1. **Min and Max using Divide and Conquer**

**#include <stdio.h>**

**struct Result {**

**int min;**

**int max;**

**};**

**struct Result findMinMax(int arr[], int low, int high) {**

**struct Result res, left, right;**

**int mid;**

**if (low == high) {**

**res.min = arr[low];**

**res.max = arr[low];**

**return res;**

**}**

**if (high == low + 1) {**

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

**res.min = arr[low];**

**res.max = arr[high];**

**} else {**

**res.min = arr[high];**

**res.max = arr[low];**

**}**

**return res;**

**}**

**mid = (low + high) / 2;**

**left = findMinMax(arr, low, mid);**

**right = findMinMax(arr, mid + 1, high);**

**res.min = (left.min < right.min) ? left.min : right.min;**

**res.max = (left.max > right.max) ? left.max : right.max;**

**return res;**

**}**

**int main() {**

**int arr[] = {4, 7, 1, 9, 12, 3};**

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

**struct Result result = findMinMax(arr, 0, n - 1);**

**printf("Minimum: %d\n", result.min);**

**printf("Maximum: %d\n", result.max);**

**return 0;**

**}**

**2. Maximum Subarray Problem (Divide and Conquer)**

#include <stdio.h>

#include <limits.h>

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;

}

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

if (low == high)

return arr[low];

int mid = (low + high) / 2;

int left = maxSubArray(arr, low, mid);

int right = maxSubArray(arr, mid + 1, high);

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

if (left >= right && left >= cross)

return left;

else if (right >= left && right >= cross)

return right;

else

return cross;

}

int main() {

int arr[] = {13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7};

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

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

printf("Maximum subarray sum is %d\n", max\_sum);

return 0;

}

**3. Hash Table with Chaining (Menu-driven)**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 10

struct Node {

int data;

struct Node\* next;

};

struct Node\* hashTable[SIZE] = {NULL};

int hashFunction(int key) {

return key % SIZE;

}

void insert(int key) {

int index = hashFunction(key);

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = key;

newNode->next = hashTable[index];

hashTable[index] = newNode;

printf("Inserted %d\n", key);

}

void search(int key) {

int index = hashFunction(key);

struct Node\* temp = hashTable[index];

while (temp != NULL) {

if (temp->data == key) {

printf("Found %d at index %d\n", key, index);

return;

}

temp = temp->next;

}

printf("Key %d not found.\n", key);

}

void deleteKey(int key) {

int index = hashFunction(key);

struct Node \*temp = hashTable[index], \*prev = NULL;

while (temp != NULL) {

if (temp->data == key) {

if (prev == NULL)

hashTable[index] = temp->next;

else

prev->next = temp->next;

free(temp);

printf("Deleted %d\n", key);

return;

}

prev = temp;

temp = temp->next;

}

printf("Key %d not found for deletion.\n", key);

}

void display() {

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

printf("%d:", i);

struct Node\* temp = hashTable[i];

while (temp) {

printf(" -> %d", temp->data);

temp = temp->next;

}

printf("\n");

}

}

int main() {

int choice, key;

while (1) {

printf("\n1. Insert\n2. Search\n3. Delete\n4. Display\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter key to insert: ");

scanf("%d", &key);

insert(key);

break;

case 2:

printf("Enter key to search: ");

scanf("%d", &key);

search(key);

break;

case 3:

printf("Enter key to delete: ");

scanf("%d", &key);

deleteKey(key);

break;

case 4:

display();

break;

case 5:

printf("Exiting...\n");

exit(0);

default:

printf("Invalid choice. Try again.\n");

}

}

return 0;

}