



AMRITA

VISHWA VIDYAPEETHAM

DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK

WEEK - 2

NAME : B S CHENTHIL HARI

ROLL NUMBER : CH.SC.U4CSE24103

CLASS : CSE-B

Question 1: BUBBLE SORT

CODE:

```
(globals)
bubble.cpp
1 //CH.SC.U4CSE24103
2 #include <stdio.h>
3 int main() {
4     int n, i, j, temp;
5     int arr[100];
6
7     printf("Enter number of elements: ");
8     scanf("%d", &n);
9
10    printf("Enter the elements:\n");
11    for(i = 0; i < n; i++) {
12        scanf("%d", &arr[i]);
13    }
14
15    // Bubble Sort
16    for(i = 0; i < n - 1; i++) {
17        for(j = 0; j < n - i - 1; j++) {
18            if(arr[j] > arr[j + 1]) {
19                temp = arr[j];
20                arr[j] = arr[j + 1];
21                arr[j + 1] = temp;
22            }
23        }
24    }
25
26    printf("Sorted array:\n");
27    for(i = 0; i < n; i++) {
28        printf("%d ", arr[i]);
29    }
30
31    return 0;
32 }
33
```

OUTPUT:

```
C:\Users\chent\OneDrive - An  X + v
Enter number of elements: 5
Enter the elements:
3
4
5
7
2
Sorted array:
2 3 4 5 7
-----
Process exited after 4.599 seconds with return value 0
Press any key to continue . . . |
```

SPACE AND TIME COMPLEXITY JUSTIFICATION:

Space Complexity: $O(1)$

Variables in main() :

- $\text{int } i \rightarrow 4 \text{ bytes}$
- $\text{int } j \rightarrow 4 \text{ bytes}$
- $\text{int size} \rightarrow 4 \text{ bytes}$
- $\text{int temp} \rightarrow 4 \text{ bytes}$

Array used:

- $\text{int arr[size]} \rightarrow \text{size} \times 4 \text{ bytes}$
- 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 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:

```
1 //CH.SC.U4CSE24103
2 #include <stdio.h>
3
4 int main() {
5     int n, i, key, j;
6     int arr[100];
7
8     printf("Enter number of elements: ");
9     scanf("%d", &n);
10
11     printf("Enter the elements:\n");
12     for (i = 0; i < n; i++) {
13         scanf("%d", &arr[i]);
14     }
15
16     // Insertion Sort
17     for (i = 1; i < n; i++) {
18         key = arr[i];
19         j = i - 1;
20
21         while (j >= 0 && arr[j] > key) {
22             arr[j + 1] = arr[j];
23             j--;
24         }
25         arr[j + 1] = key;
26     }
27
28     printf("Sorted array:\n");
29     for (i = 0; i < n; i++) {
30         printf("%d ", arr[i]);
31     }
32
33     return 0;
34 }
35
```

OUTPUT:

```
C:\Users\chent\OneDrive - An  X + v
Enter number of elements: 5
Enter the elements:
6
4
21
3
5
Sorted array:
3 4 5 6 21
-----
Process exited after 4.196 seconds with return value 0
Press any key to continue . . . |
```

TIME AND SPACE COMPLEXITY JUSTIFICATION:

Space Complexity: $O(1)$

Variables in main() :

- $\text{int } i \rightarrow 4 \text{ bytes}$
- $\text{int } j \rightarrow 4 \text{ bytes}$
- $\text{int size} \rightarrow 4 \text{ bytes}$
- $\text{int key} \rightarrow 4 \text{ bytes}$

• Array used:

- $\text{int arr[size]} \rightarrow \text{size} \times 4 \text{ bytes}$

• 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):

- $1 + 2 + 3 + \dots + (n - 1)$
- $= n(n - 1) / 2$

• Overall time complexity: $O(n^2)$

Question 3:

CODE:

```
bubble.cpp ×
1 //CH.SC.U4CSE24103
2 #include <stdio.h>
3
4 int main() {
5     int i, j, size, min, temp;
6
7     printf("Enter the size of your array: ");
8     scanf("%d", &size);
9
10    int arr[size];
11
12    printf("Enter the elements of your array:\n");
13    for (i = 0; i < size; i++) {
14        printf("Enter element %d: ", i + 1);
15        scanf("%d", &arr[i]);
16    }
17
18    // Selection Sort
19    for (i = 0; i < size - 1; i++) {
20        min = i;
21
22        for (j = i + 1; j < size; j++) {
23            if (arr[j] < arr[min]) {
24                min = j;
25            }
26        }
27
28        // Swap
29        temp = arr[i];
30        arr[i] = arr[min];
31        arr[min] = temp;
32    }
33
34    printf("Sorted array:\n");
35    for (i = 0; i < size; i++) {
36        printf("%d ", arr[i]);
37    }
38
39    return 0;
40 }
41
```

