



AMRITA

VISHWA VIDYAPEETHAM

DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK

WEEK - 2

NAME : S V ADITYA

ROLL NUMBER : CH.SC.U4CSE24256

CLASS : CSE-C

Question 1:BUBBLE SORT

CODE:

```
C bubblesort.c > ...
1 //CH.SC.U4CSE24140 - Rohith S
2 #include<stdio.h>
3 int main(){
4     int i,j,size,temp;
5     printf("Enter the size of your array: ");
6     scanf("%d",&size);
7     printf("Enter the elements of your array: ");
8     int arr[size];
9     printf("\n");
10    for(i=0;i<size;i++){
11        printf("Enter element %d: ",i+1);
12        scanf("%d",&arr[i]);
13    }
14    for(i=0;i<size-1;i++){
15        for(j=0;j<size-1-i;j++){
16            if(arr[j]>arr[j+1]){
17                temp=arr[j];
18                arr[j]=arr[j+1];
19                arr[j+1]=temp;
20            }
21        }
22    }
23    printf("Sorted array\n");
24    for(i=0;i<size;i++){
25        printf("%d ",arr[i]);
26    }
27    return 0;
28 }
```

OUTPUT:

```
Enter the elements of your array:
Enter element 1: 8
Enter element 2: 2
Enter element 3: 1
Enter element 4: 6
Enter element 5: 2
Enter element 6: 5
Sorted array
1 2 2 5 6 8
```

SPACE AND TIME COMPLEXITY

JUSTIFICATION:

Space Complexity: O(1)

Variables in main() :

- int i → 4 bytes
- int j → 4 bytes
- int size → 4 bytes
- int temp → 4 bytes

Array used:

- int arr[size] → size × 4 bytes
- Total memory used by variables (excluding input array):
- $4 + 4 + 4 + 4 = 16$ bytes

Time Complexity: O(n^2)

• **Outer loop:**

Runs $(n - 1)$ times

• **Inner loop:**

Runs $(n - 1 - i)$ times for each outer loop iteration

• **Total number of comparisons:**

$$(n - 1) + (n - 2) + \dots + 1$$

$$= n(n - 1) / 2$$

Overall time complexity:

- $O(n^2)$

Question 2: INSERTION SORT

CODE:

```
C insertionsort.c > main()
1 #include<stdio.h>
2 int main(){
3     int i,j,size,key;
4     printf("Enter the size of your array: ");
5     scanf("%d", &size);
6     int arr[size];
7     printf("Enter the elements of your array:\n");
8     for (i = 0; i < size; i++) {
9         printf("Enter element %d: ", i + 1);
10        scanf("%d", &arr[i]);
11    }
12    for(i=1;i<size;i++){
13        key=arr[i];
14        j=i-1;
15
16        while(j>=0&&arr[j]>key){
17            arr[j+1]=arr[j];
18            j--;
19        }
20        arr[j+1]=key;
21    }
22    printf("Sorted array:\n");
23    for (i = 0; i < size; i++) {
24        printf("%d ", arr[i]);
25    }
26
27    return 0;
28 }
```

OUTPUT:

```
PS C:\Users\123sr\OneDrive\Desktop\COLLEGE STUFF\SEM IV\DAA LAB\Week 2>
Enter element 1: 59
Enter element 2: 20
Enter element 3: 50
Enter element 4: 201
Enter element 5: 50
Sorted array:
20 50 50 59 201
```

TIME AND SPACE COMPLEXITY JUSTIFICATION:

Space Complexity: $O(1)$

Variables in main() :

- int i → 4 bytes
- int j → 4 bytes
- int size → 4 bytes
- int key → 4 bytes

• Array used:

- int arr[size] → size × 4 bytes

• Total memory used by variables (excluding input array):

$$• 4 + 4 + 4 + 4 = 16 \text{ bytes}$$

Time Complexity: $O(n^2)$

Outer loop:

- Runs $(n - 1)$ times

Inner while loop:

