



**SCHOOL OF
COMPUTING**

M DHEERAJ VAMSI KRISHNA

CH.SC.U4CSE24127

WEEK 2

Design and Analysis of Algorithm(23CSE211)

Bubble sort

Code :

```
GNU nano 7.2
// ch.sc.u4cse24127
#include <stdio.h>

int main() {
    int n1, i1, j1, t1;
    int v1[100];

    printf("ch.sc.u4cse24127\n");

    printf("Enter number of elements: ");
    scanf("%d", &n1);

    printf("Enter elements:\n");
    for (i1 = 0; i1 < n1; i1++) {
        scanf("%d", &v1[i1]);
    }

    for (i1 = 0; i1 < n1 - 1; i1++) {
        for (j1 = 0; j1 < n1 - i1 - 1; j1++) {
            if (v1[j1] > v1[j1 + 1]) {
                t1 = v1[j1];
                v1[j1] = v1[j1 + 1];
                v1[j1 + 1] = t1;
            }
        }
    }

    printf("Sorted array:\n");
    for (i1 = 0; i1 < n1; i1++) {
        printf("%d ", v1[i1]);
    }

    return 0;
}
```

Output:

```
dheeraj@DheerajG15:~$ nano bubble.c
dheeraj@DheerajG15:~$ gcc bubble.c -o bubble
dheeraj@DheerajG15:~$ ./bubble
ch.sc.u4cse24127
Enter number of elements: 5
Enter elements:
2 1 4 3 6
Sorted array:
1 2 3 4 6 dheeraj@DheerajG15:~$
```

Time Complexity Justification

Bubble sort uses two nested loops.

Each element is compared with the remaining elements.

Therefore, the number of comparisons increases as $n \times n$.

Time Complexity:

Best Case: $O(n)$

Average Case: $O(n^2)$

Worst Case: $O(n^2)$

Space Complexity Justification

Sorting is done within the same array.

Only one temporary variable is used for swapping.

No extra memory is required based on input size.

Space Complexity: $O(1)$

Insertion sort :

Code :

```
// ch.sc.u4cse24127
#include <stdio.h>

int main() {
    int n1, i1, j1, k1;
    int a1[100];

    printf("ch.sc.u4cse24127\n");

    printf("Enter number of elements: ");
    scanf("%d", &n1);

    printf("Enter elements:\n");
    for (i1 = 0; i1 < n1; i1++) {
        scanf("%d", &a1[i1]);
    }

    for (i1 = 1; i1 < n1; i1++) {
        k1 = a1[i1];
        j1 = i1 - 1;

        while (j1 >= 0 && a1[j1] > k1) {
            a1[j1 + 1] = a1[j1];
            j1--;
        }
        a1[j1 + 1] = k1;
    }

    printf("Sorted array:\n");
    for (i1 = 0; i1 < n1; i1++) {
        printf("%d ", a1[i1]);
    }

    return 0;
}
```

Output:

```
dheeraj@DheerajG15:~$ nano bubble.c
dheeraj@DheerajG15:~$ nano insertion.c
dheeraj@DheerajG15:~$ gcc insertion.c -o insertion
dheeraj@DheerajG15:~$ ./insertion
ch.sc.u4cse24127
Enter number of elements: 6
Enter elements:
7 4 3 2 5 8
Sorted array:
2 3 4 5 7 8 dheeraj@DheerajG15:~$ |
```

time Complexity

Best Case: $O(n)$

Average Case: $O(n^2)$

Worst Case: $O(n^2)$

Time Complexity Justification

Elements are compared and shifted using nested loops.

In the worst case, each element is compared with all previous elements.

Space Complexity

Space Complexity: $O(1)$

Space Complexity Justification

Sorting is done in the same array.

Only one extra variable is used for shifting elements.

Selection Sort:

Code :

```
// ch.sc.u4cse24127
#include <stdio.h>

int main() {
    int n2, i2, j2, p2, t2;
    int a2[100];

    printf("ch.sc.u4cse24127\n");

    printf("Enter number of elements: ");
    scanf("%d", &n2);

    printf("Enter elements:\n");
    for (i2 = 0; i2 < n2; i2++) {
        scanf("%d", &a2[i2]);
    }

    for (i2 = 0; i2 < n2 - 1; i2++) {
        p2 = i2;
        for (j2 = i2 + 1; j2 < n2; j2++) {
            if (a2[j2] < a2[p2]) {
                p2 = j2;
            }
        }
        t2 = a2[i2];
        a2[i2] = a2[p2];
        a2[p2] = t2;
    }

    printf("Sorted array:\n");
    for (i2 = 0; i2 < n2; i2++) {
        printf("%d ", a2[i2]);
    }

    return 0;
}
```

