
Name: Ganath Avinash G.R

CSE – B

CH.SC.U4CSE24118

Week –2 (4/12/2025)

## 1.Bubble Sort

```c
#include <stdio.h>

void bubble_sort(int* arr,int l){
  for(int i=0;i<l;i++){
    for(int j=0;j<l;j++){
      if(arr[j]>arr[j+1]){
        int t=arr[j];
        arr[j]=arr[i];
        arr[i]=t;
      }
    }
  }
}

int main(){
  int arr[]={11,2,12,3,245,3,4,2,4,5,7,4,567,5,4,44};
  bubble_sort(arr,15);
  printf("Bubble Sorted O(n^2): \n");
  for(int i=0;i<15;i++){
    printf(" %d ",arr[i]);
  }
  printf("\n");
}
```

**OUTPUT:**

```
amma@amma15:~/Documents/CH.SC.U4CSE24217$ gcc -o bb bubblesort.c
amma@amma15:~/Documents/CH.SC.U4CSE24217$ ./bb
Bubble Sorted O(n^2):
 2  2  3  3  4  4  4  4  5  5  7  11  12  245  567
amma@amma15:~/Documents/CH.SC.U4CSE24217$
```

**Space Complexity:O(1)**
**Time Complexity:O(N^2)**

**Justification:**

**For whatever maybe the input no of storage boxes(variable needed) is constant. Hence the Space Complexity is O(1).**
**Uses two nested loops to repeatedly compare adjacent elements. Hence the Time Complexity is O(N^2)**

## 2.Selection Sort

```c
#include <stdio.h>

void selection_sort(int* arr,int l){
  for(int i=0;i<l;i++){
    int midx=i;
    for(int j=i+1;j<l;j++){
      if(arr[midx]>arr[j]){
        midx=j;
      }
    }
    int t=arr[i];
    arr[i]=arr[midx];
    arr[midx]=t;
  }
}

int main(){
  int arr[]={11,2,12,3,245,3,4,2,4,5,7,4,567,5,4,44};
  selection_sort(arr,15);
  printf("Selection Sorted O(n^2): \n");
  for(int i=0;i<15;i++){
    printf(" %d ",arr[i]);
  }
  printf("\n");
}
```

**OUTPUT:**

```
amma@amma15:~/Documents/CH.SC.U4CSE24217$ gcc -o ss selectsort.c
amma@amma15:~/Documents/CH.SC.U4CSE24217$ ./ss
Selection Sorted O(n^2):
 2  2  3  3  4  4  4  4  5  5  7  11  12  245  567
amma@amma15:~/Documents/CH.SC.U4CSE24217$
```

**Space Complexity:O(1)**
**Time Complexity:O(N^2)**

**Justification:**

**For whatever maybe the input no of storage boxes(variable needed) is constant. Hence the Space Complexity is O(1).**
**Uses two nested loops to repeatedly compare adjacent elements. Hence the Time Complexity is O(N^2)**

## 3.Insertion Sort

```c
#include <stdio.h>

void inser_sort(int* arr,int l){
  for(int i=0;i<l;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[]={11,2,12,3,245,3,4,2,4,5,7,4,567,5,4,44};
  inser_sort(arr,15);
  printf("Insertion Sorted O(n^2): \n");
  for(int i=0;i<15;i++){
    printf(" %d ",arr[i]);
  }
  printf("\n");
}
```

**OUTPUT:**

```
amma@amma15:~/Documents/CH.SC.U4CSE24217$ gcc -o is inserts.c
amma@amma15:~/Documents/CH.SC.U4CSE24217$ ./is
Insertion Sorted O(n^2):
 2  2  3  3  4  4  4  4  5  5  7  11  12  245  567
amma@amma15:~/Documents/CH.SC.U4CSE24217$
```

**Space Complexity:O(1)**
**Time Complexity:O(N^2)**

**Justification:**

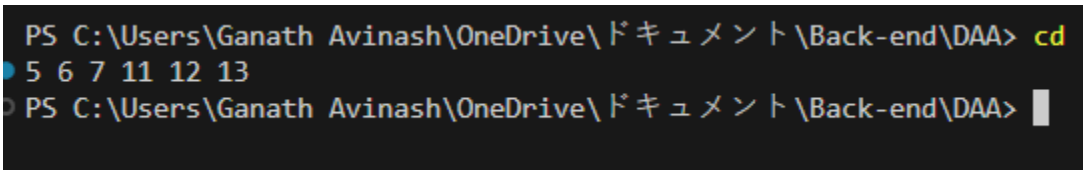
**For whatever maybe the input no of storage boxes(variable needed) is constant. Hence the Space Complexity is O(1).**
**Uses two nested loops to repeatedly compare adjacent elements. Hence the Time Complexity is O(N^2)**

## 4.Maxheap

C maxHeap.c > heapify(int [], int, int)

```c
#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 heapSort(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 arr[] = {12, 11, 13, 5, 6, 7};
    int n = 6;

    heapSort(arr, n);

    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
```

**OUTPUT:**




**Space Complexity:O(1)**
**Time Complexity:O(N log N)**

## Justification:

**For whatever maybe the input no of storage boxes(variable needed) is constant. Hence the Space Complexity is O(1).**
**Uses two nested loops to repeatedly compare adjacent elements. Hence the Time Complexity is O(N^2)**

