped from the stack.\n";

delete temp;

}

void display() const {

if (top == nullptr) {

cout << "The stack is empty.\n";

return;

}

Node\* current = top;

cout << "Stack elements: ";

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

~Stack() {

while (top != nullptr) {

Node\* temp = top;

top = top->next;

delete temp;

}

}

};

int main() {

Stack stack;

stack.push(10);

stack.push(20);

stack.push(30);

stack.display();

stack.pop();

stack.display();

stack.pop();

stack.display();

stack.pop();

stack.display();

return 0;

}

Preorder travel

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

Node\* insert(Node\* root, int data) {

if (root == nullptr) {

return new Node(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

}

else {

root->right = insert(root->right, data);

}

return root;

void preorderTraversal(Node\* root) {

if (root == nullptr) {

return;

}

cout << root->data << " ";

preorderTraversal(root->left);

preorderTraversal(root->right);

}

int main() {

Node\* root = nullptr;

int n, value;

cout << "Enter the number of elements you want to insert in the BST: ";

cin >> n;

cout << "Enter the elements to insert into the BST:\n";

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

cin >> value;

root = insert(root, value);

}

cout << "Preorder traversal of the BST: ";

preorderTraversal(root);

cout << endl;

return 0;

}

Quick sort

#include <iostream>

using namespace std;

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] < pivot) {

i++;

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

}

}

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

return i + 1;

}

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);

}

}

void displayArray(int arr[], int n) {

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

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

}

cout << endl;

}

int main() {

int n;

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

cin >> n;

int arr[n];

cout << "Enter the elements of the array:\n";

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

cin >> arr[i];

}

cout << "Original array: ";

displayArray(arr, n);

quickSort(arr, 0, n - 1);

cout << "Sorted array: ";

displayArray(arr, n);

return 0;

}

Pseudo code for bubble sort:

Initialize n = Length of Array

BubbleSort(Array, n)

{

for i = 0 to n-2

{

for j = 0 to n-2

{

if Array[j] > Array[j+1]

{

swap(Array[j], Array[j+1])

}

}

}

}

Bubble sort

#include <iostream>

using namespace std;

void bubbleSort(int arr[], int n) {

bool swapped;

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

swapped = false;

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

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

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

swapped = true;

}

}

if (!swapped) {

break;

}

}

}

void displayArray(int arr[], int n) {

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

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

}

cout << endl;

}

int main() {

int n;

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

cin >> n;

int arr[n];

cout << "Enter the elements of the array:\n";

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

cin >> arr[i];

}

cout << "Original array: ";

displayArray(arr, n);

bubbleSort(arr, n);

cout << "Sorted array: ";

displayArray(arr, n);

return 0;

}

Insertion sort

#include <iostream>

using namespace std;

void insertionSort(int arr[], int n) {

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

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

j = j - 1;

}

arr[j + 1] = key;

}

}

void displayArray(int arr[], int n) {

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

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

}

cout << endl;

}

int main() {

int n;

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

cin >> n;

int arr[n];

cout << "Enter the elements of the array:\n";

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

cin >> arr[i];

}

cout << "Original array: ";

displayArray(arr, n);

insertionSort(arr, n);

cout << "Sorted array: ";

displayArray(arr, n);

return 0;

}

Pseudocode for Insertion Sort

function insertionSort(array):

n = length(array)

for i from 1 to n - 1:

key = array[i]

j = i - 1

while j >= 0 and array[j] > key:

array[j + 1] = array[j]

j = j - 1

array[j + 1] = key

return array

Selection sort

#include <iostream>

using namespace std;

void selectionSort(int arr[], int n) {

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

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

if (minIndex != i) {

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

}

}

}

void displayArray(int arr[], int n) {

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

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

}

cout << endl;

}

int main() {

int n;

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

cin >> n;

int arr[n];

cout << "Enter the elements of the array:\n";

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

cin >> arr[i];

}

cout << "Original array: ";

displayArray(arr, n);

selectionSort(arr, n);

cout << "Sorted array: ";

displayArray(arr, n);

return 0;

}

Pseudocode Quick Sort

function quickSort(array, low, high):

if low < high:

pivotIndex = partition(array, low, high)

quickSort(array, low, pivotIndex - 1)

quickSort(array, pivotIndex + 1, high)

function partition(array, low, high):

pivot = array[high]

i = low - 1

for j from low to high - 1:

if array[j] < pivot:

i = i + 1

swap(array[i], array[j])

swap(array[i + 1], array[high])

return i + 1

Pseudo code merge sort

function mergeSort(array):

if length(array) <= 1:

return array

mid = length(array) / 2

left = mergeSort(array[0:mid])

right = mergeSort(array[mid:length(array)])

return merge(left, right)

function merge(left, right):

sortedArray = []

i = 0

j = 0

while i < length(left) and j < length(right):

if left[i] < right[j]:

append left[i] to sortedArray

i = i + 1

else:

append right[j] to sortedArray

j = j + 1

while i < length(left):

append left[i] to sortedArray

i = i + 1

while j < length(right):

append right[j] to sortedArray

j = j + 1

return sortedArray

Enqueue

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

class Queue {

Node\* front;