Output:

```
dheeraj@DheerajG15:~$ nano insertion.c
dheeraj@DheerajG15:~$ nano selection.c
dheeraj@DheerajG15:~$ gcc selection.c -o selection
dheeraj@DheerajG15:~$ ./selection
ch.sc.u4cse24127
Enter number of elements: 4
Enter elements:
6 9 5 1
Sorted array:
1 5 6 9 dheeraj@DheerajG15:~$ |
```

Time Complexity

Best Case: $O(n^2)$

Average Case: $O(n^2)$

Worst Case: $O(n^2)$

Time Complexity Justification

Two nested loops are used to find the minimum element.

The number of comparisons remains the same for all cases.

Space Complexity

Space Complexity: $O(1)$

Space Complexity Justification

Sorting is performed in the original array.

Only one temporary variable is used for swapping.

Bucket sort:

Code:

```
// ch.sc.u4cse24127
#include <stdio.h>

int main() {
    int n1, i1, j1, k1;
    int a1[100], b1[10][100], c1[10];

    printf("ch.sc.u4cse24127\n");

    printf("Enter number of elements: ");
    scanf("%d", &n1);

    printf("Enter elements (0-99):\n");
    for (i1 = 0; i1 < n1; i1++) {
        scanf("%d", &a1[i1]);
    }

    for (i1 = 0; i1 < 10; i1++)
        c1[i1] = 0;

    for (i1 = 0; i1 < n1; i1++) {
        k1 = a1[i1] / 10;
        b1[k1][c1[k1]++] = a1[i1];
    }

    k1 = 0;
    for (i1 = 0; i1 < 10; i1++) {
        for (j1 = 0; j1 < c1[i1]; j1++) {
            a1[k1++] = b1[i1][j1];
        }
    }

    printf("Sorted array:\n");
    for (i1 = 0; i1 < n1; i1++)
        printf("%d ", a1[i1]);

    return 0;
}
```

Output:

```
dheeraj@DheerajG15:~$ nano bucket.c
dheeraj@DheerajG15:~$ gcc bucket.c -o bucket
dheeraj@DheerajG15:~$ ./bucket
ch.sc.u4cse24127
Enter number of elements: 5
Enter elements (0-99):
66 88 45 23 16
Sorted array:
16 23 45 66 88 dheeraj@DheerajG15:~$ |
```

Time Complexity

Best Case: $O(n)$

Average Case: $O(n + k)$

Worst Case: $O(n^2)$

Time Complexity Justification

Elements are distributed into buckets.

Sorting inside buckets may take more time in worst case.

Space Complexity

Space Complexity: $O(n + k)$

Space Complexity Justification

Extra buckets are used to store elements.

Heap sort:

Code:

```
// ch.sc.u4cse24127
#include <stdio.h>

void h1(int a1[], int n1, int i1) {
    int l1, r1, m1, t1;
    m1 = i1;
    l1 = 2 * i1 + 1;
    r1 = 2 * i1 + 2;

    if (l1 < n1 && a1[l1] > a1[m1])
        m1 = l1;
    if (r1 < n1 && a1[r1] > a1[m1])
        m1 = r1;

    if (m1 != i1) {
        t1 = a1[i1];
        a1[i1] = a1[m1];
        a1[m1] = t1;
        h1(a1, n1, m1);
    }
}

int main() {
    int n2, i2, t2;
    int a2[100];

    printf("ch.sc.u4cse24127\n");

    printf("Enter number of elements: ");
    scanf("%d", &n2);

    printf("Enter elements:\n");
    for (i2 = 0; i2 < n2; i2++) {
        scanf("%d", &a2[i2]);
    }

    for (i2 = n2 / 2 - 1; i2 >= 0; i2--)
        h1(a2, n2, i2);

    for (i2 = n2 - 1; i2 > 0; i2--) {
        t2 = a2[0];
        a2[0] = a2[i2];
        a2[i2] = t2;
        h1(a2, i2, 0);
    }

    printf("Sorted array:\n");
    for (i2 = 0; i2 < n2; i2++)
        printf("%d ", a2[i2]);

    return 0;
}
```

Output:

```
dheeraj@DheerajG15:~$ nano heap.c
dheeraj@DheerajG15:~$ gcc heap.c -o heap
dheeraj@DheerajG15:~$ ./heap
ch.sc.u4cse24127
Enter number of elements: 4
Enter elements:
7 9 5 2
Sorted array:
2 5 7 9 dheeraj@DheerajG15:~$ |
```

Time Complexity

Best Case: $O(n \log n)$

Average Case: $O(n \log n)$

Worst Case: $O(n \log n)$

Time Complexity Justification

Heap creation takes linear time.