- In the worst case, runs up to i times for each iteration

• Total number of comparisons (worst case):

$$\begin{aligned} &• 1 + 2 + 3 + \dots + (n - 1) \\ &• = n(n - 1) / 2 \end{aligned}$$

• Overall time complexity: $O(n^2)$

Question 3:

CODE:

```
C selectionsort.c > main()
1 //CH.SC.U4CSE24140 - Rohith S
2 #include <stdio.h>
3 int main() {
4     int i, j, size, min, temp;
5     printf("Enter the size of your array: ");
6     scanf("%d", &size);
7     int arr[size];
8     printf("Enter the elements of your array:\n");
9     for (i = 0; i < size; i++) {
10         printf("Enter element %d: ", i + 1);
11         scanf("%d", &arr[i]);
12     }
13     for (i = 0; i < size - 1; i++) {
14         min = i;
15         for (j = i + 1; j < size; j++) {
16             if (arr[j] < arr[min]) {
17                 min = j;
18             }
19         }
20         temp = arr[i];
21         arr[i] = arr[min];
22         arr[min] = temp;
23     }
24     printf("Sorted array:\n");
25     for (i = 0; i < size; i++) {
26         printf("%d ", arr[i]);
27     }
28     return 0;
29 }
```

OUTPUT:

```
Enter the size of your array: 5
Enter the elements of your array:
Enter element 1: 10
Enter element 2: 200
Enter element 3: 30
Enter element 4: 500
Enter element 5: 1
Sorted array:
1 10 30 200 500
```

TIME AND SPACE COMPLEXITY JUSTIFICATION:

Space Complexity: O(1)

Variables in main() :

- int i → 4 bytes
- int j → 4 bytes
- int size → 4 bytes
- int min → 4 bytes
- int temp → 4 bytes

Array used:

- int arr[size] → size × 4 bytes

Total memory used by variables (excluding input array):

$$\bullet 4 + 4 + 4 + 4 + 4 = 20 \text{ bytes}$$

Auxiliary space:

- Constant → O(1)
-

Time Complexity: O(n²)**Outer loop:**

- Runs (n - 1) times

Inner loop:

- Runs (n - 1 - i) times for each outer loop iteration

Total number of comparisons:

$$\bullet (n - 1) + (n - 2) + \dots + 1$$

$$\bullet = n(n - 1) / 2$$

$$O(n^2)$$

Question 4:BUCKET SORT**CODE:**

```
C bucketsort.c > ↴ main()
1 //CH.SC.U4CSE24140 - Rohith S
2 #include <stdio.h>
3 int main() {
4     int i, j, size;
5     printf("Enter the size of your array: ");
6     scanf("%d", &size);
7     int arr[size];
8     printf("Enter the elements of your array:\n");
9     for (i = 0; i < size; i++) {
10         printf("Enter element %d: ", i + 1);
11         scanf("%d", &arr[i]);
12     }
13     int bucket[10][size];
14     int count[10];
15     for (i = 0; i < 10; i++) {
16         count[i] = 0;
17     }
18     for (i = 0; i < size; i++) {
19         int index = arr[i] / 10;
20         bucket[index][count[index]++] = arr[i];
21     }
22     for (i = 0; i < 10; i++) {
23         for (j = 1; j < count[i]; j++) {
24             int key = bucket[i][j];
25             int k = j - 1;
26             while (k >= 0 && bucket[i][k] > key) {
27                 bucket[i][k + 1] = bucket[i][k];
28                 k--;
29             }
30             bucket[i][k + 1] = key;
31         }
32     }
33     int index = 0;
34     for (i = 0; i < 10; i++){
35         for (j=0;j<count[i];j++) {
36             arr[index++]=bucket[i][j];
37         }
38     }
39     printf("Sorted array:\n");
40     for (i=0;i<size;i++) {
41         printf("%d ",arr[i]);
42     }
43     return 0;
44 }
45
```