Node\* rear;

public:

Queue() : front(nullptr), rear(nullptr) {}

void enqueue(int data) {

Node\* newNode = new Node(data);

if (rear == nullptr) {

front = rear = newNode;

} else {

rear->next = newNode;

rear = newNode;

}

cout << data << " enqueued to the queue.\n";

}

void dequeue() {

if (front == nullptr) {

cout << "Queue underflow. Nothing to dequeue.\n";

return;

}

Node\* temp = front;

front = front->next;

cout << temp->data << " dequeued from the queue.\n";

delete temp;

if (front == nullptr) {

rear = nullptr;

}

}

void display() const {

if (front == nullptr) {

cout << "The queue is empty.\n";

return;

}

Node\* current = front;

cout << "Queue elements: ";

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

~Queue() {

while (front != nullptr) {

Node\* temp = front;

front = front->next;

delete temp;

}

}

};

int main() {

Queue queue;

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

queue.display();

queue.dequeue();

queue.display();

queue.dequeue();

queue.display();

queue.dequeue();

queue.display();

return 0;

}

DLL

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node\* prev;

Node(int value) : data(value), next(nullptr), prev(nullptr) {}

};

class DoublyLinkedList {

Node\* head;

public:

DoublyLinkedList() : head(nullptr) {}

void insertAtEnd(int data) {

Node\* newNode = new Node(data);

if (head == nullptr) {

head = newNode;

} else {

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

}

cout << data << " inserted at the end.\n";

}

void deleteFromBeginning() {

if (head == nullptr) {

cout << "The list is empty. Nothing to delete.\n";

return;

}

Node\* temp = head;

head = head->next;

if (head != nullptr) {

head->prev = nullptr;

}

delete temp;

cout << "Node deleted from the beginning.\n";

}

void deleteFromPosition(int position) {

if (head == nullptr) {

cout << "The list is empty. Nothing to delete.\n";

return;

}

if (position < 1) {

cout << "Position should be greater than 0.\n";

return;

}

Node\* current = head;

int currentPosition = 1;

if (position == 1) {

deleteFromBeginning();

return;

}

while (current != nullptr && currentPosition < position) {

current = current->next;

currentPosition++;

}

if (current == nullptr) {

cout << "The position is out of range.\n";

return;

}

if (current->prev != nullptr) {

current->prev->next = current->next;

}

if (current->next != nullptr) {

current->next->prev = current->prev;

}

delete current;

cout << "Node deleted from position " << position << ".\n";

}

void printList() const {

if (head == nullptr) {

cout << "The list is empty.\n";

return;

}

Node\* current = head;

while (current != nullptr) {

cout << current->data << " <-> ";

current = current->next;

}

cout << "null\n";

}

~DoublyLinkedList() {

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

DoublyLinkedList list;

list.insertAtEnd(10);

list.insertAtEnd(20);

list.insertAtEnd(30);

list.insertAtEnd(40);

list.insertAtEnd(50);

cout << "Initial Linked List: ";

list.printList();

list.deleteFromBeginning();

cout << "After deleting from the beginning: ";

list.printList();

list.deleteFromPosition(3);

cout << "After deleting from position 3: ";

list.printList();

return 0;

}

Inorder traversal bst

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

Node\* insert(Node\* root, int data) {

if (root == nullptr) {

return new Node(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

}

else {

root->right = insert(root->right, data);

}

return root;

}

void inorderTraversal(Node\* root) {

if (root == nullptr) {

return;

}

inorderTraversal(root->left);

cout << root->data << " ";

inorderTraversal(root->right);

}

int main() {

Node\* root = nullptr;

int n, value;

cout << "Enter the number of elements you want to insert in the BST: ";

cin >> n;

cout << "Enter the elements to insert into the BST:\n";

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

cin >> value;

root = insert(root, value);

}

cout << "Inorder traversal of the BST: ";

inorderTraversal(root);

cout << endl;

return 0;

}

Insert in bst

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

Node\* insert(Node\* root, int data) {

if (root == nullptr) {

return new Node(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

}

else {

root->right = insert(root->right, data);

}

return root;

}

void inorderTraversal(Node\* root) {

if (root == nullptr) {

return;

}

inorderTraversal(root->left);

cout << root->data << " ";

inorderTraversal(root->right);

}

int main() {

Node\* root = nullptr;

int n, value;

cout << "Enter the number of elements you want to insert in the BST: ";

cin >> n;

cout << "Enter the elements to insert into the BST:\n";

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

cin >> value;

root = insert(root, value);

}

cout << "Inorder traversal of the BST: ";

inorderTraversal(root);

cout << endl;

return 0;

}

SLL at the beginning and any position

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node(int value) {

data = value;

next = nullptr;

}

};

class SinglyLinkedList {

private:

Node\* head;

public:

SinglyLinkedList() : head(nullptr) {}

void insertAtBeginning(int data) {

Node\* newNode = new Node(data);

newNode->next = head;

head = newNode;

cout << data << " inserted at the beginning.\n";

