AMRITA VISHWA VIDYAPEETHAM
CHENNAI

Computer Science and Engineering
(CSE)

DESIGN AND ANALYSIS OF ALGORITHMS

Roll No: CH.SC.U4CSE24112

Name: Deepak SN

Class: CSE-B

Year: 2024-2028

Design and Analysis of Algorithms

SORTING:

1. Bubble Sort:

Code:

```c
//bubble sort
#include <stdio.h>
void bubbleSort(int arr[],int n){
    for(int i=0;i<n-1;i++){
        for(int j=0;j<n-i-1;j++){
            if(arr[j]>arr[j+1]){
                int temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
}
int main(){
    int arr[]={64,34,25,12,22,11,90};
    int n=sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr,n);
    printf("Sorted array: ");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    return 0;
}
```

Output:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> gcc bubblesort.c -o bubblesort
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> ./bubblesort
Sorted array: 11 12 22 25 34 64 90
```

2. Insertion Sort:

Code:

```c
//insertion sort
#include <stdio.h>
void insertionSort(int arr[],int n){
    for(int i=1;i<n;i++){
        int key=arr[i];
        int j=i-1;
        while(j>=0&&arr[j]>key){
            arr[j+1]=arr[j];
```

```
            j--;
        }
        arr[j + 1] = key;
    }
}
int main(){
    int arr[]={12,11,13,5,6};
    int n=sizeof(arr)/sizeof(arr[0]);
    insertionSort(arr, n);
    printf("Sorted array: ");
    for(int i=0;i<n;i++)
        printf("%d ",arr[i]);
    printf("\n");
    return 0;
}
```

Output:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> gcc insertionsort.c -o insertionsort
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> ./insertionsort
Sorted array: 5 6 11 12 13
```

3. Selection Sort:

Code:

```
#include <stdio.h>
void selectionSort(int arr[],int n){
    for (int i=0;i<n-1;i++){
        int minIndex=i;
        for(int j=i+1;j<n;j++){
            if(arr[j]<arr[minIndex])
                minIndex=j;
        }
        if(minIndex != i){
            int temp=arr[i];
            arr[i]=arr[minIndex];
            arr[minIndex]=temp;
        }
    }
}
int main(){
    int arr[]={64,25,12,22,11};
    int n=sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr,n);
    printf("Sorted array: ");
    for(int i=0;i<n;i++)
        printf("%d ",arr[i]);
    printf("\n");
    return 0;
}
```

Output:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> gcc selectionsort.c -o selectionsort
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> ./selectionsort
Sorted array: 11 12 22 25 64
```

4. Bucket Sort:

Code:

```c
#include <stdio.h>
#include <stdlib.h>
void bucket_sort(int a[],int n){
    int i,j,k,max=a[0];
    for(i=1;i<n;i++)
        if(a[i]>max)
            max=a[i];
    int bcount=max+1;
    int *b=calloc(bcount,sizeof(int));
    for(i=0;i<n;i++)
        b[a[i]]++;
        k=0;
    for(i=0;i<bcount;i++){
        for(j=0;j<b[i];j++){
            a[k++]=i;
        }
    }
    free(b);
}
int main(){
    int a[]={4,1,3,4,2,0,1};
    int n=sizeof(a)/sizeof(a[0]);
    bucket_sort(a,n);
    for(int i=0;i<n;i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

Output:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> gcc bucketsort.c -o bucketsort
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> ./bucketsort
0 1 1 2 3 4 4
```

5. Heap Sort:

a. Min-heap Sort:

Code:

```c
#include <stdio.h>
void heapify_min(int a[],int n,int i){
    int s=i,l=2*i+1,r=2*i+2,t;
```

```
    if(l<n&&a[l]<a[s])
        s=l;
    if(r<n&&a[r]<a[s])
        s=r;
    if(s!=i){
        t=a[i];
        a[i]=a[s];
        a[s]=t;
        heapify_min(a,n,s);
    }
}
void heapsort_min(int a[],int n){
    for(int i=n/2-1;i>=0;i--){
        heapify_min(a,n,i);
    }
    for(int i=n-1;i>=0;i--){
        int t=a[0];
        a[0]=a[i];
        a[i]=t;
        heapify_min(a,i,0);
    }
}
int main(){
    int a[]={4,1,3,2,0};
    int n=5;
    heapsort_min(a,n);
    for(int i=0;i<n;i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

Output:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> gcc minheapsort.c -o minheapsort
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> ./minheapsort
4 3 2 1 0
```

b. Max-heap Sort:

Code:

```
#include <stdio.h>
void heapify_max(int a[],int n,int i){
    int l=2*i+1,r=2*i+2,m=i,t;
    if(l<n&&a[l]>a[m])
        m=l;
    if(r<n&&a[r]>a[m])
        m=r;
    if(m!=i){
        t=a[i];
        a[i]=a[m];
```

```
        a[m]=t;
        heapify_max(a,n,m);
    }
}
void heapsort_max(int a[],int n){
    for(int i=n/2-1;i>=0;i--){
        heapify_max(a,n,i);
    }
    for(int i=n-1;i>=0;i--){
        int t=a[0];
        a[0]=a[i];
        a[i]=t;
        heapify_max(a,i,0);
    }
}
int main(){
    int a[]={4,1,3,2,0};
    int n=5;
    heapsort_max(a,n);
    for(int i=0;i<n;i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

