**B.Tech Computer Engineering**

**Semester – III**

**Design and Analysis of Algorithm**

**Course Code: 20CP209P**

**2024-25**



**Name:Samarth Vala**

**Roll No:23BCP422**

Submitted to

Dr. Shivangi Surti

Department of Computer Engineering

School of Technology

Pandit Deendayal Energy University

Gandhinagar, Gujarat, India

April 2025

**SCHOOL OF TECHNOLOGY**

**PANDIT DEENDAYAL ENERGY UNIVERSITY**

**GANDHINAGAR, GUJARAT, INDIA**

**CERTIFICATE**

This is to certify that **Mr Samarth Vala**, Enrollment no**. 23BCP422** of **4rd Semester** degree course in **Computer Engineering** has satisfactorily prepared and presented her Term Work in **Design and Analysis of Algorithm LAB 20CP209P** within four walls of the laboratory of this Institute during \_\_\_\_\_\_\_\_\_\_\_ to \_\_\_\_\_\_\_\_\_\_\_.

**Date of Submission: (Faculty In-Charge)**

**Dr. Shivangi Surti**

**Practical 1:**

**Aim: Implement sorting algorithms and find their time and space complexities**

**Bubble sort:**

**Code:**

#include<stdio.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

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

int noPass=0;

int noIteration=0;

int noExchange=0;

int flag=0;

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

noPass++;

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

noIteration++;

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

swap(&a[j],&a[j+1]);

flag=0;

noExchange++;

}

}

if(flag){

break;

}

}

printf("The sorted array is: ");

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

printf("%d ",a[i]);

}

printf("\n");

printf("Number of passes: %d\n", noPass);

printf("Number of iterations: %d\n", noIteration);

printf("Number of exchanges: %d\n", noExchange);

}

int main() {

printf("enter the size of array: ");

int n;

scanf("%d",&n);

int a[n];

printf("Enter the elements of array:\n");

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

printf("enter element %d:",i);

scanf("%d",&a[i]);

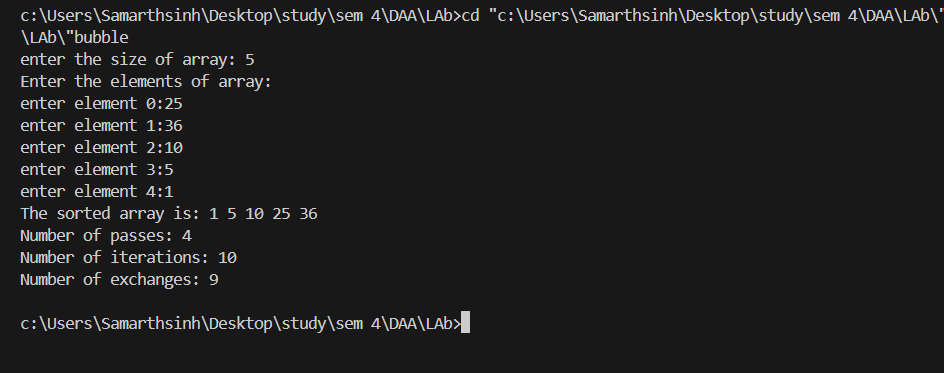
}

bubbleSort(a,n);

return 0;

}

**Output:**

****

**Time complexity for bubble sort:**

Worst: O(n^2)

Best: O(n)

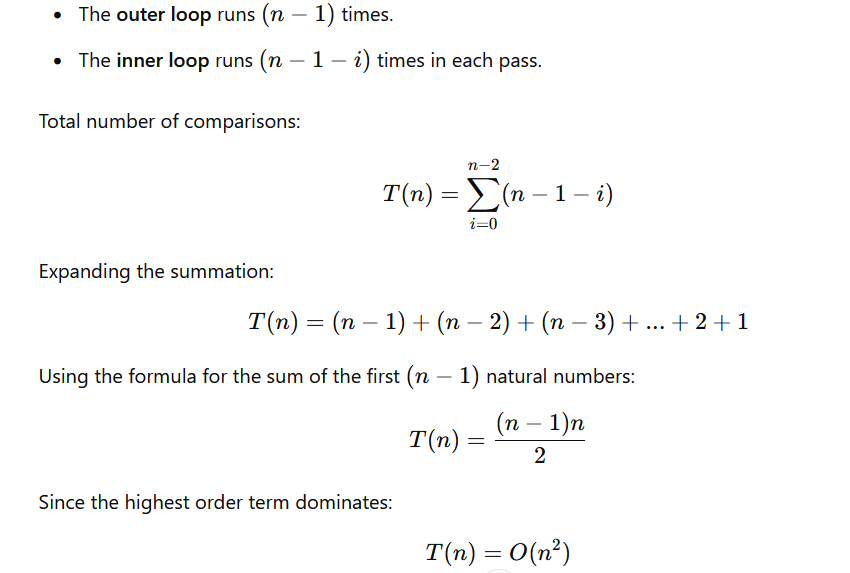
Average: O(n^2)

