East West University
Department of Computer Science and Engineering

Course: CSE 246(Algorithms)
Section - 02

Submitted To :	**Submitted By :**
Dr. Md. Tauhid Bin Iqbal Assistant Professor Department of CSE East West University	**Name** : Hafsa Ferdousi **Student ID** : 2023-3-60-321 **Department** : CSE **Date** : 28 / 10 / 2025

Lab Report:04

Problem 1: Prime Number Check

Original code	Optimized code

#include <iostream>
#include <cmath>
using namespace std;
bool isPrime(int n) {
if (n <= 1) {
return false;
}
int totalComparisons = 0;
int totalIterations = 0;
for (int i = 2; i < n; i++) {
 	
totalIterations++;
totalComparisons++;
if (n % i == 0) {
cout << "Total comparisons: " <<
totalComparisons << endl;
cout << "Total iterations: " << totalIterations
<< endl;
return false;
}
}
 	
cout << "Total comparisons: " <<
totalComparisons << endl;
cout << "Total iterations: " << totalIterations
<< endl;
return true;
}
int main() {
int n = 29;
if (isPrime(n)) {
cout << n << " is a prime number." << endl;
} else {
cout << n << " is not a prime number." <<
endl;
}
return 0;}
	#include <iostream>
	#include <cmath>
	using namespace std;
	
	bool isPrime(int n) {
	  if (n <= 1)
	  return false;
	  int totalComparisons = 0;
	  int totalIterations = 0;
	
	  for (int i = 2; i * i <= n; i++) {
	  totalIterations++;
	  totalComparisons++;
	  if (n % i == 0) {
	  cout << "Total comparisons: " <<
	totalComparisons << endl;
	  cout << "Total iterations: " <<
	totalIterations << endl;
	  return false;
	  }
	  }
	
	  cout << "Total comparisons: " <<
	totalComparisons << endl;
	  cout << "Total iterations: " <<
	totalIterations << endl;
	  return true;
	}
	
	int main() {
	  int n = 29;
	  if (isPrime(n))
	  cout << n << " is a prime number." <<
	endl;
	  else
	  cout << n << " is not a prime number."
	<< endl;
	  return 0;
	}

Time complexity O(n) Space complexity O(1)	Time complexity O($\sqrt{n}$) Space complexity O(1)

Problem 2: search a element in a array (Insertion Search)

Original code	Optimized code

#include <iostream>
using namespace std;
int main() {
int arr[] = {4, 8, 1, 7, 3};
int n = 5;
 	
int x = 7;
bool found = false;
for(int i = 0; i < n; i++) {
int current = arr[i];
 	
if(current == x) {
for(int j = i; j < n; j++) {
if(arr[j] == x) {
found = true;
break;
}
}
break;
}
}
if(found)
cout << "Element " << x << " is present in the array." << endl;
else
cout << "Element " << x << " is NOT present in the array." << endl;
return 0;
}
	#include <iostream>
	using namespace std;
	
	int main() {
	  int arr[] = {4, 8, 1, 7, 3};
	  int n = 5;
	  int x = 7;
	
	  bool found = false;
	  for(int i = 0; i < n; i++) {
	  if(arr[i] == x) {
	  found = true;
	  break;
	  }
	  }
	
	  if(found)
	  cout << "Element " << x << " is present in the array." << endl;
	  else
	  cout << "Element " << x << " is NOT present in the array." << endl;
	
	  return 0;
	}

Time complexity O(n^2) Space complexity O(1)	Time complexity O(n) Space complexity O(1)

Problem 3: Merge Sort

Original code	Optimized code

#include <iostream>
#include <climits>
using namespace std;
// Merge two sorted subarrays
void merge(int arr[], int temp[], int left, int mid, int right) {
int i = left; // Starting index for left subarray
int j = mid + 1; // Starting index for right subarray
int k = left; // Starting index to store merged result
while (i <= mid && j <= right) {
if (arr[i] <= arr[j]) {
temp[k++] = arr[i++];
} else {
temp[k++] = arr[j++];
}
}
// Copy the remaining elements of the left subarray
while (i <= mid) {
temp[k++] = arr[i++];
 	
