AMRITA VISHWA VIDYAPEETHAM

CHENNAI CAMPUS

Second Year. B.TECH

(Computer Science and Engineering)

(DAA Lab Work)

Name: M CYNTHIA SHREE

RollNo.: CH.SC.U4CSE24110

Department: CSE-B

Academic Year: 2025-26

1. Bubble Sort

CODE:

```
//ch.sc.u4cse24110
//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-1-i;j++) {
if (arr[j]>arr[j+1]) {
int temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;;
}
    }
        }
            }
int main() {
int n;
printf("enter size of array:");
scanf("%d",&n);
int arr[n];
for (int i=0;i<n;i++) {
printf("enter element at position %d:",i);
scanf("%d",&arr[i]);
}
bubblesort(arr,n);
printf("sorted array: ");
for (int i=0;i<n;i++) {
printf("%d ",arr[i]);
}
printf("\n");
}
```

OUTPUT:

```
C:\Users\mrgns\Documents>gcc bubblesort.c -o bubblesort

C:\Users\mrgns\Documents>.\bubblesort
enter size of array:6
enter element at position 0:2
enter element at position 1:8
enter element at position 2:1
enter element at position 3:9
enter element at position 4:3
enter element at position 5:4
sorted array: 1 2 3 4 8 9
```

TIME COMPLEXITY: O(n^2) because 2 nested loop.

SPACE COMPLEXITY: O(1) because only variable temp is used and no extra memory.

2. Insertion Sort

CODE:

```c
//ch.sc.u4cse24110
//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=j-1;
}
arr[j+1]= key;
}
}
int main() {
int n;
printf("enter size of array:");
scanf("%d",&n);
int arr[n];
for (int i=0;i<n;i++) {
printf("enter element at position %d:",i);
scanf("%d",&arr[i]);
}
insertionsort(arr,n);
printf("sorted array:");
for (int i=0;i<n;i++) {
printf("%d ",arr[i]);
}
printf("\n");
}
```

OUTPUT:

```
amma@amma08:~/Desktop$ gcc insertionsort.c -o insertionsort
amma@amma08:~/Desktop$ ./insertionsort

enter size of array:6
enter element at position 0:2
enter element at position 1:8
enter element at position 2:1
enter element at position 3:6
enter element at position 4:7
enter element at position 5:4
sorted array:1 2 4 6 7 8
```

TIME COMPLEXITY: Worst case is a reversed sorted array which will take n times into the for loop which runs n-1 times so its O(n^2).

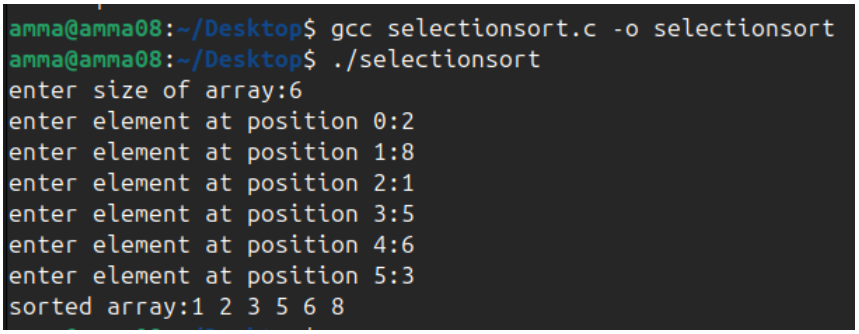
SPACE COMPLEXITY: O(1) because only key is used, no additional memory.

3. Selection Sort

CODE:

```c
//ch.sc.u4cse24110
//selection sort
#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 n;
printf("enter size of array:");
scanf("%d",&n);
int arr[n];
for (int i=0;i<n;i++) {
printf("enter element at position %d:",i);
scanf("%d",&arr[i]);
}
selectionsort(arr,n);
printf("sorted array:");
for (int i=0;i<n;i++) {
printf("%d ",arr[i]);  }
printf("\n");   }
```

OUTPUT:



TIME COMPLEXITY: 2 for loops so O(n^2).

SPACE COMPLEXITY: O(1) because just temp and no additionary memory used.

4. Bucket Sort

CODE:

```c
//ch.sc.u4cse24110
//Bucket sort
#include <stdio.h>
#define MAX 100
void bucketsort(int arr[], int n) {
int bucket[MAX]={0};
for (int i=0;i<n;i++) {
bucket[arr[i]]++;
}
int k=0;
for (int i=0;i<MAX;i++) {
while(bucket[i]>0) {
arr[k++]=i;
bucket[i]--;
}
}
}

int main() {
int n;
printf("enter size of array:");
scanf("%d",&n);
int arr[n];
for (int i=0;i<n;i++) {
printf("enter element at position %d:",i);
scanf("%d",&arr[i]);
}
bucketsort(arr,n);
printf("sorted array: ");
for (int i=0;i<n;i++) {
printf("%d ",arr[i]);
}
printf("\n");
}
```

OUTPUT:

```
C:\Users\mrgns\Documents>gcc bucketsort.c -o bucketsort

C:\Users\mrgns\Documents>.\bucketsort
enter size of array:5
enter element at position 0:2
enter element at position 1:8
enter element at position 2:1
enter element at position 3:9
enter element at position 4:4
sorted array: 1 2 4 8 9
```

TIME COMPLEXITY: Since MAX is defined only while loop and for loop is considered which each runs at n times so O(n).

SPACE COMPLEXITY: O(1) because bucket[MAX] uses MAX spaces but it is defines so it becomes constant.

