#include <iostream>

#include <climits>

using namespace std;

struct Pair {

int min;

int max;

};

// Function to find min and max using divide and conquer

Pair getMinMax(int arr[], int low, int high) {

Pair result, leftPair, rightPair;

// If there's only one element

if (low == high) {

result.min = arr[low];

result.max = arr[low];

return result;

}

// If there are two elements

if (high == low + 1) {

if (arr[low] < arr[high]) {

result.min = arr[low];

result.max = arr[high];

} else {

result.min = arr[high];

result.max = arr[low];

}

return result;

}

// If more than two elements, divide the array

int mid = (low + high) / 2;

leftPair = getMinMax(arr, low, mid);

rightPair = getMinMax(arr, mid + 1, high);

// Compare the results of two halves

result.min = (leftPair.min < rightPair.min) ? leftPair.min : rightPair.min;

result.max = (leftPair.max > rightPair.max) ? leftPair.max : rightPair.max;

return result;

}

int main() {

int arr[] = {12, 3, 45, 7, 19, 34, 1};

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

Pair result = getMinMax(arr, 0, n - 1);

cout << "Minimum element is " << result.min << endl;

cout << "Maximum element is " << result.max << endl;

return 0;

}

#include <iostream>

#include <climits>

using namespace std;

// Function to find the maximum sum crossing the midpoint

int maxCrossingSum(int arr[], int low, int mid, int high) {

int left\_sum = INT\_MIN;

int sum = 0;

for (int i = mid; i >= low; i--) {

sum += arr[i];

if (sum > left\_sum)

left\_sum = sum;

}

int right\_sum = INT\_MIN;

sum = 0;

for (int i = mid + 1; i <= high; i++) {

sum += arr[i];

if (sum > right\_sum)

right\_sum = sum;

}

return left\_sum + right\_sum;

}

// Function to find the maximum subarray sum using divide and conquer

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

if (low == high)

return arr[low];

int mid = (low + high) / 2;

int left\_sum = maxSubArraySum(arr, low, mid);

int right\_sum = maxSubArraySum(arr, mid + 1, high);

int cross\_sum = maxCrossingSum(arr, low, mid, high);

return max(max(left\_sum, right\_sum), cross\_sum);

}

int main() {

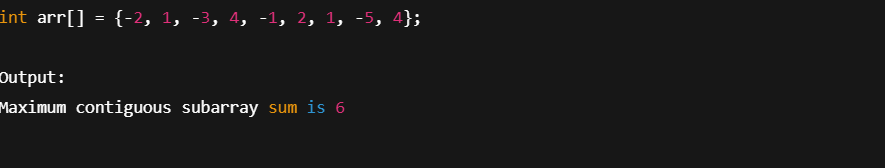
int arr[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4};

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

int max\_sum = maxSubArraySum(arr, 0, n - 1);

cout << "Maximum contiguous subarray sum is " << max\_sum << endl;

return 0;

}

#include <iostream>

#include <list>

using namespace std;

// Hash Table class using chaining

class HashTable {

int bucketSize;

list<int> \*table;

public:

HashTable(int size) {

bucketSize = size;

table = new list<int>[bucketSize];

}

// Hash function

int hashFunction(int key) {

return key % bucketSize;

}

// Insert element

void insert(int key) {

int index = hashFunction(key);

table[index].push\_back(key);

cout << key << " inserted.\n";

}

// Search element

bool search(int key) {

int index = hashFunction(key);

for (int val : table[index]) {

if (val == key)

return true;

}

return false;

}

// Delete element

void remove(int key) {

int index = hashFunction(key);

table[index].remove(key);

cout << key << " deleted if it existed.\n";

}

// Display hash table

void display() {

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

cout << "Bucket " << i << ":";

for (int val : table[i]) {

cout << " -> " << val;

}

cout << endl;

}

}

};

int main() {

int size;

cout << "Enter the size of hash table: ";

cin >> size;

HashTable ht(size);

int choice, key;

while (true) {

cout << "\n1. Insert\n2. Search\n3. Delete\n4. Display\n5. Exit\nEnter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter key to insert: ";

cin >> key;

ht.insert(key);

break;

case 2:

cout << "Enter key to search: ";

cin >> key;

if (ht.search(key))

cout << key << " found in the hash table.\n";

else

cout << key << " not found in the hash table.\n";

break;

case 3:

cout << "Enter key to delete: ";

cin >> key;

ht.remove(key);

break;

case 4:

ht.display();

break;

case 5:

cout << "Exiting...\n";

return 0;

default:

cout << "Invalid choice! Please try again.\n";

}

}

}