

Selection Sort

(i) Explain the following sorting algorithm

Consider the following elements are to be sorted in ascending order using selection sort-

6, 2, 11, 7, 5

Selection sort works as-

- It finds the first smallest element (2).
- It swaps it with the first element of the unordered list.
- It finds the second smallest element (5).
- It swaps it with the second element of the unordered list.
- Similarly, it continues to sort the given elements.

(ii) Write down algorithm

Consider the following elements are to be sorted in ascending order-

6, 2, 11, 7, 5

The selection sort algorithm works as illustrated below-

Step-01: For i = 0

[6, 2, 11, 7, 5]

we find minimum element of array and swap it with 1st element of array

[2, 6, 11, 7, 5]

Step-02: For i = 1

[2, 6, 11, 7, 5]

we find minimum element of sub array and swap it with 2nd element of array

[2, 5, 11, 7, 6]

Step-03: For i = 2

[2, 5, 11, 7, 6]

we find minimum element of sub array and swap it with 3rd element of array

[2, 5, 6, 7, 11]

Step-04: For $i = 3$

[2, 5, 6, 7, 11]

we find minimum element of sub array , but there is no need of swap

Step-05: For $i = 4$

Loop gets terminated as 'i' becomes 4.

The state of array after the loops are finished is

[2, 5, 6, 7, 11]

With each loop cycle,

- The minimum element in unsorted sub-array is selected.
- It is then placed at the correct location in the sorted sub-array until array A is completely sorted.

(iii) Evaluate Time and Space complexity

Time Complexity Analysis

Selection sort algorithm consists of two nested loops.
Owing to the two nested loops, it has $O(n^2)$ time complexity.

	Time Complexity
Best Case	n^2
Average Case	n^2
Worst Case	n^2

Space Complexity Analysis

- Selection sort is an in-place algorithm.
- It performs all computation in the original array and no other array is used.
- Hence, the space complexity works out to be $O(1)$.

(iv) Write down code and attach screenshot of output

Let A be an array with n elements. Then, selection sort algorithm used for sorting is as follows-

```
#include <stdio.h>

int main()
{
    int A[] = {6, 2, 11, 7, 5};
    int n = 5;
    int index = 0, temp = 0, i = 0, j = 0;
    for (i = 0; i < n; i++)
    {
        printf("%d\n", A[i]);
    }
    for (i = 0; i < n - 1; i++)
    {
        index = i;
        for (j = i + 1; j < n; j++)
        {
            if (A[j] < A[index])
                index = j;
        }
        temp = A[i];
        A[i] = A[index];
        A[index] = temp;
    }

    printf("\nafter selection sort\n\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\n", A[i]);
    }
}
```

Here,

- i = variable to traverse the array A
- index = variable to store the index of minimum element
- j = variable to traverse the unsorted sub-array
- temp = temporary variable used for swapping

```
t://-//399298:zsh
1 #include <stdio.h>
2
3 int main()
4 {
5     int A[] = {6, 2, 11, 7, 5};
6     int n = 5;
7     int index = 0, temp = 0, i = 0, j = 0;
8     for (i = 0; i < n; i++)
9     {
10         printf("%d\n", A[i]);
11     }
12     for (i = 0; i < n - 1; i++)
13     {
14         index = i;
15         for (j = i + 1; j < n; j++)
16         {
17             if (A[j] < A[index])
18                 index = j;
19         }
20         temp = A[i];
21         A[i] = A[index];
22         A[index] = temp;
23     }
24     printf("\nafter selection sort\n\n");
25     for (i = 0; i < n; i++)
26     {
27         printf("%d\n", A[i]);
28     }
29 }
```

```
18 state:      discharging
19 percentage: 83%
20 ~ gcc selesort.c -o selec && ./selec
21 6
22 2
23 11
24 7
25 5
26
27 9 after selection sort
28
29 7 2
30 6 5
31 5 6
32 4 7
33 3 11
34
35 2 ~ file selesort.c
36 1 selesort.c: C source, ASCII text
37
38 19 ~
39
40 1
41 2
42 3
43 4
44 5
45 6
46 7
47 8
48 9
49 10
50 11
51 12
52 13
53 14
54 15
55 16
56 17
57 18
58 19
59 20
60 21
```

selesort.c c utf-8[unix] 3% ln:1/30 %:1 TERMINAL zsh ln:19/40

E21: Cannot make changes, 'modifiable' is off

i

Bubble Sort

(i) Explain the following sorting algorithm

Bubble sort uses multiple passes (scans) through an array.

- In each pass, bubble sort compares the adjacent elements of the array.
- It then swaps the two elements if they are in the wrong order.
- In each pass, bubble sort places the next largest element to its proper position.
- In short, it bubbles down the largest element to its correct position.

(ii) Write down algorithm

Consider the array A-

6, 2, 11, 7, 5

Step-01:

- We have pass=1 and i=0.
- We perform the comparison $A[0] > A[1]$ and swaps if the 0th element is greater than the 1th element.
- Since $6 > 2$, so we swap the two elements.

[6, 2, 11, 7, 5] ----> [2, 6, 11, 7, 5]

Step-02:

- We have pass=1 and i=1.
- We perform the comparison $A[1] > A[2]$ and swaps if the 1th element is greater than the 2th element.
- Since $6 < 11$, so no swapping is required.

[2, 6, 11, 7, 5] ----> [2, 6, 11, 7, 5]

Step-03:

- We have pass=1 and i=2.
- We perform the comparison $A[2] > A[3]$ and swaps if the 2nd element is greater than the 3rd element.
- Since $11 > 7$, so we swap the two elements.