5. Min Heap Sort

CODE:

```
//ch.sc.u4cse24110
//minheap sort
#include <stdio.h>
void heapify(int arr[], int n, int i) {
int smallest=i;
int left=2*i+1;
int right=2*i+2;

if (left<n&&arr[left]<arr[smallest])
smallest=left;
if (right<n&&arr[right]<arr[smallest])
smallest=right;

if (smallest!=i) {
int temp=arr[i];
arr[i]=arr[smallest];
arr[smallest]=temp;

heapify(arr,n,smallest);
    }
}

void minheapsort(int arr[], int n) {
for (int i=n/2-1;i>=0;i--) {
heapify(arr,n,i);
    }
for (int i=n-1;i>=0;i--) {
int temp=arr[0];
arr[0]=arr[i];
arr[i]=temp;

heapify(arr,i,0);
    }
}
```

```
int main() {
int n;
printf("enter size of array:");
scanf("%d",&n);
int arr[n];

for (int i=0;i<n;i++) {
printf("enter element at position %d:",i);
scanf("%d",&arr[i]);
}

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

OUTPUT:

```
C:\Users\mrgns\Documents>gcc minheapsort.c -o minheapsort

C:\Users\mrgns\Documents>.\minheapsort
enter size of array:5
enter element at position 0:2
enter element at position 1:8
enter element at position 2:1
enter element at position 3:9
enter element at position 4:4
sorted array: 9 8 4 2 1
```

TIME COMPLEXITY: O(n log n) because heapify is called n times and each heapify takes O(log n).

SPACE COMPLEXITY: O(1) because only variable temp is used and no extra memory.

6. Max Heap Sort

CODE:

```
//ch.sc.u4cse24110
//maxheap sort
#include <stdio.h>
void heapify(int arr[], int n, int i) {
int largest=i;
int left=2*i+1;
int right=2*i+2;

if (left<n&&arr[left]>arr[largest])
largest=left;
if (right<n&&arr[right]>arr[largest])
largest=right;

if (largest!=i) {
int temp=arr[i];
arr[i]=arr[largest];
arr[largest]=temp;

heapify(arr,n,largest);
    }
}

void maxheapsort(int arr[], int n) {
for (int i=n/2-1;i>=0;i--) {
heapify(arr,n,i);
    }
for (int i=n-1;i>=0;i--) {
int temp=arr[0];
arr[0]=arr[i];
arr[i]=temp;

heapify(arr,i,0);
    }
}
```

```
int main() {
int n;
printf("enter size of array:");
scanf("%d",&n);
int arr[n];

for (int i=0;i<n;i++) {
printf("enter element at position %d:",i);
scanf("%d",&arr[i]);
}

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

OUTPUT:

```
C:\Users\mrgns\Documents>gcc maxheapsort.c -o maxheapsort

C:\Users\mrgns\Documents>.\maxheapsort
enter size of array:5
enter element at position 0:2
enter element at position 1:9
enter element at position 2:1
enter element at position 3:7
enter element at position 4:5
sorted array: 1 2 5 7 9
```

TIME COMPLEXITY: O(n log n) because heapify is called n times and each heapify takes O(log n).

SPACE COMPLEXITY: O(1) because only variable temp is used and no extra memory.

7. BFS

CODE:

```
//ch.sc.u4cse24110
//bfs
#include <stdio.h>
#define MAX 100

int visited[MAX];
int adj[MAX][MAX];
int queue[MAX];
int front=0,rear=0;

void enqueue(int n) {
queue[rear++]=n;
}

int dequeue() {
return queue[front++];
}

int isEmpty() {
return front==rear;
}

void bfs(int start,int n) {
for(int i=0;i<n;i++) {
visited[i]=0;
}
visited[start]=1;
enqueue(start);

while (!isEmpty()) {
int node=dequeue();
printf("%d",node);
for (int i=0;i<n;i++) {
if (adj[node][i]==1&&!visited[i]) {
visited[i]=1;
enqueue(i);
            }
        }
    }
}
```

```
int main() {
int n=5;
adj[0][1] = adj[1][0] = 1;
adj[0][2] = adj[2][0] = 1;
adj[1][3] = adj[3][1] = 1;
adj[2][4] = adj[4][2] = 1;
bfs(0, n);
}
```

OUTPUT:

```
C:\Users\mrgns\Documents>gcc bfs.c -o bfs

C:\Users\mrgns\Documents>.\bfs
01234
```

TIME COMPLEXITY: O(n^2) because while loop runs n times in worst case and inner for loop runs n times hence nxn.

SPACE COMPLEXITY: O(n^2) because of adj matrix.

8. DFS

CODE:

```c
//ch.sc.u4cse24110
//dfs
#include <stdio.h>
#define MAX 100

int visited[MAX];
int adj[MAX][MAX];

void dfs(int node,int n) {
visited[node]=1;
printf("%d",node);

for (int i=0;i<n;i++) {
if (adj[node][i]==1&&!visited[i]) {
dfs(i,n);
            }
        }
    }

int main() {
int n=5;
adj[0][1] = adj[1][0] = 1;
adj[0][2] = adj[2][0] = 1;
adj[1][3] = adj[3][1] = 1;
adj[2][4] = adj[4][2] = 1;

for(int i=0;i<n;i++) {
visited[i]=0;
}

dfs(0, n);
}
```

OUTPUT:

```
C:\Users\mrgns\Documents>gcc dfs.c -o dfs

C:\Users\mrgns\Documents>.\dfs
01324
```

TIME COMPLEXITY: O(n^2) because dfs is called n times and the inner for loop is n times hence nxn.

SPACE COMPLEXITY: O(n^2) because adj matrix.