OUTPUT:

```
Enter the size of your array: 5
Enter the elements of your array:
Enter element 1: 20 50 10 40 60
Enter element 2: Enter element 3: Enter element 4: Enter element 5: Sorted array:
10 20 40 50 60
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: $O(n + k)$

Variables in main() :

- int i → 4 bytes
- int j → 4 bytes
- int size → 4 bytes

Buckets used:

- k buckets (arrays or lists)

Array used:

- int arr[size] → size × 4 bytes

Auxiliary space:

- Buckets require extra memory proportional to number of elements and buckets

Overall space usage:

- Input array → $O(n)$
- Buckets → $O(k)$

Total space complexity:

- $O(n + k)$

Time Complexity

Distribution into buckets:

- Each element placed into a bucket once
- Time → $O(n)$

Overall time complexity:

- Best / Average → $O(n + k)$
- Worst → $O(n^2)$

Question 5:HEAP SORT

CODE:

```
c heapsort.c > main()
1 //CH.SC.U4CSE24140 - Rohith S
2 #include <stdio.h>
3 int main() {
4     int i, j, size, temp;
5     printf("Enter the size of your array: ");
6     scanf("%d", &size);
7     int arr[size];
8     printf("Enter the elements of your array:\n");
9     for (i = 0; i < size; i++) {
10         printf("Enter element %d: ", i + 1);
11         scanf("%d", &arr[i]);
12     }
13     for (i = size / 2 - 1; i >= 0; i--) {
14         int parent = i;
15         while (1) {
16             int left = 2 * parent + 1;
17             int right = 2 * parent + 2;
18             int largest = parent;
19             if (left < size && arr[left] > arr[largest])
20                 largest = left;
21             if (right < size && arr[right] > arr[largest])
22                 largest = right;
23             if (largest != parent) {
24                 temp = arr[parent];
25                 arr[parent] = arr[largest];
26                 arr[largest] = temp;
27                 parent = largest;
28             } else {
29                 break;
30             }
31         }
32     }
33     for (i = size - 1; i > 0; i--) {
34         temp = arr[0];
35         arr[0] = arr[i];
36         arr[i] = temp;
37         int parent = 0;
38         while (1) {
39             int left = 2 * parent + 1;
40             int right = 2 * parent + 2;
41             int largest = parent;
42             if (left < i && arr[left] > arr[largest])
43                 largest = left;
44             if (right < i && arr[right] > arr[largest])
45                 largest = right;
46             if (largest != parent) {
47                 temp = arr[parent];
48                 arr[parent] = arr[largest];
49                 arr[largest] = temp;
50                 parent = largest;
51             } else {
52                 break;
53             }
54         }
55     }
56     printf("Sorted array:\n");
57     for (i = 0; i < size; i++) {
58         printf("%d ", arr[i]);
59     }
60 }
61 }
```

OUTPUT:

```
Enter the size of your array: 5
Enter the elements of your array:
Enter element 1: 10 20 50 20 15
Enter element 2: Enter element 3: Enter element 4: Enter element 5: Sorted array:
10 15 20 20 50
PS C:\Users\123sr\OneDrive\Desktop\COLLEGE STUFF\SEM IV\DAA LAB\Week 2> █
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: O(1)

Variables in main() :

- int i → 4 bytes
- int j → 4 bytes
- int size → 4 bytes
- int temp → 4 bytes
- int parent → 4 bytes
- int left → 4 bytes
- int right → 4 bytes
- int largest → 4 bytes

Array used:

- int arr[size] → size × 4 bytes

Total memory used by variables (excluding input array):

- $4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 = 32$ bytes

Auxiliary space:

- Constant extra space used
- O(1)

Time Complexity: O(n log n)

Building max heap:

- Heapify runs in O(n) time

Sorting phase:

- $n - 1$ deletions from heap
- Each heapify operation takes O(log n)

Total time:

