

## Insertion and Selection Sort

Md. Abu Sufyan  
Roll: 2003085  
Department: CSE

Tameem Rahman  
Roll: 2003089  
Department: CSE

Partha Paul  
Roll: 2003078  
Department: CSE

Atik Mohtasin Rahi  
Roll: 2003118  
Department: CSE

Barkatulla Barik  
Roll: 2003071  
Department: CSE

Mir Ashikur Rahman  
Roll: 2003109  
Department: CSE

Rajshahi University of Engineering & Technology

January 22, 2024

# Welcome

# Welcome to our Presentation!

# Outline

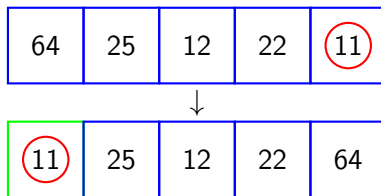
subsectionstyle=hide

# Selection Sort Overview

## Selection Sort

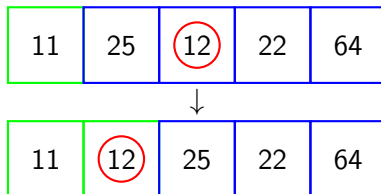
# Selection Sort: First Pass

- For the first position in the sorted array, traverse the entire array.
- Find the minimum value (11) and swap it with the element at the first position (64).



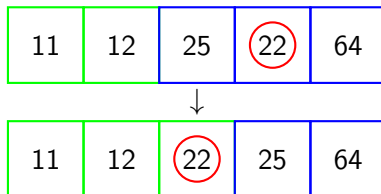
## Selection Sort: Second Pass

- For the second position, find the second minimum value (12) and swap it with the element at the second position (25).



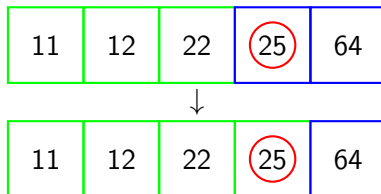
## Selection Sort: Third Pass

- For the third position, find the third minimum value (22) and swap it with the element at the third position (25).



# Selection Sort: Fourth Pass

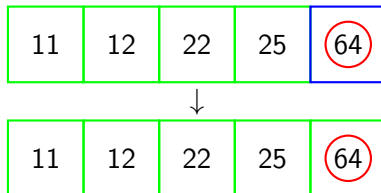
- For the fourth position, find the fourth minimum value (25) and never swap it.





## Selection Sort: Fifth Pass

- The largest value (64) is automatically placed at the last position.



# Selection Sort Algorithm: Overview

---

## Algorithm 1: Selection Sort

---

**Data:** Array  $arr$  of size  $n$

**Result:** Sorted array  $arr$

```
1 for  $i \leftarrow 0$  to  $n - 1$  do
2    $min\_idx \leftarrow i$ ;
3   for  $j \leftarrow i + 1$  to  $n$  do
4     if  $arr[j] < arr[min\_idx]$  then
5        $min\_idx \leftarrow j$ ;
6   if  $min\_idx \neq i$  then
7     Swap( $arr[min\_idx]$ ,  $arr[i]$ );
```

