Prarthana Thyagi -CH.SC.U4CSE24137

Design Analysis and Algorithm – Lab Work

WEEK-2

1.Bubble Sort

Code:

```c
#include <stdio.h>
int main() {
    int n, i, j, temp;
    int arr[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; 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 < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

```
Enter number of elements: 5
Enter 5 elements:
27 56 10 38 22
Sorted array:
10 22 27 38 56
```

Space Complexity: The space complexity of Bubble Sort is O(1) because it sorts the array in place using only a constant amount of extra memory**.**

2.Insertion Sort

Code:

```c
#include <stdio.h>
int main() {
    int n, i, j, key;
    int arr[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    for (i = 1; i < n; 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 < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

```
Enter number of elements: 5
Enter 5 elements:
12 67 9 56 20
Sorted array:
9 12 20 56 67
```

Space Complexity: Insertion Sort has a space complexity of O(1) since it sorts the array in place using only a constant amount of extra memory.

3.Selction Sort

Code:

```c
#include <stdio.h>
int main() {
    int n;
    scanf("%d", &n);
    int a[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++)
            if (a[j] < a[min])
                min = j;
        int t = a[i];
        a[i] = a[min];
        a[min] = t;
    }
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Output:

```
5
12 78 1 39 220
1 12 39 78 220
```

Space Complexity: Selection Sort is an in-place algorithm, using only a few variables, so its auxiliary space is O(1).

4.Bucket Sort

Code:

```c
#include <stdio.h>
#include <stdlib.h>
void insertionSort(int arr[], int n) {
    for(int i=1;i<n;i++){
        int key=arr[i],j=i-1;
        while(j>=0 && arr[j]>key){
            arr[j+1]=arr[j]; j--;
        }
        arr[j+1]=key;
    }
}
void bucketSort(int arr[], int n){
    int i,j,k,max=arr[0];
    for(i=1;i<n;i++) if(arr[i]>max) max=arr[i];
    int bucketCount=n;
    int **b=(int**)malloc(bucketCount*sizeof(int*));
    int *bSize=(int*)malloc(bucketCount*sizeof(int));
    for(i=0;i<bucketCount;i++){ b[i]=(int*)malloc(sizeof(int)*n); bSize[i]=0; }
    for(i=0;i<n;i++){
        int index=(arr[i]*bucketCount)/(max+1);
        b[index][bSize[index]++]=arr[i];
    }
    k=0;
    for(i=0;i<bucketCount;i++){
        insertionSort(b[i],bSize[i]);
        for(j=0;j<bSize[i];j++) arr[k++]=b[i][j];
        free(b[i]);
    }
    free(b); free(bSize);
}
int main(){
    int n; scanf("%d",&n);
    int arr[n]; for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    bucketSort(arr,n);
    for(int i=0;i<n;i++) printf("%d ",arr[i]);
    return 0;
}
```

Output:

```
5
20 58 10 28 44
10 20 28 44 58
```

Space complexity: Bucket Sort uses extra space for buckets, so its space complexity is O(n + k), where n is the number of elements and k is the number of buckets.

5.Heap Sort

Code:

```c
#include <stdio.h>
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
void heapify(int a[], int n, int i) {
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    if (l < n && a[l] > a[largest])
        largest = l;
    if (r < n && a[r] > a[largest])
        largest = r;
    if (largest != i) {
        swap(&a[i], &a[largest]);
        heapify(a, n, largest);
    }
}
void heapSort(int a[], int n) {
    for (int i = n/2 - 1; i >= 0; i--)
        heapify(a, n, i);
    for (int i = n - 1; i > 0; i--) {
        swap(&a[0], &a[i]);
        heapify(a, i, 0);
    }
}
int main() {
    int n;
    scanf("%d", &n);
    int a[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    heapSort(a, n);
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
5
28 10 38 42 18
10 18 28 38 42
```

Space complexity: Heap Sort is an in-place sorting algorithm that requires only constant extra space.
Its auxiliary space complexity is O(1), excluding the O(log n) recursive call stack used during heapify.

6.BFS

Code:

```c
#include <stdio.h>
#define MAX 10
int n, adj[MAX][MAX], visited[MAX];
void bfs(int start) {
    int queue[MAX], front = 0, rear = 0;
    queue[rear++] = start;
    visited[start] = 1;
    printf("BFS Traversal: ");
    while (front < rear) {
        int v = queue[front++];
        printf("%d ", v);

        for (int i = 0; i < n; i++) {
            if (adj[v][i] == 1 && !visited[i]) {
                queue[rear++] = i;
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}
int main() {
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix (%d x %d):\n", n, n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);
    int start;
    printf("Enter starting vertex (0 to %d): ", n-1);
    scanf("%d", &start);
    bfs(start);
    return 0;
}
```

Output:

```
Enter number of vertices: 3
Enter adjacency matrix (3 x 3):
2 8 5
2 7 1
4 9 5
Enter starting vertex (0 to 2): 0
BFS Traversal: 0
```

Space complexity: BFS requires a queue to store vertices, so its space complexity is O(V) in the worst case.

7.DFS

Code:

```c
#include <stdio.h>
#define MAX 1
int n, adj[MAX][MAX], visited[MAX];
void dfs(int v) {
    visited[v] = 1;
    printf("%d ", v);
    for (int i = 0; i < n; i++) {
        if (adj[v][i] == 1 && !visited[i])
            dfs(i);
    }
}
int main() {
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix (%d x %d):\n", n, n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);
    int start;
    printf("Enter starting vertex (0 to %d): ", n-1);
    scanf("%d", &start);
    printf("DFS Traversal: ");
    dfs(start);
    printf("\n");
    return 0;
}
```

Output:

```
Enter number of vertices: 3
Enter adjacency matrix (3 x 3):
2 9 1
7 2 6
9 8 4
Enter starting vertex (0 to 2): 2
DFS Traversal: 2
```

Space complexity: DFS uses a stack (recursive or explicit) that can grow up to the number of vertices, so its space complexity is O(V) in the worst case.