}

void insertAtPosition(int data, int position) {

if (position < 1) {

cout << "Position should be greater than 0.\n";

return;

}

Node\* newNode = new Node(data);

if (position == 1) {

newNode->next = head;

head = newNode;

cout << data << " inserted at position " << position << ".\n";

return;

}

Node\* current = head;

int currentPosition = 1;

while (current != nullptr && currentPosition < position - 1) {

current = current->next;

currentPosition++;

}

if (current == nullptr) {

cout << "The position is out of range.\n";

delete newNode;

return;

}

newNode->next = current->next;

current->next = newNode;

cout << data << " inserted at position " << position << ".\n";

}

void printList() const {

if (head == nullptr) {

cout << "The list is empty.\n";

return;

}

Node\* current = head;

while (current != nullptr) {

cout << current->data << " -> ";

current = current->next;

}

cout << "null\n";

}

~SinglyLinkedList() {

Node\* current = head;

while (current != nullptr) {

Node\* nextNode = current->next;

delete current;

current = nextNode;

}

}

};

int main() {

SinglyLinkedList list;

list.insertAtBeginning(10);

list.insertAtBeginning(20);

list.insertAtPosition(30, 2);

list.insertAtPosition(40, 1);

list.insertAtPosition(50, 4);

cout << "Linked List: ";

list.printList();

return 0;

}

Merge sort

#include <iostream>

#include <vector>

using namespace std;

void merge(vector<int>& arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

vector<int> leftArr(n1), rightArr(n2);

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

leftArr[i] = arr[left + i];

}

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

rightArr[j] = arr[mid + 1 + j];

}

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

while (i < n1 && j < n2) {

if (leftArr[i] <= rightArr[j]) {

arr[k] = leftArr[i];

i++;

} else {

arr[k] = rightArr[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = leftArr[i];

i++;

k++;

}

while (j < n2) {

arr[k] = rightArr[j];

j++;

k++;

}

}

void mergeSort(vector<int>& arr, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

void displayArray(const vector<int>& arr) {

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

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

}

cout << endl;

}

int main() {

int n;

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

cin >> n;

vector<int> arr(n);

cout << "Enter the elements of the array:\n";

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

cin >> arr[i];

}

cout << "Original array: ";

displayArray(arr);

mergeSort(arr, 0, n - 1);

cout << "Sorted array: ";

displayArray(arr);

return 0;

}

Search bst

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

Node\* insert(Node\* root, int data) {

if (root == nullptr) {

return new Node(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

}

else {

root->right = insert(root->right, data);

}

return root;

}

Node\* search(Node\* root, int key) {

if (root == nullptr || root->data == key) {

return root;

}

if (key < root->data) {

return search(root->left, key);

}

return search(root->right, key);

}

void inorderTraversal(Node\* root) {

if (root == nullptr) {

return;

}

inorderTraversal(root->left);

cout << root->data << " ";

inorderTraversal(root->right);

}

int main() {

Node\* root = nullptr;

int n, value, key;

cout << "Enter the number of elements you want to insert in the BST: ";

cin >> n;

cout << "Enter the elements to insert into the BST:\n";

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

cin >> value;

root = insert(root, value);

}

cout << "Inorder traversal of the BST: ";

inorderTraversal(root);

cout << endl;

cout << "Enter the element to search in the BST: ";

cin >> key;

Node\* result = search(root, key);

if (result != nullptr) {

cout << "Element " << key << " found in the BST." << endl;

} else {

cout << "Element " << key << " not found in the BST." << endl;

}

return 0;

}

SLL insertion at the end

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

class SinglyLinkedList {

Node\* head;

public:

SinglyLinkedList() : head(nullptr) {}

void insertAtEnd(int data) {

Node\* newNode = new Node(data);

if (head == nullptr) {

head = newNode;

} else {

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

cout << data << " inserted at the end.\n";

}

void insertAtPosition(int data, int position) {

if (position < 1) {

cout << "Position should be greater than 0.\n";

return;

}

Node\* newNode = new Node(data);

if (position == 1) {

newNode->next = head;

head = newNode;

} else {

Node\* current = head;

for (int i = 1; current != nullptr && i < position - 1; i++) {

current = current->next;

}

if (current == nullptr) {

cout << "The position is out of range.\n";

delete newNode;

return;

}

newNode->next = current->next;

current->next = newNode;

}

cout << data << " inserted at position " << position << ".\n";

}

void printList() const {

if (head == nullptr) {

cout << "The list is empty.\n";

return;

}

Node\* current = head;

while (current != nullptr) {

cout << current->data << " -> ";

current = current->next;

}

cout << "null\n";

}

~SinglyLinkedList() {

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

SinglyLinkedList list;

list.insertAtEnd(10);

list.insertAtEnd(20);

list.insertAtPosition(30, 2);

list.insertAtPosition(40, 1);

list.insertAtPosition(50, 4);

cout << "Linked List: ";

list.printList();

return 0;

}