Q-1: Given a non-negative integer 'x', find its square root with a specified precision (number of decimal places) using the binary search algorithm.

Sample test case:

|  |
| --- |
| Input:  Enter a non-negative integer: 36  Enter the precision (number of decimal places) for the square root: 2  Output: Square root of 36 with precision 2 decimal places: 6.1875 |

Solution:

#include <iostream>

using namespace std;

// Function to find the square root of a non-negative integer 'x'

double squareRoot(int x, int precision) {

double low = 0;

double high = x;

double mid;

// Binary search with a specified precision

while ((high - low) > precision) {

mid = (low + high) / 2;

double square = mid \* mid;

if (square == x) {

return mid;

} else if (square < x) {

low = mid;

} else {

high = mid;

}

}

return low + (high - low) / 2;

}

int main() {

int x;

cout << "Enter a non-negative integer: ";

cin >> x;

if (x < 0) {

cout << "Error: The entered number is negative.";

return 1;

}

int precision;

cout << "Enter the precision (number of decimal places) for the square root: ";

cin >> precision;

if (precision < 0) {

cout << "Error: Precision cannot be negative.";

return 1;

}

double sqrtValue = squareRoot(x, precision);

cout << "Square root of " << x << " with precision " << precision << " decimal places: " << sqrtValue;

return 0;

}

Q-2: Given an array of integers, sort it in ascending order using the Bubble Sort algorithm with an optimized approach.

Sample test case:

|  |
| --- |
| Input: Original Array: 64 34 25 12 22 11 90  Output: Sorted Array: 11 12 22 25 34 64 90 |

Solution:

#include <iostream>

using namespace std;

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

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

bool swapped = false;

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

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

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

swapped = true;

}

}

// If no two elements were swapped in the inner loop, the array is already sorted.

if (!swapped) {

break;

}

}

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

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

cout << "Original Array: ";

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

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

bubbleSort(arr, n);

cout << "\nSorted Array: ";

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

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

return 0;

}

Q-3: Given an array of integers, sort it in ascending order using the Insertion Sort algorithm with a binary search approach for finding the insertion position.

Sample test case:

|  |
| --- |
| Input: Original Array: 64 34 25 12 22 11 90  key=64  Output:  Sorted Array: 11 12 22 25 34 64 90  Search 64 : 5 |
| Solution: |

#include <iostream>

using namespace std;

int binarySearch(int arr[], int left, int right, int key) {

while (left <= right) {

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

if (arr[mid] == key) {

return mid;

} else if (arr[mid] > key) {

right = mid - 1;

} else {

left = mid + 1;

}

}

return left;

}

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

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

int key = arr[i];

int j = i - 1;

int insertPos = binarySearch(arr, 0, j, key);

while (j >= insertPos) {

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

j--;

}

arr[j + 1] = key;

}

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

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

cout << "Original Array: ";

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

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

insertionSort(arr, n);

cout << "\nSorted Array: ";

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

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

cout<<"\nSearch 64 : "<<binarySearch(arr,0,6,64);

return 0;

}

Q-4: Given an array of integers and an integer 'k', find the smallest 'k' elements from the array using the selection sort algorithm.

Sample test case:

|  |
| --- |
| Input: Original Array: 64 25 12 22 11  Output: Smallest 3 Elements: 11 12 22 |

Solution:

#include <iostream>

using namespace std;

void selectionSortSmallestK(int arr[], int n, int k) {

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

int minIndex = i;

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

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

minIndex = j;

}

}

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

}

}

int main() {

int arr[] = {64, 25, 12, 22, 11};

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

int k = 3;

cout << "Original Array: ";

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

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

}

selectionSortSmallestK(arr, n, k);

cout << "\nSmallest " << k << " Elements: ";

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

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

}

return 0;

}