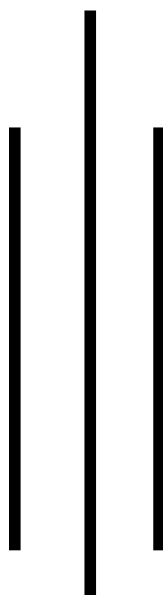




Tribhuvan University
Institute of Science and Technology
Amrit Campus
Thamel, Kathmandu



Lab Report

Design and Analysis of Algorithm (CSC325)

BSc CSIT Fifth Semester

Submitted by :

Name: -----

Roll no.: -----

Semester: -----

Batch: -----

Submitted to :

Asst. Prof. Dabbal Singh Mahara

Lab Sheet 1

Implementation of Iterative Algorithms

1. Aim

To implement iterative algorithms for generating Fibonacci series, computing GCD, performing sequential search, and sorting arrays using bubble, selection, and insertion sort.

2. Objectives

- To learn iterative programming techniques in C.
- To implement Fibonacci series, GCD, sequential search, and bubble, selection, and insertion sorting algorithms iteratively.
- To analyze the time and space complexity of each algorithm.

3. Generating Fibonacci Series

Concept

- The Fibonacci series is a sequence where each number is the sum of the two preceding numbers, starting with 0 and 1. The series is: 0, 1, 1, 2, 3, 5, 8, ...
- Iteratively, calculate each number using a loop and store only the last two numbers for efficiency.
- Time Complexity: $O(n)$
- Space Complexity: $O(1)$

Program

```
#include <stdio.h>

int main() {
    int n, a = 0, b = 1, c, i;
    printf("Enter n for Fibonacci series: ");
    scanf("%d", &n);
    printf("Fibonacci Series: %d %d ", a, b);
    for(i = 2; i < n; i++) {
        c = a + b;
```

```

    printf("%d ", c);

    a = b;

    b = c;

}

printf("\n");

return 0;

}

```

Output

```

d:\BSc CSIT\DAA\dabbal>gcc fibonacci.c

d:\BSc CSIT\DAA\dabbal>a.exe
Enter n for Fibonacci series: 10
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34

```

4. Calculating GCD using Euclidean Algorithm

Concept

- GCD is the largest number that divides two integers exactly.
- Iteratively, we repeatedly replace the larger number with the remainder until the remainder is 0.
- Formula: $\text{GCD}(a, b) = \text{GCD}(b, a \% b)$
- Time Complexity: $T(n) = O(\log(\min(a, b)))$
Space Complexity: $S(n) = O(1)$

Program

```

#include <stdio.h>

int main() {

    int a, b, x, y, temp;

    printf("Enter two numbers for GCD: ");

    scanf("%d %d", &x, &y);

    a = x; b = y;

    while(b != 0) {

```

```

    temp = b;
    b = a % b;
    a = temp;
}
printf("GCD of %d and %d is: %d\n", x, y, a);
return 0;
}

```

Output

```

d:\BSc CSIT\DAA\dabba1>gcc gcd.c

d:\BSc CSIT\DAA\dabba1>a.exe
Enter two numbers for GCD: 60 18
GCD of 60 and 18 is: 6

```

5. Implementing Sequential Search

Concept

- Sequential search scans each element of an array until it finds the key.
- Best Case Time Complexity: $T(n) = O(1)$
- Average Case Time Complexity: $T(n) = O(n)$
- Worst Case Time Complexity: $T(n) = O(n)$
- Space Complexity: $S(n) = O(1)$

Program

```

#include <stdio.h>

int main() {
    int size, arr[50], key, i, index = -1;
    printf("Enter array size: ");
    scanf("%d", &size);
    printf("Enter array elements: ");
    for(i = 0; i < size; i++)

```

```

        scanf("%d", &arr[i]);
printf("Enter key to search: ");
scanf("%d", &key);
for(i = 0; i < size; i++) {
    if(arr[i] == key) {
        index = i;
        break;
    }
}
if(index != -1)
    printf("Key found at index: %d\n", index);
else
    printf("Key not found\n");
return 0;
}