[2, 6, 11, 7, 5]

Experiment no 1

---->

[2, 6, 7, 11, 5]

Step-04:

- We have pass=1 and i=3.
- We perform the comparison $A[3] > A[4]$ and swaps if the 3rd element is greater than the 4th element.
- Since $11 > 5$, so we swap the two elements.

[2, 6, 11, 7, 5]

---->

[2, 6, 7, 5, 11]

Finally after the first pass, we see that the largest element 11 reaches its correct position.

Step-05:

- Similarly after pass=2, element 7 reaches its correct position.
- The modified array after pass=2 is shown below-

[2, 6, 11, 7, 5]

---->

[2, 6, 5, 7, 11]

Step-06:

- Similarly after pass=3, element 6 reaches its correct position.
- The modified array after pass=3 is shown below-

[2, 6, 11, 7, 5]

---->

[2, 5, 6, 7, 11]

Step-07:

- No further improvement is done in pass=4.
- This is because at this point, elements 2 and 5 are already present at their correct positions.
- The loop terminates after pass=4.
- Finally, the array after pass=4 is shown below-

[2, 5, 6, 7, 11]

(iii) Evaluate Time and Space complexity

Time Complexity Analysis

- Bubble sort uses two loops- inner loop and outer loop.
- The inner loop deterministically performs $O(n)$ comparisons.

Worst Case-

- In worst case, the outer loop runs $O(n)$ times.
- Hence, the worst case time complexity of bubble sort is $O(n \times n) = O(n^2)$.

Best Case-

- In best case, the array is already sorted but still to check, bubble sort performs $O(n)$ comparisons.
- Hence, the best case time complexity of bubble sort is $O(n)$.

Average Case-

- In average case, bubble sort may require $(n/2)$ passes and $O(n)$ comparisons for each pass.
- Hence, the average case time complexity of bubble sort is $O(n/2 \times n) = O(n^2)$.

The following table summarizes the time complexities of bubble sort in each case-

	Time Complexity
Best Case	$O(n)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

From here, it is clear that bubble sort is not at all efficient in terms of time complexity of its algorithm.

Space Complexity Analysis

- Bubble sort uses only a constant amount of extra space for variables like flag, i, n.
- Total comparisons in Bubble sort is: $n (n - 1) / 2 \approx n^2 - n$
- Hence, the space complexity of bubble sort is $O(1)$.
- It is an in-place sorting algorithm i.e. it modifies elements of the original array to sort the given array.

(iv) Write down code and attach screenshot of output

```
#include <stdio.h>

void swap(int x, int y, int A[])
{
    int temp = A[x];
    A[x] = A[y];
    A[y] = temp;
}

int main()
{
    int A[] = {6, 2, 11, 7, 5};
    int n = 5;
    int i = 0;

    for (i = 0; i < n; i++)
    {
        printf("%d\n", A[i]);
    }
    for(int pass=1 ; pass<=n-1 ; ++pass) // Making passes through array
    {
        for(int i=0 ; i<=n-2 ; ++i)
        {
            if(A[i] > A[i+1]) // If adjacent elements are in wrong order
                swap(i,i+1,A); // Swap them
        }
    }
    //swap function : Exchange elements from array A at position x,y

    printf("\nafter selection sort\n\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\n", A[i]);
    }

}
```

Here,

- pass : Variable to count the number of passes that are done till now
- n : Size of the array
- i : Variable to traverse the array A
- swap() : Function to swap two numbers from the array
- x,y : Indices of the array that needs to be swapped

```
100 72% 2 KB/s 9% 51% 01:29 Fri, 06 Aug Alacrity
```

```
1 bubble.c
2 #include <stdio.h>
3
4 void swap(int x, int y, int A[])
5 {
6     int temp = A[x];
7     A[x] = A[y];
8     A[y] = temp;
9 }
10 int main()
11 {
12     int A[] = {6, 2, 11, 7, 5};
13     int n = 5;
14     int i = 0;
15     for (i = 0; i < n; i++)
16     {
17         printf("%d\n", A[i]);
18     }
19     for(int pass=1; pass<=n-1; ++pass) // Making passes through array
20     {
21         for(int i=0; i<=n-2; ++i)
22         {
23             if(A[i] > A[i+1]) // If adjacent elements are in wrong order
24                 swap(i,i+1,A); // Swap them
25         }
26     }
27     //swap function : Exchange elements from array A at position x,y
28     printf("\nafter selection sort\n\n");
29     for (i = 0; i < n; i++)
30     {
31         printf("%d\n", A[i]);
32     }
33 }
34
35 }
```

```
24 state:      discharging
25 percentage: 82%
26 gcc bubble.c -o bubble && ./bubble
27 6
28 2
29 11
30 7
31 5
32
33 after selection sort
34 2
35 5
36 6
37 11
38 7
39
40 afetch
41
42 USER umang
43 OS EndeavourOS
44 KERNEL 5.13.7-zen1-1-zen
45 UPTIME 9h 25m
46 SHELL zsh
47 PKGS 1500
48
49 1
50 2
51 3
52 4
53 5
54 6
55 7
56 8
57 9
58 10
59 11
60 12
61 13
62 14
63 15
64 16
65 17
66 18
67 19
68 20
69 21
70 22
71 23
72 24
73 25
```

```
NORMAL bubble.c c utf-8[unix] 2% ln:1/36 618 TERMINAL zsh ln:25/50
```