**Space complexity:**

O(1) in-place

**Complexity analysis:**

**For worst case:**

****

**Selection Sort:**

**Code:**

#include <stdio.h>

void swap(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = temp;

}

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

int noPass = 0;

int noExchanges = 0;

int noIter = 0;

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

noPass++;

int min = i;

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

noIter++;

if (a[j] < a[min]) {

min = j;

}

}

if (min != i) {

swap(&a[min], &a[i]);

noExchanges++;

}

}

printf("The sorted array: ");

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

printf("%d ", a[i]);

}

printf("\n");

printf("Number of passes (outer loop): %d\n", noPass);

printf("Number of iterations (inner loop): %d\n", noIter);

printf("Number of exchanges: %d\n", noExchanges);

}

int main() {

int n;

printf("Enter the size of array: ");

scanf("%d", &n);

if (n <= 0) {

printf("Invalid array size. Exiting.\n");

return 1;

}

int a[n];

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

printf("Enter element %d: ", i);

scanf("%d", &a[i]);

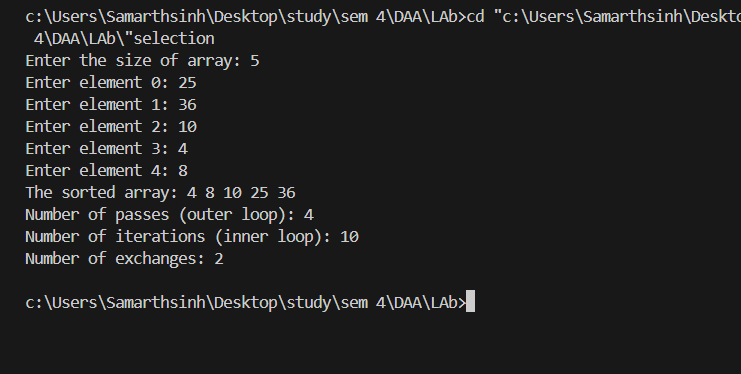
}

selectionSort(n, a);

return 0;

}

**Output:**

****

**Time complexity for selection sort:**

Worst: O(n^2)

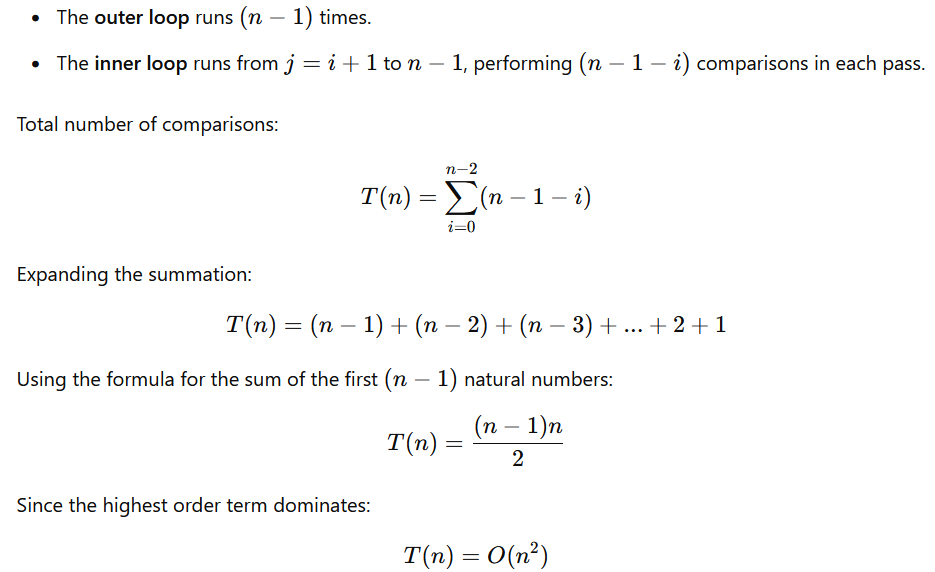
Best: O(n^2)