---

# Selection Sort: C++ Code

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;
    for (i = 0; i < n - 1; i++) {
        min_idx = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx])
                min_idx = j;
        }
        if (min_idx != i)
            swap(arr[min_idx], arr[i]);
    }
}
```

# Insertion Sort

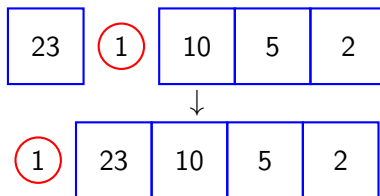
## Insertion Sort

# Insertion Sort: Step-by-Step

- For each element, insert it into its correct position in the sorted portion of the array.
- Shift elements greater than the key to the right.
- Repeat until the entire array is sorted.

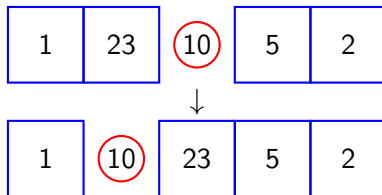
# Insertion Sort: First Pass

- Initially, the first two elements of the array are compared in insertion sort.
- Here, 23 is greater than 1 hence they are not in the ascending order and 23 is not at its correct position. Thus, swap 1 and 23.



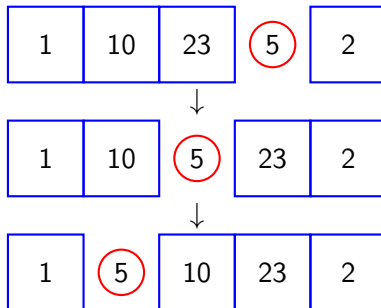
## Insertion Sort: Second Pass

- Here, 23 is greater than 10 hence they are not in the ascending order and 10 is not at its correct position. Thus, swap 1 and 23. 10 also stored in a sorted sub-array along with 1



# Insertion Sort: Third Pass

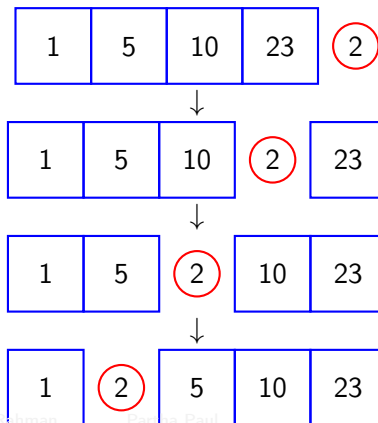
- Here, 5 isn't in correct position. So 5 has to be swapped with its previous position until 5 isn't greater than the previous value.





# Insertion Sort: Fourth Pass

- Here, 2 isn't in correct position. To place 2 in correct position, we have to follow the same procedure as third pass.



# Insertion Sort: Visualization

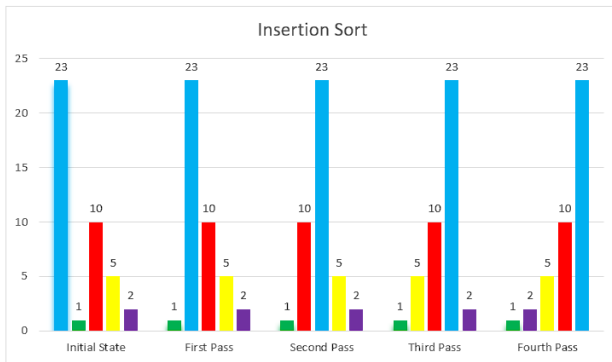


Figure: Insertion Sort Visualization

# Insertion Sort Algorithm: Overview

---

## Algorithm 2: Insertion Sort

---

**Data:** Array  $arr$  of size  $n$

**Result:** Sorted array  $arr$

```
1 for  $i \leftarrow 1$  to  $n$  do
2    $key \leftarrow arr[i]$ ;
3    $j \leftarrow i - 1$ ;
4   while  $j \geq 0$  and  $arr[j] > key$  do
5      $arr[j + 1] \leftarrow arr[j]$ ;
6      $j \leftarrow j - 1$ ;
7    $arr[j + 1] \leftarrow key$ ;
```

---

# Insertion Sort: C++ Code

```
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;  
    }  
}
```

# Result and Complexity Analysis

Size	Insertion (s)	Selection (s)
10	0.000011	0.000019
50	0.000176	0.000155
100	0.000412	0.000363
500	0.004555	0.004457
1000	0.020648	0.018857
5000	0.528669	0.485517
10000	2.105405	1.965638
50000	61.877976	55.541591
100000	238.821475	271.276342

**Table:** Performance Comparison of Insertion Sort and Selection Sort

# Graph of Complexity

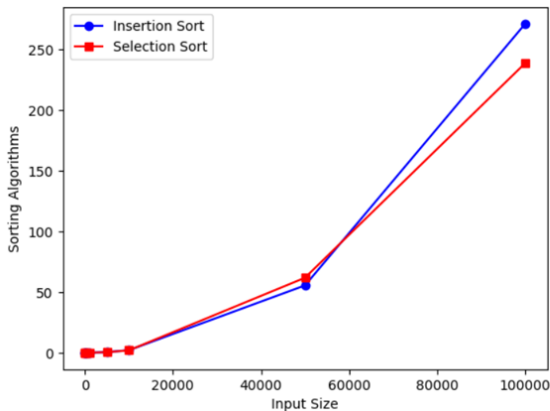


Figure: Complexity Visualization Graph

# References

# Conclusion

- Insertion Sort is a simple and intuitive sorting algorithm.
- It efficiently builds the final sorted array one element at a time.
- While not as efficient on large datasets as more advanced algorithms, it performs well on small datasets or nearly sorted datasets.



# Questions & Answers

Any Questions?

# Thanks

# Thank You for Your Attention!