OUTPUT:


```
C:\Users\chent\OneDrive - An  ×  +  ∨  
Enter number of elements: 5  
Enter the elements:  
4  
2  
5  
1  
5  
Sorted array:  
1 2 4 5 5  
-----  
Process exited after 4.401 seconds with return value 0  
Press any key to continue . . . |
```

TIME AND SPACE COMPLEXITY JUSTIFICATION:

Space Complexity: $O(1)$

Variables in main() :

- $\text{int } i \rightarrow 4 \text{ bytes}$
- $\text{int } j \rightarrow 4 \text{ bytes}$
- $\text{int size} \rightarrow 4 \text{ bytes}$
- $\text{int min} \rightarrow 4 \text{ bytes}$
- $\text{int temp} \rightarrow 4 \text{ bytes}$

Array used:

- $\text{int arr[size]} \rightarrow \text{size} \times 4 \text{ bytes}$

Total memory used by variables (excluding input array):

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

Auxiliary space:

- Constant $\rightarrow O(1)$
-

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:

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

$O(n^2)$

Question 4: BUCKET SORT

CODE:

```
1 // CH.SC.U4CSE24103
2 #include <stdio.h>
3
4 #define MAX 100
5 #define BUCKETS 10
6
7 void insertionSort(int bucket[], int n) {
8     int i, key, j;
9     for (i = 1; i < n; i++) {
10         key = bucket[i];
11         j = i - 1;
12
13         while (j >= 0 && bucket[j] > key) {
14             bucket[j + 1] = bucket[j];
15             j--;
16         }
17         bucket[j + 1] = key;
18     }
19 }
20
21 int main() {
22     int n, i, j;
23     int arr[MAX];
24
25     printf("Enter number of elements (max %d): ", MAX);
26     scanf("%d", &n);
27
28     if (n <= 0 || n > MAX) {
29         printf("Invalid array size!\n");
30         return 1;
31     }
32
33     printf("Enter elements (0-99 only):\n");
34     for (i = 0; i < n; i++) {
35         scanf("%d", &arr[i]);
36
37         if (arr[i] < 0 || arr[i] > 99) {
38             printf("Invalid input! Elements must be between 0 and 99.\n");
39             return 1;
40         }
41     }
42
43     int bucket[BUCKETS][MAX];
44     int count[BUCKETS] = {0};
45
46     // Distribute elements into buckets
47     for (i = 0; i < n; i++) {
48         int index = arr[i] / 10; // 0-9
49         bucket[index][count[index]++] = arr[i];
50     }
51
52     // Sort each bucket
53     for (i = 0; i < BUCKETS; i++) {
54         insertionSort(bucket[i], count[i]);
55     }
56
57     // Merge buckets
58     int k = 0;
59     for (i = 0; i < BUCKETS; i++) {
60         for (j = 0; j < count[i]; j++) {
61             arr[k++] = bucket[i][j];
62         }
63     }
64
65     printf("Sorted array:\n");
66     for (i = 0; i < n; i++) {
67         printf("%d ", arr[i]);
68     }
69
70     return 0;
71 }
```