}
// Copy the remaining elements of the right subarray
while (j <= right) {
temp[k++] = arr[j++];
}
// Copy the sorted subarray into the original array
for (int i = left; i <= right; i++) {
arr[i] = temp[i];
}
}
// Merge Sort function
void mergeSort(int arr[], int temp[], int left, int right) {
if (left < right) {
int mid = left + (right - left) / 2; // Calculate mid point
// Recursively divide the array into two halves
mergeSort(arr, temp, left, mid); // Sort the left
	#include <iostream>
	using namespace std;
	
	void merge(int arr[], int temp[], int left, int mid, int right){
	int i = left, j = mid + 1, k = left;
	while(i <= mid && j <= right){
	if(arr[i] <= arr[j]) temp[k++] = arr[i++];
	else temp[k++] = arr[j++];
	}
	while(i <= mid) temp[k++] = arr[i++];
	while(j <= right) temp[k++] = arr[j++];
	for(int x = left; x <= right; x++) arr[x] = temp[x];
	}
	
	void mergeSort(int arr[], int temp[], int left, int right){
	if(left < right){
	int mid = left + (right-left)/2;
	mergeSort(arr, temp, left, mid);
	mergeSort(arr, temp, mid+1, right);
	merge(arr, temp, left, mid, right);
	}
	}
	
	int main(){
	int arr[] = {12,11,13,5,6,7};
	int n = sizeof(arr)/sizeof(arr[0]);
	int temp[n];
	
	mergeSort(arr,temp,0,n-1);
	
	cout<<"Sorted array: ";
	for(int i=0;i<n;i++) cout<<arr[i]<<" ";
	}

half
mergeSort(arr, temp, mid + 1, right); // Sort the right half
// Merge the sorted halves
merge(arr, temp, left, mid, right);
}
}
int main() {
int arr[] = {12, 11, 13, 5, 6, 7};
int n = sizeof(arr) / sizeof(arr[0]);
// Temporary array for merging
int temp[n];
cout << "Original array: ";
for (int i = 0; i < n; i++) {
cout << arr[i] << " ";
}
cout << endl;
mergeSort(arr, temp, 0, n - 1);
cout << "Sorted array: ";
for (int i = 0; i < n; i++) {
cout << arr[i] << " ";
}
cout << endl;
return 0;
}	

Time complexity O(n log n) Space complexity O(n)	Time complexity O(n log n) Space complexity O(n)

Problem 4: Finding Maximum in an Array

Original code	Optimized code

#include <iostream>
using namespace std;
int main() {
int arr[] = {2, 4, 6, 8, 10};
int n = 5;
int maxArr[100];
for(int i = 0; i < n; i++) {
maxArr[i] = arr[i];
	#include <iostream>
	using namespace std;
	
	int main() {
	  int arr[] = {2, 4, 6, 8, 10};
	  int n = 5;
	
	  int maxVal = arr[0];

}
int maxVal = maxArr[0];
for(int i = 1; i < n; i++) {
if(maxArr[i] > maxVal) {
maxVal = maxArr[i];
}
}
cout << "Maximum Element: " << maxVal << endl;
return 0;
}
	for(int i = 1; i < n; i++) {
	if(arr[i] > maxVal) maxVal = arr[i];
	}
	
	cout << "Maximum Element: " << maxVal << endl;
	return 0;
	}

Time complexity O(n) Space complexity O(n)	Time complexity O(n) Space complexity O(1)

Problem 5: Remove Duplicate Elements from an Array

Original code	Optimized code

#include <iostream>
using namespace std;
int main() {
int arr[] = {1, 2, 3, 2, 4, 5, 5};
int n = 7;
int copy[100];
for(int i = 0; i < n; i++) {
copy[i] = arr[i];
}
int unique[100];
int uniqueCount = 0;
for(int i = 0; i < n; i++) {
bool found = false;
for(int j = 0; j < uniqueCount; j++) {
if(copy[i] == unique[j]) {
 	
found = true;
break;
}
}
if(!found) {
unique[uniqueCount++] = copy[i];
}
	#include <iostream>
	using namespace std;
	
	int main() {
	int arr[] = {1,2,3,2,4,5,5};
	int n = 7;
	
	bool seen[1000] = {false};
	
	cout << "Array with unique elements: ";
	for(int i = 0; i < n; i++) {
	if(!seen[arr[i]]) {
	cout << arr[i] << " ";
	seen[arr[i]] = true;
	}
	}
	cout << endl;
	return 0;
	}

}
cout << "Array with unique elements: ";
for(int i = 0; i < uniqueCount; i++) {
cout << unique[i] << " ";
}
cout << endl;
return 0;
}	

