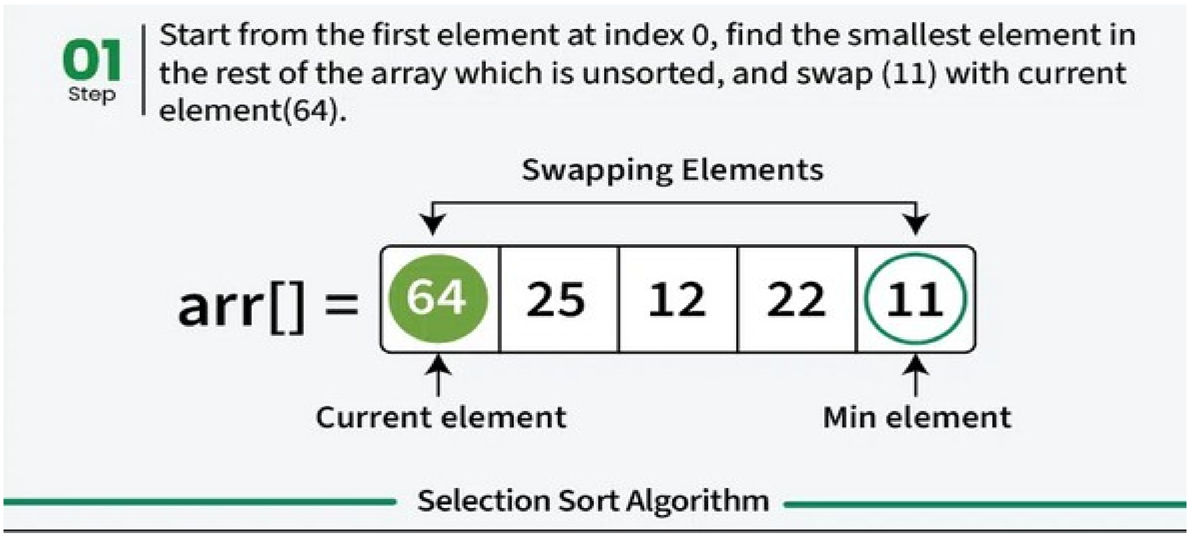
SELECTION SORT

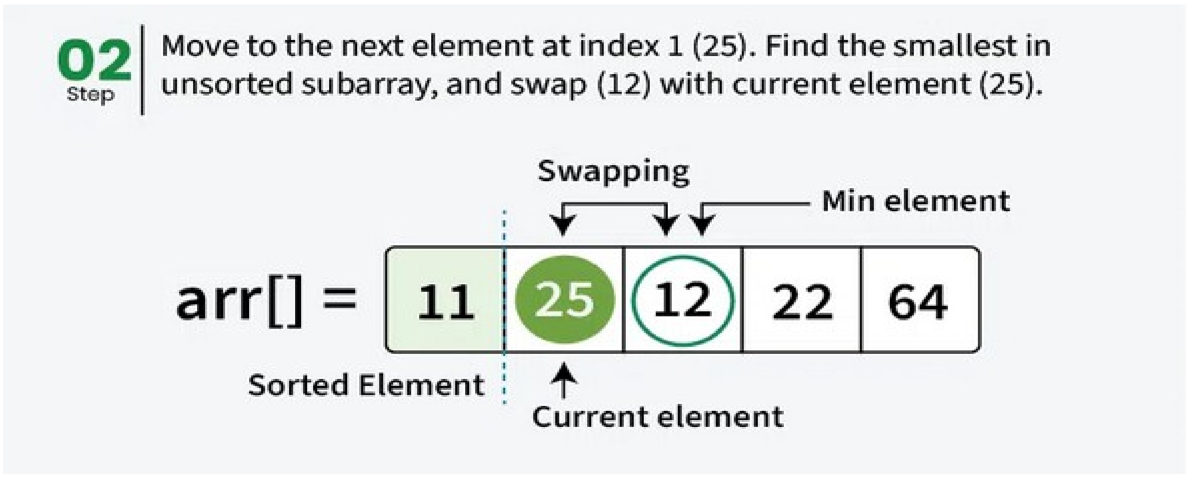
Algorithm:

1. First we find the smallest element and swap it with the first element. This way we get the smallest element at its correct position.