Each deletion takes logarithmic time.

Space Complexity

Space Complexity: $O(1)$

Space Complexity Justification

Sorting is done in the same array.

No extra memory is used.

BFS

Code:

```
// ch.sc.u4cse24127
#include <stdio.h>

int main() {
    int v1, e1, i1, j1, s1;
    int q1[100], f1 = 0, r1 = -1;
    int a1[100][100] = {0};
    int v2[100] = {0};

    printf("ch.sc.u4cse24127\n");

    printf("Enter number of vertices: ");
    scanf("%d", &v1);

    printf("Enter number of edges: ");
    scanf("%d", &e1);

    printf("Enter edges (u v):\n");
    for (i1 = 0; i1 < e1; i1++) {
        scanf("%d %d", &j1, &s1);
        a1[j1][s1] = 1;
        a1[s1][j1] = 1;
    }

    printf("Enter starting vertex: ");
    scanf("%d", &s1);

    q1[++r1] = s1;
    v2[s1] = 1;

    printf("BFS Traversal: ");
    while (f1 <= r1) {
        j1 = q1[f1++];
        printf("%d ", j1);

        for (i1 = 0; i1 < v1; i1++) {
            if (a1[j1][i1] == 1 && v2[i1] == 0) {
                q1[++r1] = i1;
                v2[i1] = 1;
            }
        }
    }

    return 0;
}
```

Output:

```
dheeraj@DheerajG15:~$ nano bfs.c
dheeraj@DheerajG15:~$ gcc bfs.c -o bfs
dheeraj@DheerajG15:~$ nano bfs.c
dheeraj@DheerajG15:~$ gcc bfs.c -o bfs
dheeraj@DheerajG15:~$ ./bfs
ch.sc.u4cse24127
Enter number of vertices: 5
Enter number of edges: 6
Enter edges (u v):
0 1
0 2
1 3
1 4
2 4
3 4
Enter starting vertex: 0
BFS Traversal: 0 1 2 3 4 dheeraj@DheerajG15:~$ |
```

Time Complexity

Time Complexity: $O(V + E)$

Time Complexity Justification

Each vertex is visited once.

Each edge is checked once.

Space Complexity

Space Complexity: $O(V)$

Space Complexity Justification

Queue and visited array store vertices.

DFS

Code:

```
// ch.sc.u4cse24127
#include <stdio.h>

int a2[100][100], v3[100], v4;

void d1(int s2) {
    int i2;
    printf("%d ", s2);
    v3[s2] = 1;

    for (i2 = 0; i2 < v4; i2++) {
        if (a2[s2][i2] == 1 && v3[i2] == 0) {
            d1(i2);
        }
    }
}

int main() {
    int e2, i2, x2, y2, s2;

    printf("ch.sc.u4cse24127\n");

    printf("Enter number of vertices: ");
    scanf("%d", &v4);

    printf("Enter number of edges: ");
    scanf("%d", &e2);

    printf("Enter edges (u v):\n");
    for (i2 = 0; i2 < e2; i2++) {
        scanf("%d %d", &x2, &y2);
        a2[x2][y2] = 1;
        a2[y2][x2] = 1;
    }

    printf("Enter starting vertex: ");
    scanf("%d", &s2);

    printf("DFS Traversal: ");
    d1(s2);

    return 0;
}
```

Output:

```
dheeraj@DheerajG15:~$ nano dfs.c
dheeraj@DheerajG15:~$ gcc dfs.c -o dfs
dheeraj@DheerajG15:~$ ./dfs
ch.sc.u4cse24127
Enter number of vertices: 5
Enter number of edges: 6
Enter edges (u v):
0 1
0 2
1 3
1 4
2 4
3 4
Enter starting vertex: 0
DFS Traversal: 0 1 3 4 2 dheeraj@DheerajG15:~$ |
```

Time Complexity

Time Complexity: $O(V + E)$

Time Complexity Justification

Each vertex and edge is visited once.

Space Complexity

Space Complexity: $O(V)$

Space Complexity Justification

Stack and visited array store vertices.