AMRITA VISHWA VIDYAPEETHAM

DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK

WEEK - 2

NAME : ROHITH S

ROLL NUMBER : CH.SC.U4CSE24140

CLASS : CSE-B

Question 1: BUBBLE SORT

CODE:

bubblesort.c > ...

```c
//CH.SC.U4CSE24140 - Rohith S
#include<stdio.h>
int main(){
    int i,j,size,temp;
    printf("Enter the size of your array: ");
    scanf("%d",&size);
    printf("Enter the elements of your array: ");
    int arr[size];
    printf("\n");
    for(i=0;i<size;i++){
        printf("Enter element %d: ",i+1);
        scanf("%d",&arr[i]);
    }
    for(i=0;i<size-1;i++){
        for(j=0;j<size-1-i;j++){
            if(arr[j]>arr[j+1]){
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
    printf("Sorted array\n");
    for(i=0;i<size;i++){
        printf("%d ",arr[i]);
    }
    return 0;
}
```

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 JUSTFICATION:

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) + ... + 1
  = n(n − 1) / 2

Overall time complexity:

- **$O(n^2)$**

Question 2: INSERTION SORT

CODE:

insertionsort.c > main()

```c
#include<stdio.h>
int main(){
    int i,j,size,key;
     printf("Enter the size of your array: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements of your array:\n");
    for (i = 0; i < size; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }
    for(i=1;i<size;i++){
        key=arr[i];
        j=i-1;

        while(j>=0&&arr[j]>key){
            arr[j+1]=arr[j];
            j--;
        }
        arr[j+1]=key;
    }
    printf("Sorted array:\n");
    for (i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

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 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 + … + (n − 1)
- = n(n − 1) / 2

- **Overall time complexity: $O(n^2)$**

Question 3:

CODE:

selectionsort.c > main()

```c
//CH.SC.U4CSE24140 - Rohith S
#include <stdio.h>
int main() {
    int i, j, size, min, temp;
    printf("Enter the size of your array: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements of your array:\n");
    for (i = 0; i < size; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }
    for (i = 0; i < size - 1; i++) {
        min = i;
        for (j = i + 1; j < size; j++) {
            if (arr[j] < arr[min]) {
                min = j;
            }
        }
        temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
    printf("Sorted array:\n");
    for (i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

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):

- 4 + 4 + 4 + 4 + 4 = 20 bytes

Auxiliary space:

- Constant → 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) + ... + 1
- = n(n − 1) / 2

$O(n^2)$

Question 4: BUCKET SORT

CODE:

bucketsort.c > main()

```c
//CH.SC.U4CSE24140 - Rohith S
#include <stdio.h>
int main() {
    int i, j, size;
    printf("Enter the size of your array: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements of your array:\n");
    for (i = 0; i < size; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }
    int bucket[10][size];
    int count[10];
    for (i = 0; i < 10; i++) {
        count[i] = 0;
    }
    for (i = 0; i < size; i++) {
        int index = arr[i] / 10;
        bucket[index][count[index]++] = arr[i];
    }
    for (i = 0; i < 10; i++) {
        for (j = 1; j < count[i]; j++) {
            int key = bucket[i][j];
            int k = j - 1;
            while (k >= 0 && bucket[i][k] > key) {
                bucket[i][k + 1] = bucket[i][k];
                k--;
            }
            bucket[i][k + 1] = key;
        }
    }
    int index = 0;
    for (i = 0; i < 10; i++){
        for (j=0;j<count[i];j++) {
            arr[index++]=bucket[i][j];
        }
    }
    printf("Sorted array:\n");
    for (i=0;i<size;i++) {
        printf("%d ",arr[i]);
    }
    return 0;
}
```