2. Then we find the smallest among remaining elements (or second smallest) and swap it with the second element.

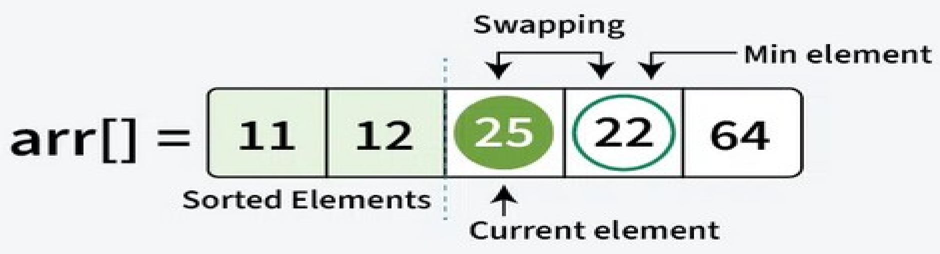
3. We keep doing this until we get all elements moved to correct position.


01 Step
Start from the first element at index 0, find the smallest element in the rest of the array which is unsorted, and swap (11) with current element(64).
Swapping Elements
arr[] = 64 25 12 22 11
Current element
Min element
Selection Sort Algorithm


02 Step
Move to the next element at index 1 (25). Find the smallest in unsorted subarray, and swap (12) with current element (25).
Swapping
Min element
arr[] = 11 25 12 22 64
Sorted Element
Current element

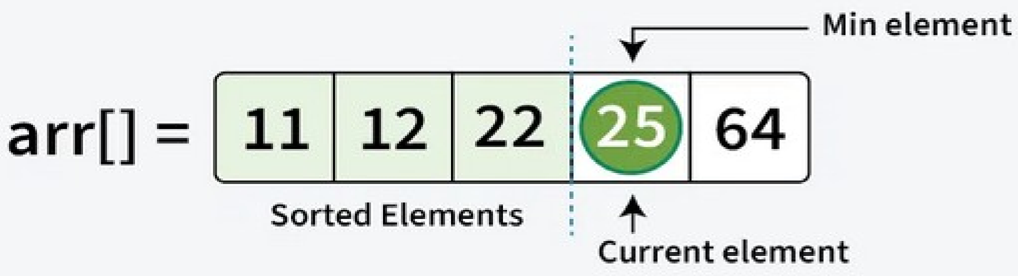
03 Step

Move to element at index 2 (25). Find the minimum element from unsorted subarray, Swap (22) with current element (25).



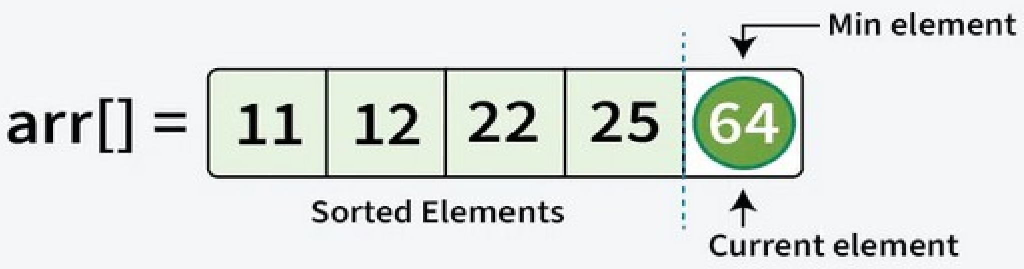
04 Step

Move to element at index 3 (25), find the minimum from unsorted subarray and swap (25) with current element (25).