## 5.Minheap

C minheap.c > heapSort(int [], int)

```c
#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 heapSort(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 arr[] = {12, 11, 13, 5, 6, 7};
    int n = 6;

    heapSort(arr, n);

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

    return 0;
```

**OUTPUT:**

```
PS C:\Users\Ganath Avinash\OneDrive\ドキュメント\Back-end\DAA> cd
Sorted Array:
13 12 11 7 6 5
PS C:\Users\Ganath Avinash\OneDrive\ドキュメント\Back-end\DAA>
```

**Space Complexity:O(1)**
**Time Complexity:O(N log N)**

## Justification:

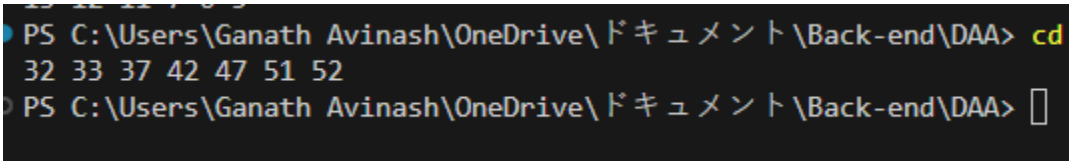
**For whatever maybe the input no of storage boxes(variable needed) is constant. Hence the Space Complexity is O(1).**
**Uses two nested loops to repeatedly compare adjacent elements. Hence the Time Complexity is O(N^2)**

## 6.Bucket Sort

bucket.c > bucketSort(int [], int)

```c
#include <stdio.h>

void bucketSort(int arr[], int n) {
    int bucket[10][10], count[10] = {0};

    for (int i = 0; i < n; i++) {
        int index = arr[i] / 10;
        bucket[index][count[index]++] = arr[i];
    }

    for (int i = 0; i < 10; i++) {
        for (int 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 idx = 0;
    for (int i = 0; i < 10; i++)
        for (int j = 0; j < count[i]; j++)
            arr[idx++] = bucket[i][j];
}

int main() {
    int arr[] = {42, 32, 33, 52, 37, 47, 51};
    int n = 7;

    bucketSort(arr, n);

    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

**OUTPUT:**




**Space Complexity:O(N^2)**
**Time Complexity:O(N^2)**

## Justification:

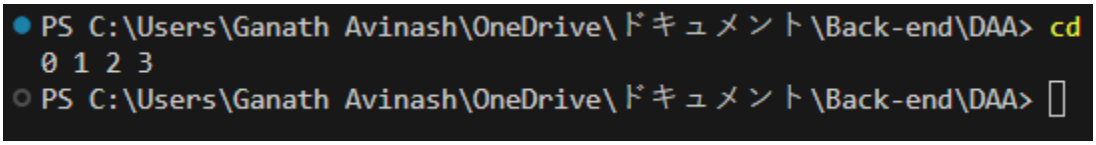
**In the worst case, all elements may fall into a single bucket and are sorted using insertion sort.Hence the Space Complexity is O(N^2).**
**Uses two nested loops to repeatedly compare adjacent elements. Hence the Time Complexity is O(N^2)**

7.BFS

C BFS.c > ...

```c
#include <stdio.h>

int queue[20], front = -1, rear = -1;
int visited[20];

void bfs(int graph[20][20], int n, int start) {
    queue[++rear] = start;
    visited[start] = 1;

    while (front != rear) {
        int v = queue[++front];
        printf("%d ", v);

        for (int i = 0; i < n; i++) {
            if (graph[v][i] == 1 && visited[i] == 0) {
                queue[++rear] = i;
                visited[i] = 1;
            }
        }
    }
}

int main() {
    int n = 4;
    int graph[20][20] = {
        {0,1,1,0},
        {1,0,0,1},
        {1,0,0,1},
        {0,1,1,0}
    };

    bfs(graph, n, 0);
    return 0;
}
```

**OUTPUT:**




**Space Complexity:O(V)**
**Time Complexity:O(V+E)**

## Justification:

**The queue can store upto V vertices in the worst case. Hence the Space Complexity is O(V).**
**BFS visits every vertex once and checks every edge once while exploring adjacency lists .Hence the Time Complexity is O(V+E)**

## 8.DFS

```c
#include <stdio.h>

int visited[20];

void dfs(int graph[20][20], int n, int v) {
    printf("%d ", v);
    visited[v] = 1;

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

int main() {
    int n = 4;
    int graph[20][20] = {
        {0,1,1,0},
        {1,0,0,1},
        {1,0,0,1},
        {0,1,1,0}
    };

    dfs(graph, n, 0);
    return 0;
}
```

**OUTPUT:**

```
amma@amma15:~/Documents/CH.SC.U4CSE24217$ gcc -o is inserts.c
amma@amma15:~/Documents/CH.SC.U4CSE24217$ ./is
Insertion Sorted O(n^2):
 2  2  3  3  4  4  4  4  5  5  7  11  12  245  567
amma@amma15:~/Documents/CH.SC.U4CSE24217$
```

**Space Complexity:O(V)**
**Time Complexity:O(V+E)**

## Justification:

**The recursion stack can grow upto V levels and the visited [ ] array stores V entries. Hence the Space Complexity is O(V).**
**DFS recursively explores every vertex and inspects all edges exactly once through the adjacency list. Hence the Time Complexity is O(V+E).**