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 → $O(n^2)$

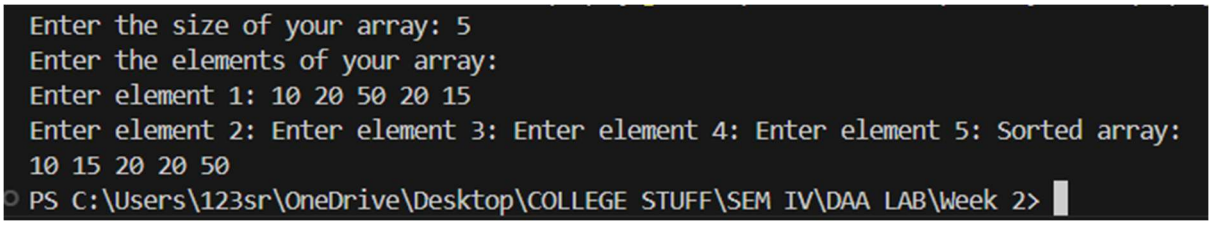
Question 5: HEAP SORT

CODE:

heapsort.c > main()

```c
//CH.SC.U4CSE24140 - Rohith S
#include <stdio.h>
int main() {
    int i, j, size, temp;
    printf("Enter the size of your array: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements of your array:\n");
    for (i = 0; i < size; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }
    for (i = size / 2 - 1; i >= 0; i--) {
        int parent = i;
        while (1) {
            int left = 2 * parent + 1;
            int right = 2 * parent + 2;
            int largest = parent;
            if (left < size && arr[left] > arr[largest])
                largest = left;
            if (right < size && arr[right] > arr[largest])
                largest = right;
            if (largest != parent) {
                temp = arr[parent];
                arr[parent] = arr[largest];
                arr[largest] = temp;
                parent = largest;
            } else {
                break;
            }
        }
    }
    for (i = size - 1; i > 0; i--) {
        temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        int parent = 0;
        while (1) {
            int left = 2 * parent + 1;
            int right = 2 * parent + 2;
            int largest = parent;
            if (left < i && arr[left] > arr[largest])
                largest = left;
            if (right < i && arr[right] > arr[largest])
                largest = right;
            if (largest != parent) {
                temp = arr[parent];
                arr[parent] = arr[largest];
                arr[largest] = temp;
                parent = largest;
            } else {
                break;
            }
        }
    }
    printf("Sorted array:\n");
    for (i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

OUTPUT:



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

bfs.c > ...

```c
//CH.SC.U4CSE24140 - Rohith S
#include <stdio.h>
int main() {
    int n, i, j, start;
    int queue[50], front = 0, rear = 0;
    int visited[50] = {0};
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    int graph[n][n];
    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    queue[rear++] = start;
    visited[start] = 1;
    printf("BFS traversal: ");
    while (front < rear) {
        int v = queue[front++];
        printf("%d ", v);
        for (i = 0; i < n; i++) {
            if (graph[v][i] == 1 && visited[i] == 0) {
                queue[rear++] = i;
                visited[i] = 1;
            }
        }
    }
    return 0;
}
```

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 × 4 bytes
• int queue[n] → n × 4 bytes
• int graph[n][n] → n^2 × 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 × n comparisons

Question 7: DFS

CODE:

dfs.c > main()

```c
//CH.SC.U4CSE24140 - Rohith S
#include <stdio.h>
int n;
int graph[50][50];
int visited[50];
void dfs(int v) {
    int i;
    printf("%d ", v);
    visited[v] = 1;

    for (i = 0; i < n; i++) {
        if (graph[v][i] == 1 && visited[i] == 0) {
            dfs(i);
        }
    }
}
int main() {
    int i, j, start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    for (i = 0; i < n; i++) {
        visited[i] = 0;
    }
    printf("DFS traversal: ");
    dfs(start);
    return 0;
}
```

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: $O(n^2)$

Variables used:

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

Arrays used:

- int visited[n] → n × 4 bytes
- int graph[n][n] → n^2 × 4 bytes

Recursion stack:

- Maximum depth = n
- Stack space → 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 × n comparisons