OUTPUT:

```
C:\Users\chent\OneDrive - An  × + v
Enter number of elements (max 100): 5
Enter elements (0-99 only):
23
4
6
7
3
Sorted array:
3 4 6 7 23
-----
Process exited after 5.972 seconds with return value 0
Press any key to continue . . . |
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: $O(n + k)$

Variables in main() :

- int i \rightarrow 4 bytes
- int j \rightarrow 4 bytes
- int size \rightarrow 4 bytes

Buckets used:

- k buckets (arrays or lists)

Array used:

- int arr[size] \rightarrow size \times 4 bytes

Auxiliary space:

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

Overall space usage:

- Input array $\rightarrow O(n)$
- Buckets $\rightarrow O(k)$

Total space complexity:

- $O(n + k)$

Time Complexity

Distribution into buckets:

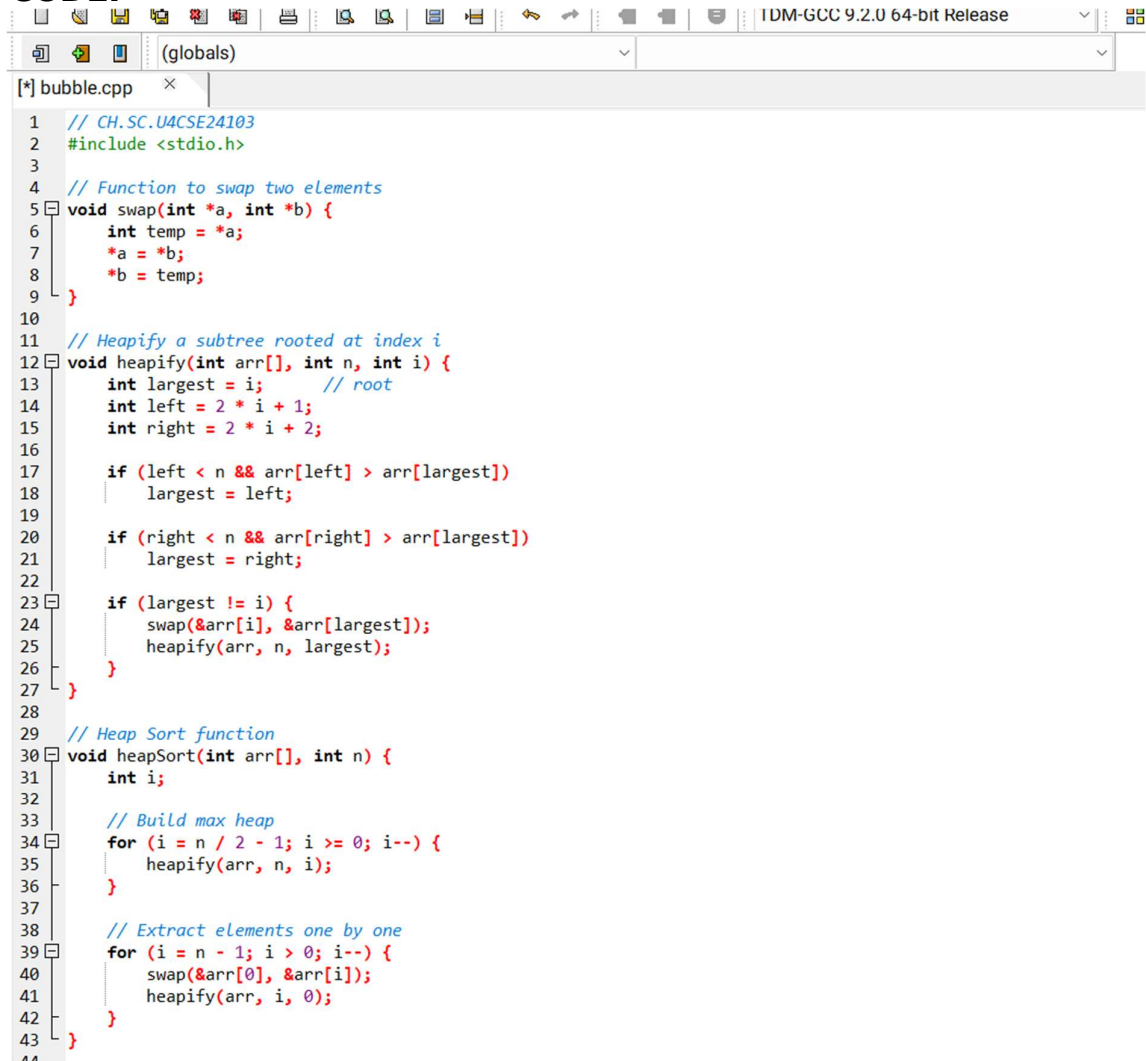
- Each element placed into a bucket once
- Time $\rightarrow O(n)$

Overall time complexity:

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

Question 5: HEAP SORT

CODE:



The screenshot shows a code editor window titled "bubble.cpp" with a toolbar at the top. The code implements the Heap Sort algorithm. It includes a swap function, a heapify function, and a main heapSort function. The code is written in C++ and uses standard library headers. The editor's status bar at the top right indicates "IDM-GCC 9.2.0 64-bit Release".