Average: O(n^2)

**Space complexity:**

O(1) in-place

**Complexity analysis for worst case:**



**Insertion Sort:**

**Code:**

#include <stdio.h>

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

int noComparisons = 0;

int noShifts = 0;

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

int key = arr[i];

int j = i - 1;

// Counting comparisons and shifts

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

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

j--;

noComparisons++;

}

arr[j + 1] = key;

if (j >= 0) {

noComparisons++;

}

}

printf("Number of comparisons: %d\n", noComparisons);

printf("Number of shifts: %d\n", noShifts);

}

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

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

printf("%d ", arr[i]);

}

printf("\n");

}

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d elements: ", n);

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

scanf("%d", &arr[i]);

}

printf("Original Array: ");

printArray(arr, n);

insertionSort(arr, n);

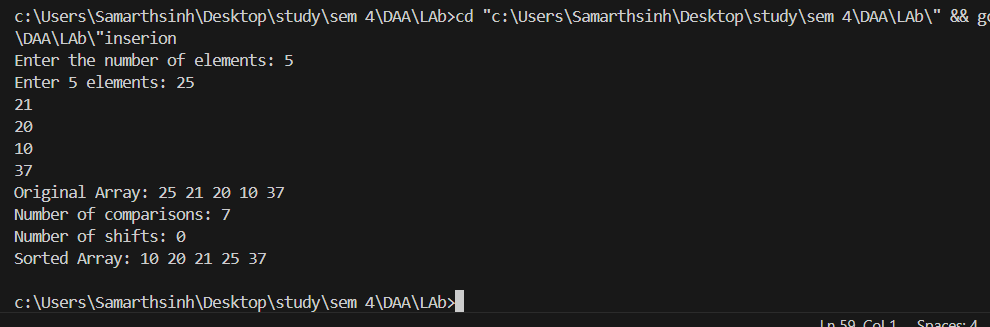
printf("Sorted Array: ");

printArray(arr, n);

return 0;

}

**Output:**

****

**Time complexity for insertion sort:**

Worst: O(n^2)

Best:O(n)

Average: O(n^2)

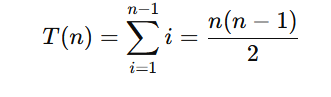
**Space complexity:**

O(1) in-place

**Complexity analysis for worst case:**

The inner loop will execute for every element. For the ith element, it will make ‘i’ comparisons. In the worst case, each element is compared with every other element in the sorted portion.

Total number of comparisons in worst case:



Thus T(n)=O(n^2)

**Merge sort:**

**Code:**

#include <stdio.h>

int comparisons = 0, swaps = 0;

void merge(int arr[], int p, int q, int r) {

int n1 = q - p + 1;

int n2 = r - q;

int arr1[n1], arr2[n2];

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

arr1[i] = arr[p + i];

swaps++;

}

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

arr2[j] = arr[q + 1 + j];

swaps++;

}

int i = 0, j = 0;

for (int k = p; k <= r; k++) {

comparisons++;

if (i < n1 && (j >= n2 || arr1[i] <= arr2[j])) {

arr[k] = arr1[i];

i++;

} else {

arr[k] = arr2[j];

j++;

}

swaps++;

}

}

void mergesort(int arr[], int p, int r) {

if (p < r) {

int q = p + (r - p) / 2;

mergesort(arr, p, q);

mergesort(arr, q + 1, r);

merge(arr, p, q, r);

}

}

int main() {

int n;

printf("Enter the size of the array: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements of the array:\n");

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

scanf("%d", &arr[i]);

}

mergesort(arr, 0, n - 1);

printf("Sorted array:\n");

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

printf("%d ", arr[i]);

}

printf("\n");

