



Name: Ganath Avinash G.R

CSE – B

CH.SC.U4CSE24118

Week –2 (4/12/2025)

1.Bubble Sort

```
C bubblesort.c > ...
1  #include <stdio.h>
2
3  void bubble_sort(int* arr,int l){
4      for(int i=0;i<l;i++){
5          for(int j=0;j<l;j++){
6              if(arr[j]>arr[j+1]){
7                  int t=arr[j];
8                  arr[j]=arr[i];
9                  arr[i]=t;
10             }
11         }
12     }
13 }
14
15 int main(){
16     int arr[]={11,2,12,3,245,3,4,2,4,5,7,4,567,5,4,44};
17     bubble_sort(arr,15);
18     printf("Bubble Sorted O(n^2): \n");
19     for(int i=0;i<15;i++){
20         printf(" %d ",arr[i]);
21     }
22     printf("\n");
23 }
```

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(N²)

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(N²)

2.Selection Sort

```
C selectsort.c > ...
1  #include <stdio.h>
2
3  void selection_sort(int* arr,int l){
4      for(int i=0;i<l;i++){
5          int midx=i;
6          for(int j=i+1;j<l;j++){
7              if(arr[midx]>arr[j]){
8                  midx=j;
9              }
10         }
11         int t=arr[i];
12         arr[i]=arr[midx];
13         arr[midx]=t;
14     }
15 }
16
17 int main(){
18     int arr[]={11,2,12,3,245,3,4,2,4,5,7,4,567,5,4,44};
19     selection_sort(arr,15);
20     printf("Selection Sorted O(n^2): \n");
21     for(int i=0;i<15;i++){
22         printf(" %d ",arr[i]);
23     }
24     printf("\n");
25 }
```

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(N²)

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(N²)

3.Insertion Sort

```
C inserts.c > ...
1 #include <stdio.h>
2
3 void inser_sort(int* arr,int l){
4     for(int i=0;i<l;i++){
5         int key=arr[i];
6         int j=i-1;
7         while(j>=0 && arr[j]>key){
8             arr[j+1]=arr[j];
9             j--;
10        }
11        arr[j+1]=key;
12    }
13 }
14
15 int main(){
16     int arr[]={11,2,12,3,245,3,4,2,4,5,7,4,567,5,4,44};
17     inser_sort(arr,15);
18     printf("Insertion Sorted O(n^2): \n");
19     for(int i=0;i<15;i++){
20         printf(" %d ",arr[i]);
21     }
22     printf("\n");
23 }
```

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)
1  #include <stdio.h>
2
3 void heapify(int arr[], int n, int i) {
4     int largest = i;
5     int left = 2*i + 1;
6     int right = 2*i + 2;
7
8     if (left < n && arr[left] > arr[largest])
9         largest = left;
10
11    if (right < n && arr[right] > arr[largest])
12        largest = right;
13
14    if (largest != i) {
15        int temp = arr[i];
16        arr[i] = arr[largest];
17        arr[largest] = temp;
18
19        heapify(arr, n, largest);
20    }
21}
22
23 void heapSort(int arr[], int n) {
24     for (int i = n/2 - 1; i >= 0; i--)
25         heapify(arr, n, i);
26
27     for (int i = n - 1; i > 0; i--) {
28         int temp = arr[0];
29         arr[0] = arr[i];
30         arr[i] = temp;
31
32         heapify(arr, i, 0);
33     }
34 }
35
36 int main() {
37     int arr[] = {12, 11, 13, 5, 6, 7};
38     int n = 6;
39
40     heapSort(arr, n);
41
42     for (int i = 0; i < n; i++)
43         printf("%d ", arr[i]);
44
45     return 0;
}
```

OUTPUT:

```
PS C:\Users\Ganath Avinash\OneDrive\ドキュメント\Back-end\DAA> cd  
● 5 6 7 11 12 13  
○ 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 $O(N^2)$