```
1 // CH.SC.U4CSE24103
2 #include <stdio.h>
3
4 // Function to swap two elements
5 void swap(int *a, int *b) {
6     int temp = *a;
7     *a = *b;
8     *b = temp;
9 }
10
11 // Heapify a subtree rooted at index i
12 void heapify(int arr[], int n, int i) {
13     int largest = i; // root
14     int left = 2 * i + 1;
15     int right = 2 * i + 2;
16
17     if (left < n && arr[left] > arr[largest])
18         largest = left;
19
20     if (right < n && arr[right] > arr[largest])
21         largest = right;
22
23     if (largest != i) {
24         swap(&arr[i], &arr[largest]);
25         heapify(arr, n, largest);
26     }
27 }
28
29 // Heap Sort function
30 void heapSort(int arr[], int n) {
31     int i;
32
33     // Build max heap
34     for (i = n / 2 - 1; i >= 0; i--) {
35         heapify(arr, n, i);
36     }
37
38     // Extract elements one by one
39     for (i = n - 1; i > 0; i--) {
40         swap(&arr[0], &arr[i]);
41         heapify(arr, i, 0);
42     }
43 }
```

```
int main() {
    int n, i;
    int arr[100];

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

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

    heapSort(arr, n);

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

    return 0;
}
```

OUTPUT:

```
C:\Users\chent\OneDrive - An  X + v
Enter number of elements: 7
Enter elements:
4
5
3
1
8
5
4
Sorted array:
1 3 4 4 5 5 8
-----
Process exited after 6.883 seconds with return value 0
Press any key to continue . . . |
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: $O(1)$

Variables in main() :

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

Array used:

- int arr[size] \rightarrow size \times 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

```
1 // CH.SC.UACSE24103
2 #include <stdio.h>
3
4 #define MAX 10
5
6 int queue[MAX], front = -1, rear = -1;
7 int visited[MAX] = {0};
8 int graph[MAX][MAX];
9 int n;
10
11 // Enqueue function
12 void enqueue(int v) {
13     if (rear == MAX - 1)
14         return;
15     if (front == -1)
16         front = 0;
17     queue[++rear] = v;
18 }
19
20 // Dequeue function
21 int dequeue() {
22     return queue[front++];
23 }
24
25 // BFS function
26 void BFS(int start) {
27     int i;
28     printf("BFS Traversal: ");
29
30     visited[start] = 1;
31     enqueue(start);
32
33     while (front <= rear) {
34         int v = dequeue();
35         printf("%d ", v);
36
37         for (i = 0; i < n; i++) {
38             if (graph[v][i] == 1 && visited[i] == 0) {
39                 visited[i] = 1;
40                 enqueue(i);
41             }
42         }
43     }
44 }
45
46 int main() {
47     int i, j, start;
48
49     printf("Enter number of vertices: ");
50     scanf("%d", &n);
51
52     printf("Enter adjacency matrix:\n");
53     for (i = 0; i < n; i++) {
54         for (j = 0; j < n; j++) {
55             scanf("%d", &graph[i][j]);
56         }
57     }
58
59     printf("Enter starting vertex: ");
60     scanf("%d", &start);
61
62     BFS(start);
63
64     return 0;
65 }
66
```

OUTPUT:

```
C:\Users\chent\OneDrive - An  ×  +  v
Enter number of vertices: 3
Enter adjacency matrix:
4 5 2
23
34
23
12
112
4
Enter starting vertex: 4
BFS Traversal: 4
-----
Process exited after 10.35 seconds with return value 0
Press any key to continue . . . |
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: $O(n)$

Variables in main() :

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

Arrays used:

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

Auxiliary space:

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

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:

```
double.cpp ^
1 // CH.SC.U4CSE24103
2 #include <stdio.h>
3
4 #define MAX 10
5
6 int visited[MAX] = {0};
7 int graph[MAX][MAX];
8 int n;
9
10 // DFS function
11 void DFS(int v) {
12     int i;
13     visited[v] = 1;
14     printf("%d ", v);
15
16     for (i = 0; i < n; i++) {
17         if (graph[v][i] == 1 && visited[i] == 0) {
18             DFS(i);
19         }
20     }
21 }
22
23 int main() {
24     int i, j, start;
25
26     printf("Enter number of vertices: ");
27     scanf("%d", &n);
28
29     printf("Enter adjacency matrix:\n");
30     for (i = 0; i < n; i++) {
31         for (j = 0; j < n; j++) {
32             scanf("%d", &graph[i][j]);
33         }
34     }
35
36     printf("Enter starting vertex: ");
37     scanf("%d", &start);
38
39     printf("DFS Traversal: ");
40     DFS(start);
41
42     return 0;
43 }
44
```

OUTPUT:

```
Enter number of vertices: 3
Enter adjacency matrix:
4 5 6 5
3 5 3 2
2 3 5 4
Enter starting vertex: DFS Traversal: 3
-----
Process exited after 11.49 seconds with return value 0
Press any key to continue . . .
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: $O(n^2)$

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