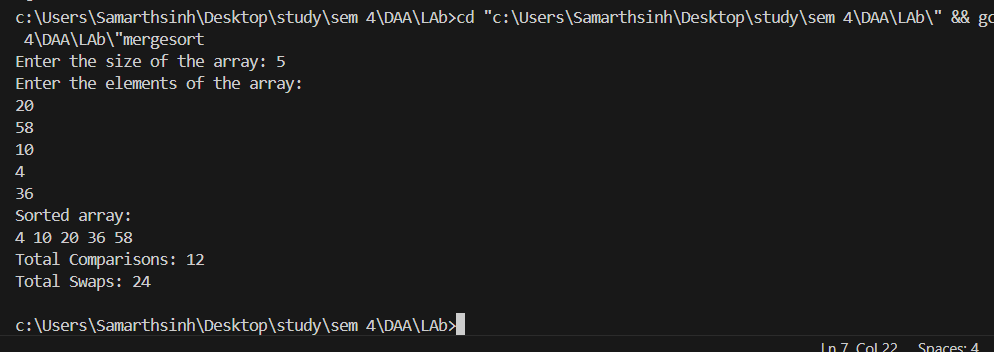
printf("Total Comparisons: %d\n", comparisons);

printf("Total Swaps: %d\n", swaps);

return 0;

}

**Output:**

****

**Time complexity for merge Sort:**

Worst: O(n log n)

Best: O(n log n)

Average: O(n log n)

**Space complexity:**

O(n)

**Complexity analysis:**

Worst case:

For the recurrence:

T(n)=2T(n/2)+O(n)

We have:

* a=2 (number of subproblems),
* b=2 (factor by which the problem size is divided),
* d=1 (power of n in the cost of merging, O(n))

Now, compare the value of a and b^d:

a=b^d

This matches **Case 2** of the Master Theorem for Divide and Conquer recurrences:

Thus T(n)=O(n log n)

**Quick sort:**

**Code:**

#include <stdio.h>

int comparisons = 0;

int swaps = 0;

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

swaps++;

}

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

int pivot = a[low];

int left = low + 1;

int right = high;

while (left <= right) {

while (left <= right && a[left] <= pivot) {

left++;

comparisons++;

}

while (left <= right && a[right] > pivot) {

right--;

comparisons++;

}

if (left < right) {

swap(&a[left], &a[right]);

}

}

swap(&a[low], &a[right]);

return right;

}

void quickSort(int a[], int low, int high) {

if (low < high) {

int pi = partition(a, low, high);

quickSort(a, low, pi - 1);

quickSort(a, pi + 1, high);

}

}

void printArray(int a[], int n) {

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

printf("%d ", a[i]);

}

printf("\n");

}

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int a[n];

printf("Enter %d elements: ", n);

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

scanf("%d", &a[i]);

}

printf("Original Array: ");

printArray(a, n);

quickSort(a, 0, n - 1);

printf("Sorted Array: ");

printArray(a, n);

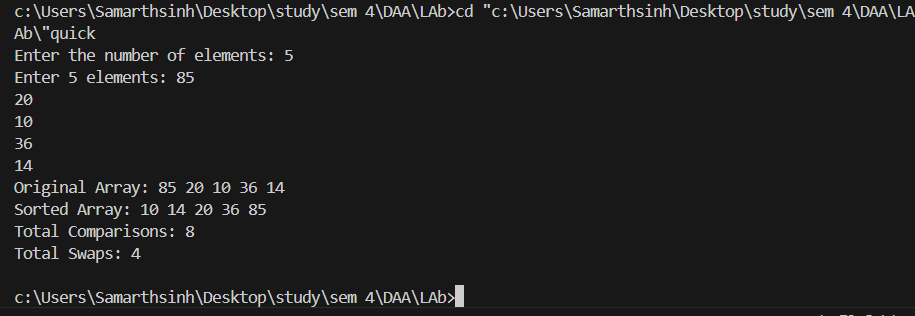
printf("Total Comparisons: %d\n", comparisons);

printf("Total Swaps: %d\n", swaps);

return 0;

}

**Output:**

****

**Time complexity for Quick Sort:**

Worst: O(n^2)

Best: O( n log n)

Average: O( n log n)

**Space complexity:**

O(log n) recursive calls

**Complexity analysis:**

In the worst case, the pivot chosen is always the smallest or largest element, leading to highly unbalanced partitions. In this case, one side will always be empty, and the recursion depth will be O(n).