- $O(n) + O(n \log n)$

Question 6:BFS

CODE

```
C bfs.c > ...
1 //CH.SC.U4CSE24140 - Rohith S
2 #include <stdio.h>
3 int main() {
4     int n, i, j, start;
5     int queue[50], front = 0, rear = 0;
6     int visited[50] = {0};
7     printf("Enter number of vertices: ");
8     scanf("%d", &n);
9     int graph[n][n];
10    printf("Enter adjacency matrix:\n");
11    for (i = 0; i < n; i++) {
12        for (j = 0; j < n; j++) {
13            scanf("%d", &graph[i][j]);
14        }
15    }
16    printf("Enter starting vertex: ");
17    scanf("%d", &start);
18    queue[rear++] = start;
19    visited[start] = 1;
20    printf("BFS traversal: ");
21    while (front < rear) {
22        int v = queue[front++];
23        printf("%d ", v);
24        for (i = 0; i < n; i++) {
25            if (graph[v][i] == 1 && visited[i] == 0) {
26                queue[rear++] = i;
27                visited[i] = 1;
28            }
29        }
30    }
31    return 0;
32 }
```

OUTPUT:

```
Enter number of vertices: 3
Enter adjacency matrix:
9 8 7
6 5 4
3 2 1
Enter starting vertex: 1
BFS traversal: 1
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: $O(n)$

Variables in main() :

- int n → 4 bytes
- int i, j → 8 bytes
- int start → 4 bytes
- int front, rear → 8 bytes

Arrays used:

- int visited[n] → $n \times 4$ bytes
- int queue[n] → $n \times 4$ bytes
- int graph[n][n] → $n^2 \times 4$ bytes

Auxiliary space:

- Queue + visited array → $O(n)$
- Adjacency matrix → $O(n^2)$

Overall space complexity:

- $O(n^2)$ (due to adjacency matrix)

Time Complexity: $O(n^2)$

Traversal logic:

- Each vertex is visited once

Adjacency matrix scan:

- For each vertex, all n vertices are checked

Total operations:

- $n \times n$ comparisons

Question 7:DFS

CODE:

```

C dfs.c > main()
1 //CH.SC.U4CSE24140 - Rohith S
2 #include <stdio.h>
3 int n;
4 int graph[50][50];
5 int visited[50];
6 void dfs(int v) {
7     int i;
8     printf("%d ", v);
9     visited[v] = 1;
10
11    for (i = 0; i < n; i++) {
12        if (graph[v][i] == 1 && visited[i] == 0) {
13            dfs(i);
14        }
15    }
16}
17 int main() {
18    int i, j, start;
19    printf("Enter number of vertices: ");
20    scanf("%d", &n);
21    printf("Enter adjacency matrix:\n");
22    for (i = 0; i < n; i++) {
23        for (j = 0; j < n; j++) {
24            scanf("%d", &graph[i][j]);
25        }
26    }
27    printf("Enter starting vertex: ");
28    scanf("%d", &start);
29    for (i = 0; i < n; i++) {
30        visited[i] = 0;
31    }
32    printf("DFS traversal: ");
33    dfs(start);
34    return 0;
35}

```

OUTPUT:

```

Enter number of vertices: 3
Enter adjacency matrix:
9 8 7 6 5 4 3 2 1
Enter starting vertex: 6
DFS traversal: 6

```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: O(n²)

Variables used:

- int $n \rightarrow 4$ bytes
- int $i, j \rightarrow 8$ bytes
- int start $\rightarrow 4$ bytes

Arrays used:

- int visited[n] $\rightarrow n \times 4$ bytes
- int graph[n][n] $\rightarrow n^2 \times 4$ bytes

Recursion stack:

- Maximum depth = n
- Stack space $\rightarrow O(n)$

Overall space complexity:

- $O(n^2)$ (dominant adjacency matrix)

Time Complexity: $O(n^2)$ **Traversal logic:**

- Each vertex visited once

Adjacency matrix scan:

- For each vertex, all n vertices are checked

Total operations:

- $n \times n$ comparisons