Output:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> gcc maxheapsort.c -o maxheapsort
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> ./maxheapsort
0 1 2 3 4
```

SEARCHING:

1. Breadth-First Search:

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct Node{
    int vertex;
    struct Node* next;
}Node;
typedef struct Graph{
    int numVertices;
    Node* adjList[MAX];
    int visited[MAX];
}Graph;
```

```c
typedef struct Queue{
    int items[MAX];
    int front;
    int rear;
}Queue;
Node* createNode(int v){
    Node* newNode=(Node*)malloc(sizeof(Node));
    newNode->vertex=v;
    newNode->next=NULL;
    return newNode;
}
Graph* createGraph(int vertices){
    Graph* graph=(Graph*)malloc(sizeof(Graph));
    graph->numVertices=vertices;

    for (int i=0;i<vertices;i++){
        graph->adjList[i]=NULL;
        graph->visited[i]=0;
    }
    return graph;
}
void addEdge(Graph* graph, int src, int dest) {
    Node* newNode=createNode(dest);
    newNode->next=graph->adjList[src];
    graph->adjList[src]=newNode;
    newNode=createNode(src);
    newNode->next=graph->adjList[dest];
    graph->adjList[dest]=newNode;
}
Queue* createQueue(){
    Queue*q=(Queue*)malloc(sizeof(Queue));
    q->front=-1;
    q->rear=-1;
    return q;
}
int isEmpty(Queue* q){
    return (q->rear==-1);
}
void enqueue(Queue* q, int value){
    if (q->rear==MAX-1)
        return;
    if (q->front==-1)
        q->front=0;
    q->rear++;
    q->items[q->rear]=value;
}
int dequeue(Queue* q){
    if (isEmpty(q))
```

```
        return -1;
    int item=q->items[q->front];
    q->front++;
    if (q->front>q->rear){
        q->front=q->rear=-1;
    }
    return item;
}
void bfs(Graph* graph, int startVertex) {
    Queue* q=createQueue();
    graph->visited[startVertex]=1;
    enqueue(q, startVertex);
    printf("BFS Traversal: ");
    while (!isEmpty(q)){
        int currentVertex=dequeue(q);
        printf("%d ", currentVertex);
        Node* temp=graph->adjList[currentVertex];
        while (temp){
            int adjVertex=temp->vertex;
            if (!graph->visited[adjVertex]){
                graph->visited[adjVertex]=1;
                enqueue(q, adjVertex);
            }
            temp=temp->next;
        }
    }
    printf("\n");
}
int main() {
    int vertices=6;
    Graph* graph=createGraph(vertices);
    addEdge(graph,0,1);
    addEdge(graph,0,2);
    addEdge(graph,1,3);
    addEdge(graph,1,4);
    addEdge(graph,2,4);
    addEdge(graph,3,5);
    addEdge(graph,4,5);
    bfs(graph,0);
    return 0;
}
```

Output:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> gcc bfs.c -o bfs
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> ./bfs
BFS Traversal: 0 2 1 4 3 5
```

2. Depth-First Search:

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct Node{
    int vertex;
    struct Node* next;
}Node;
typedef struct Graph{
    int numVertices;
    Node* adjList[MAX];
    int visited[MAX];
}Graph;
Node* createNode(int v){
    Node* newNode=(Node*)malloc(sizeof(Node));
    newNode->vertex=v;
    newNode->next=NULL;
    return newNode;
}
Graph* createGraph(int vertices){
    Graph* graph=(Graph*)malloc(sizeof(Graph));
    graph->numVertices=vertices;
    for (int i=0;i<vertices;i++) {
        graph->adjList[i]=NULL;
        graph->visited[i]=0;
    }
    return graph;
}
void addEdge(Graph* graph, int src, int dest){
    Node* newNode=createNode(dest);
    newNode->next=graph->adjList[src];
    graph->adjList[src]=newNode;
    newNode=createNode(src);
    newNode->next=graph->adjList[dest];
    graph->adjList[dest]=newNode;
}
void dfsUtil(Graph* graph,int vertex){
    graph->visited[vertex]=1;
    printf("%d ", vertex);
    Node* temp=graph->adjList[vertex];
    while (temp){
        int adjVertex=temp->vertex;
        if (!graph->visited[adjVertex]){
            dfsUtil(graph, adjVertex);
        }
        temp=temp->next;
```

```
    }
}
void dfs(Graph* graph, int startVertex){
    printf("DFS Traversal: ");
    dfsUtil(graph, startVertex);
    printf("\n");
}
int main(){
    int vertices=6;
    Graph* graph=createGraph(vertices);
    addEdge(graph,0,1);
    addEdge(graph,0,2);
    addEdge(graph,1,3);
    addEdge(graph,1,4);
    addEdge(graph,2,4);
    addEdge(graph,3,5);
    addEdge(graph,4,5);
    dfs(graph,0);
    return 0;
}
```

Output:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> gcc dfs.c -o dfs
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-4\Design and Analysis of Algorithms\Practice> ./dfs
DFS Traversal: 0 2 4 5 3 1
```