```

Output

```

d:\BSc CSIT\DAA\dabbal>a.exe
Enter array size: 10
Enter array elements: 10 5 6 8 9 15 60 85 90 45
Enter key to search: 15
Key found at index: 5

```

6. Implementing Bubble Sort

Concept

- Bubble sort repeatedly swaps adjacent elements if they are in the wrong order.
- Best case: $O(n)$ (already sorted), Worst/Average case: $O(n^2)$.
- Space Complexity = $O(1)$

Program

```
#include <stdio.h>

int main() {
    int arr[50], size, i, j, temp;
    printf("Enter array size: ");
    scanf("%d", &size);
    printf("Enter array elements: ");
    for(i = 0; i < size; i++)
        scanf("%d", &arr[i]);
    for(i = 0; i < size-1; i++) {
        for(j = 0; j < size-i-1; j++) {
            if(arr[j] > arr[j+1]) {
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
    printf("Bubble Sorted Array: ");
    for(i = 0; i < size; i++) printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

Output

```
d:\BSc CSIT\DAA\dabbal>gcc bubblesort.c
```

```
d:\BSc CSIT\DAA\dabbal>a.exe
```

```
Enter array size: 10
```

```
Enter array elements: 10 5 6 80 90 45 55 65 15 3
```

```
Bubble Sorted Array: 3 5 6 10 15 45 55 65 80 90
```

7. Implementing Selection Sort

Concept

- Selection sort repeatedly selects the minimum element from unsorted part and swaps it with the first unsorted element.
- Time complexity always $O(n^2)$,
- Space complexity $O(1)$.

Program

```
#include <stdio.h>

int main() {
    int arr[50], size, i, j, min_idx, temp;
    printf("Enter array size: ");
    scanf("%d", &size);
    printf("Enter array elements: ");
    for(i = 0; i < size; i++) scanf("%d", &arr[i]);

    for(i = 0; i < size-1; i++) {
        min_idx = i;
        for(j = i+1; j < size; j++) {
            if(arr[j] < arr[min_idx]) min_idx = j;
        }
        temp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = temp;
    }

    printf("Selection Sorted Array: ");
```

```

    for(i = 0; i < size; i++) printf("%d ", arr[i]);
    printf("\n");
    return 0;
}

```

Output

```

d:\BSc CSIT\DAA\dabbal>gcc selectionsort.c

d:\BSc CSIT\DAA\dabbal>a.exe
Enter array size: 10
Enter array elements: 5 6 78 45 68 97 30 55 61 23
Selection Sorted Array: 5 6 23 30 45 55 61 68 78 97

```

8. Implementing Insertion Sort

Concept

- Insertion sort builds the sorted portion one element at a time by inserting each element into its correct position.
- Efficient for nearly sorted arrays.
- Best case: $O(n)$
- Worst/Average case: $O(n^2)$.
- Space Complexity = $O(1)$

Program

```

#include <stdio.h>

int main() {
    int arr[50], size, i, j, key_val;
    printf("Enter array size: ");
    scanf("%d", &size);
    printf("Enter array elements: ");
    for(i = 0; i < size; i++) scanf("%d", &arr[i]);
}

```

```

for(i = 1; i < size; i++) {
    key_val = arr[i];
    j = i - 1;
    while(j >= 0 && arr[j] > key_val) {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = key_val;
}

printf("Insertion Sorted Array: ");
for(i = 0; i < size; i++) printf("%d ", arr[i]);
printf("\n");
return 0;
}

```

Output

```

d:\BSc CSIT\DAA\dabbal>gcc insertionsort.c

d:\BSc CSIT\DAA\dabbal>a.exe
Enter array size: 10
Enter array elements: 10 5 6 8 90 45 66 78 95 40
Insertion Sorted Array: 5 6 8 10 40 45 66 78 90 95

```

Conclusion

Fibonacci, GCD, sequential search, and bubble sort, selection sort and insertion sorting algorithms were successfully implemented iteratively. Iterative methods are memory-efficient compared to recursion. All the algorithms executed correctly. Bubble, selection, and insertion sort have different efficiencies depending on the array.