5.Minheap

```
C minheap.c > heapSort(int [], int)
1  #include <stdio.h>
2  void heapify(int arr[], int n, int i) {
3      int smallest = i;
4      int left = 2*i + 1;
5      int right = 2*i + 2;
6
7      if (left < n && arr[left] < arr[smallest])
8          smallest = left;
9
10     if (right < n && arr[right] < arr[smallest])
11         smallest = right;
12
13     if (smallest != i) {
14         int temp = arr[i];
15         arr[i] = arr[smallest];
16         arr[smallest] = temp;
17
18         heapify(arr, n, smallest);
19     }
20 }
21 void heapSort(int arr[], int n) {
22
23     for (int i = n/2 - 1; i >= 0; i--)
24         heapify(arr, n, i);
25
26
27     for (int i = n - 1; i > 0; i--) {
28         int temp = arr[0];
29         arr[0] = arr[i];
30         arr[i] = temp;
31
32         heapify(arr, i, 0);
33     }
34 }
35 int main() {
36     int arr[] = {12, 11, 13, 5, 6, 7};
37     int n = 6;
38
39     heapSort(arr, n);
40
41     printf("Sorted Array:\n");
42     for (int i = 0; i < n; i++)
43         printf("%d ", arr[i]);
44
45     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

```
C bucket.c > ⌂ bucketSort(int [], int)
1  #include <stdio.h>
2
3  void bucketSort(int arr[], int n) {
4      int bucket[10][10], count[10] = {0};
5
6      for (int i = 0; i < n; i++) {
7          int index = arr[i] / 10;
8          bucket[index][count[index]++] = arr[i];
9      }
10
11     for (int i = 0; i < 10; i++) {
12         for (int j = 1; j < count[i]; j++) {
13             int key = bucket[i][j];
14             int k = j - 1;
15             while (k >= 0 && bucket[i][k] > key) {
16                 bucket[i][k + 1] = bucket[i][k];
17                 k--;
18             }
19             bucket[i][k + 1] = key;
20         }
21     }
22     int idx = 0;
23     for (int i = 0; i < 10; i++)
24         for (int j = 0; j < count[i]; j++)
25             arr[idx++] = bucket[i][j];
26 }
27
28 int main() {
29     int arr[] = {42, 32, 33, 52, 37, 47, 51};
30     int n = 7;
31
32     bucketSort(arr, n);
33
34     for (int i = 0; i < n; i++)
35         printf("%d ", arr[i]);
36
37     return 0;
38 }
```

OUTPUT:

```
PS C:\Users\Ganath Avinash\OneDrive\ドキュメント\Back-end\DAA> cd  
32 33 37 42 47 51 52  
PS C:\Users\Ganath Avinash\OneDrive\ドキュメント\Back-end\DAA> []
```

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 > ...
1  #include <stdio.h>
2
3  int queue[20], front = -1, rear = -1;
4  int visited[20];
5
6  void bfs(int graph[20][20], int n, int start) {
7      queue[++rear] = start;
8      visited[start] = 1;
9
10     while (front != rear) {
11         int v = queue[++front];
12         printf("%d ", v);
13
14         for (int i = 0; i < n; i++) {
15             if (graph[v][i] == 1 && visited[i] == 0) {
16                 queue[++rear] = i;
17                 visited[i] = 1;
18             }
19         }
20     }
21 }
22
23 int main() {
24     int n = 4;
25     int graph[20][20] = {
26         {0,1,1,0},
27         {1,0,0,1},
28         {1,0,0,1},
29         {0,1,1,0}
30     };
31
32     bfs(graph, n, 0);
33     return 0;
34 }
```

OUTPUT:

```
● PS C:\Users\Ganath Avinash\OneDrive\ドキュメント\Back-end\DAA> cd  
0 1 2 3  
○ PS C:\Users\Ganath Avinash\OneDrive\ドキュメント\Back-end\DAA> [ ]
```

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 DFS.c > ...
1  #include <stdio.h>
2
3  int visited[20];
4
5  void dfs(int graph[20][20], int n, int v) {
6      printf("%d ", v);
7      visited[v] = 1;
8
9      for (int i = 0; i < n; i++) {
10         if (graph[v][i] == 1 && visited[i] == 0) {
11             dfs(graph, n, i);
12         }
13     }
14 }
15
16 int main() {
17     int n = 4;
18     int graph[20][20] = {
19         {0,1,1,0},
20         {1,0,0,1},
21         {1,0,0,1},
22         {0,1,1,0}
23     };
24
25     dfs(graph, n, 0);
26     return 0;
27 }
```

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 $O(V+E)$.