Thus, the recurrence becomes:

T(n)=T(n−1)+O(n)

Expanding this recurrence:

T(n)=O(n)+O(n−1)+O(n−2)+⋯+O(1)

This results in a worst-case time complexity of:

T(n)=O(n^2)

For best case the recurrence equation is:

T(n)=2T(n/2​)+O(n)

So complexity is O(n logn)

**Practical 2:**

**Aim: Implement mathematical operation using LinkedList:**

**Addition:**

**Code:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node \*next;**

**};**

**struct Node \*createNode(int data) {**

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

**newNode->data = data;**

**newNode->next = NULL;**

**return newNode;**

**}**

**void insertAtFront(struct Node \*\*head, int data) {**

**struct Node \*newNode = createNode(data);**

**newNode->next = \*head;**

**\*head = newNode;**

**}**

**void insertAtEnd(struct Node \*\*head, int data) {**

**struct Node \*newNode = createNode(data);**

**if (\*head == NULL) {**

**\*head = newNode;**

**return;**

**}**

**struct Node \*temp = \*head;**

**while (temp->next != NULL) {**

**temp = temp->next;**

**}**

**temp->next = newNode;**

**}**

**void printList(struct Node \*head) {**

**struct Node \*temp = head;**

**while (temp != NULL) {**

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

**if (temp->next != NULL) {**

**printf(" => ");**

**}**

**temp = temp->next;**

**}**

**printf("\n");**

**}**

**struct Node \*AddList(struct Node \*h1, struct Node \*h2){**

**struct Node \*temp1=h1;**

**struct Node \*temp2=h2;**

**struct Node \*h3=NULL;**

**int carry = 0, sum = 0;**

**while (temp1!= NULL || temp2 != NULL || carry != 0) {**

**sum = carry;**

**if (temp1 != NULL) {**

**sum += temp1->data;**

**temp1 = temp1->next;**

**}**

**if (temp2 != NULL) {**

**sum += temp2->data;**

**temp2 = temp2->next;**

**}**

**int ans = sum % 10;**

**carry = sum / 10;**

**insertAtFront(&h3, ans);**

**}**

**return h3;**

**};**

**int main(){**

**struct Node \*h1 = NULL;**

**struct Node \*h2=NULL;**

**int data, choice, position, value;**

**printf("Enter the elements to insert into the linked list for 1st number (enter -1 to stop):\n");**

**while (1) {**

**printf("Enter value: ");**

**scanf("%d", &data);**

**if (data == -1) break;**

**insertAtFront(&h1, data);**

**}**

**printf("Enter the elements to insert into the linked list for 2nd number (enter -1 to stop):\n");**

**while (1) {**

**printf("Enter value: ");**

**scanf("%d", &data);**

**if (data == -1) break;**

**insertAtFront(&h2, data);**

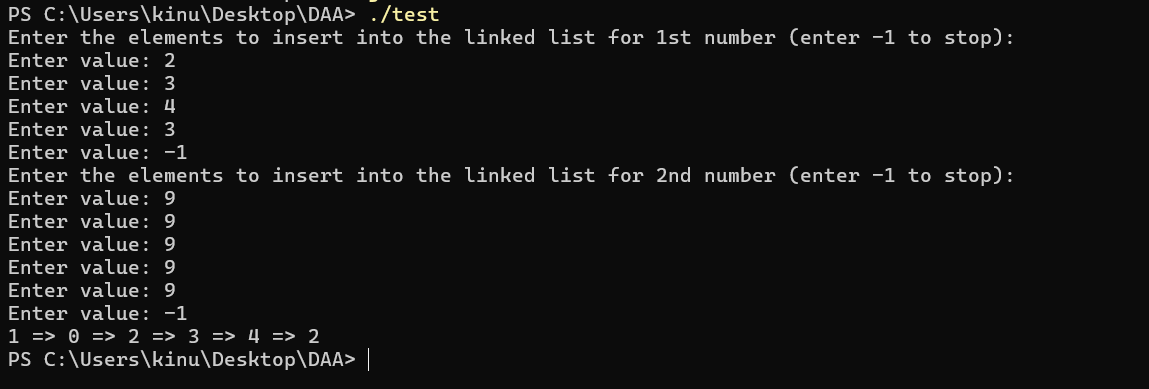
**}**

**struct Node \*sumList = AddList(h1, h2);**

**printList(sumList);**

**}**

**Output:**

****