Time complexity O(n^2) Space complexity O(n)	Time complexity O(n) Space complexity O(n)

Problem 6: Check if a Sub array with Given Sum Exists

Original code	Optimized code

#include <iostream>
#include <vector>
using namespace std;
int main() {
vector<int> arr = {1, 2, 3, 7, 5};
int n = arr.size();
int targetSum = 12;
bool found = false;
for (int i = 0; i < n; i++) {
for (int j = i; j < n; j++) {
int sum = 0;
for (int k = i; k <= j; k++) {
sum += arr[k];
}
if (sum == targetSum) {
found = true;
break;
}
}
if (found) break;
}
if (found) {
cout << "Subarray with given sum exists." << endl;
} else {
cout << "No subarray with given sum exists." << endl;
	#include <iostream>
	using namespace std;
	
	int main() {
	int arr[] = {1,2,3,7,5};
	int n = 5, targetSum = 12;
	
	int start = 0, sum = 0;
	bool found = false;
	
	for(int end = 0; end < n; end++){
	sum += arr[end];
	
	while(sum > targetSum && start <= end){
	sum -= arr[start];
	start++;
	}
	
	if(sum == targetSum){
	found = true;
	break;
	}
	}
	
	if(found) cout << "Subarray with given sum exists." << endl;
	else cout << "No subarray with given sum

} return 0; }	exists." << endl; return 0; }
Time complexity O(n^3) Space complexity O(1)	Time complexity O(n) Space complexity O(1)

Problem 7: Find the First Repeated Element in an Array

Original code	Optimized code

#include <iostream>
#include <vector>
using namespace std;
int main() {
vector<int> arr = {1, 2, 3, 4, 5, 3, 6, 7};
int n = arr.size();
int repeatedElement = -1;
bool found = false;
for (int i = 0; i < n; i++) {
for (int j = i + 1; j < n; j++) {
if (arr[i] == arr[j]) {
repeatedElement = arr[i];
found = true;
break;
}
}
if (found) break;
}
if (repeatedElement != -1) {
cout << "First repeated element: " <<
repeatedElement << endl;
} else {
cout << "No repeated elements found." <<
endl;
}
return 0;
}
	#include <iostream>
	using namespace std;
	
	int main() {
	int arr[] = {1,2,3,4,5,3,6,7};
	int n = 8;
	
	bool seen[1000] = {false};
	
	for(int i = 0; i < n; i++){
	if(seen[arr[i]]){
	cout << "First repeated element: " <<
	arr[i];
	return 0;
	}
	seen[arr[i]] = true;
	}
	
	cout << "No repeated elements found." <<
	endl;
	return 0;
	}

Time complexity O(n^2) Space complexity O(1)	Time complexity O(n) Space complexity O(n)

Problem 8: Insertion Sort

Original code	Optimized code

#include <iostream>
#include <climits>
using namespace std;
void insertionSort(int arr[], int n) {
int totalComparisons = 0;
int totalSwaps = 0;
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--;
totalComparisons++;
totalSwaps++;
}
arr[j + 1] = key;
if (j >= 0) {
totalComparisons++;
}
}
cout << "Total comparisons: " <<
totalComparisons << endl;
cout << "Total swaps: " << totalSwaps <<
endl;
}
int main() {
int arr[] = {12, 11, 13, 5, 6};
int n = sizeof(arr) / sizeof(arr[0]);
cout << "Original array: ";
for (int i = 0; i < n; i++) {
cout << arr[i] << " ";
}
cout << endl;
insertionSort(arr, n);
cout << "Sorted array: ";
for (int i = 0; i < n; i++) {
cout << arr[i] << " ";
}
cout << endl;
return 0;
	#include <iostream>
	using namespace std;
	
	void insertionSort(int arr[], int n) {
	for(int i = 1; i < n; i++){
	int key = arr[i], j = i - 1;
	while(j >= 0 && arr[j] > key){
	arr[j+1] = arr[j];
	j--;
	}
	arr[j+1] = key;
	}
	}
	
	int main() {
	int arr[] = {12, 11, 13, 5, 6};
	int n = 5;
	insertionSort(arr, n);
	
	cout << "Sorted array: ";
	for(int i = 0; i < n; i++) cout << arr[i] << "
	";
	}

}	
Time complexity O(n^2) wost,O(n)best Space complexity O(1)	Time complexity O(n^2) wost,O(n)best Space complexity O(1)