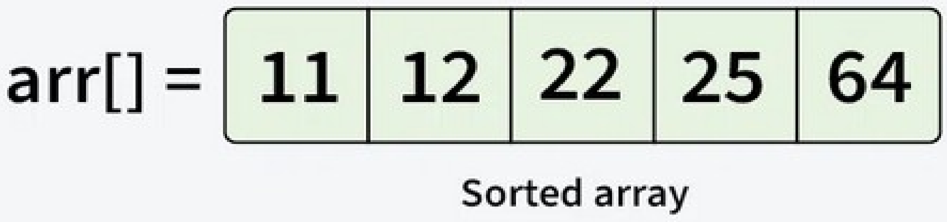
05 Step

Move to element at index 4 (64), find the minimum from unsorted subarray and swap (64) with current element (64).



06 Step

We get the sorted array at the end.



INPUT:

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n - 1):

        # Assume the current position holds
        # the minimum element
        min_idx = i

        # Iterate through the unsorted portion
        # to find the actual minimum
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:

                # Update min_idx if a smaller element is found
                min_idx = j

        # Move minimum element to its
        # correct position
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

def print_array(arr):
    for val in arr:
        print(val, end=" ")
    print()

arr = [64, 25, 12, 22, 11]
print("Original array: ", end="")
print_array(arr)
```

```
selection_sort(arr)
print("Sorted array: ", end="")
print_array(arr)
```

OUTPUT:

```
Original array: 64 25 12 22 11
Sorted array: 11 12 22 25 64
```

APPLICATIONS IN AI:

1. **Sorting Small Datasets**: Can be used to sort small datasets before applying AI models or algorithms, like k-Nearest Neighbors or clustering.
2. **Feature Selection**: Used for sorting features based on metrics like information gain, particularly in small datasets or initial stages of model development.
3. **Search Algorithms**: In small search spaces, selection sort can rank or prioritize candidate solutions, such as in greedy algorithms.
4. **Improving Interpretability**: Helps in sorting outcomes or ranking in tasks like recommendation systems and decision tree-based models.
5. **Edge AI Devices**: Suitable for use in embedded systems or low-power devices with limited resources when dealing with small datasets.
6. **Educational Use**: Often used to teach sorting concepts and basic algorithms in AI-related courses or during algorithm development and debugging.

7. **Data Preprocessing**: Organizing data points before further processing steps in AI pipelines, such as normalization or transformation.
8. **Memory-Constrained Environments**: Useful in low-memory AI applications due to its in-place sorting.
9. **Simulation and Games**: Ranking game states or moves in AI-driven games or simulations.

- Selection sort is useful in **small-scale**, **memory-limited**, or **real-time decision-making** AI applications, where its simplicity and in-place nature provide advantages.
- **Inefficiency**: Selection sort has O(n2) complexity, making it inefficient for large datasets, where faster algorithms are preferred.

PRIMS ALGORITHM:

1. **Initialization**:

- Start with any arbitrary vertex, and set its key value to 0 (this vertex will be the starting point of the MST).
- Set the key value of all other vertices to a large number (infinity).
- Mark all vertices as unvisited.
- Set the parent of each vertex as `-1` (since no vertex is connected initially).

2. **Process the Minimum Key Vertex**:

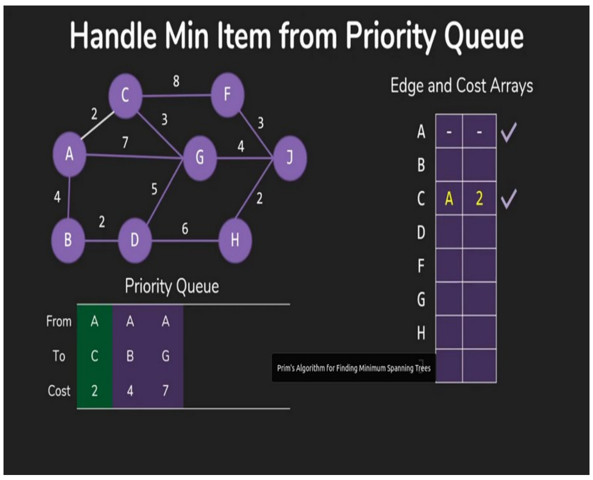
- Select the vertex with the smallest key value that is **not yet included in the MST** (use the key array).
- Include this vertex in the MST.
- For each adjacent vertex, if it is not in the MST and the weight of the edge to that vertex is smaller than the current key value of that vertex, update the key value and set the parent to the current vertex.

3. **Repeat**:

   Repeat the process until all vertices are included in the MST.

4. **Print the Result**:

   Once all vertices are included in the MST, the parent array will contain the tree structure. The parent of each vertex is the vertex that connects it to the MST.



Handle Min Item from Priority Queue

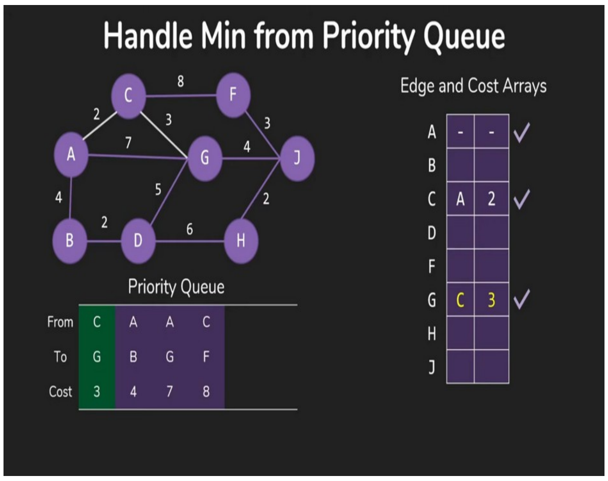
Priority Queue

From	A	A	A
To	C	B	G
Cost	2	4	7

Edge and Cost Arrays

A	-	-	✓
B			
C	A	2	✓
D			
F			
G			
H			
J			

Prim's Algorithm for Finding Minimum Spanning Trees



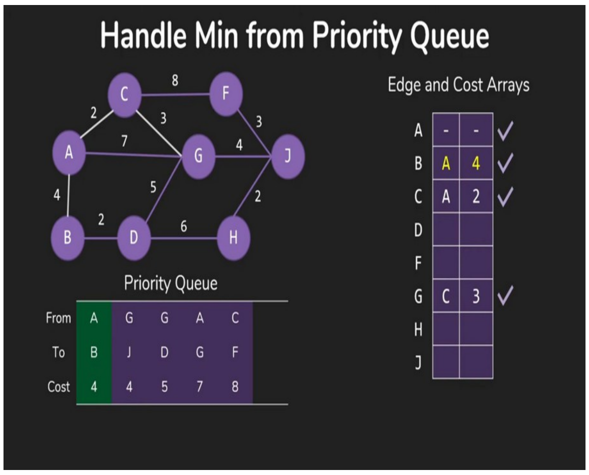
Handle Min from Priority Queue

Priority Queue

From	C	A	A	C
To	G	B	G	F
Cost	3	4	7	8

Edge and Cost Arrays

A	-	-	✓
B			
C	A	2	✓
D			
F			
G	C	3	✓
H			
J			



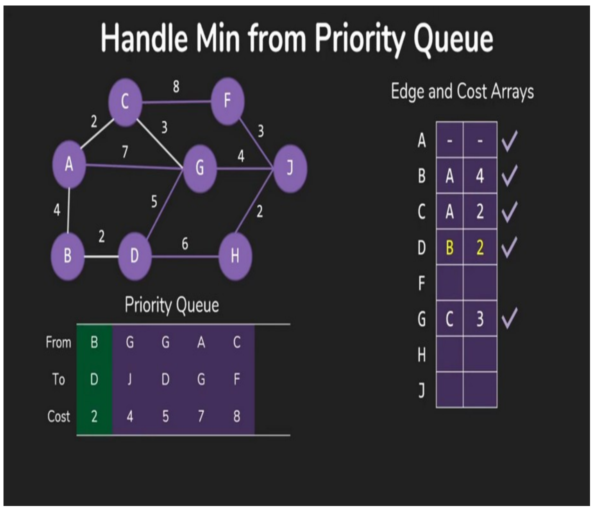
Handle Min from Priority Queue

Edge and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D			
F			
G	C	3	✓
H			
J			

Priority Queue

From	A	G	G	A	C
To	B	J	D	G	F
Cost	4	4	5	7	8



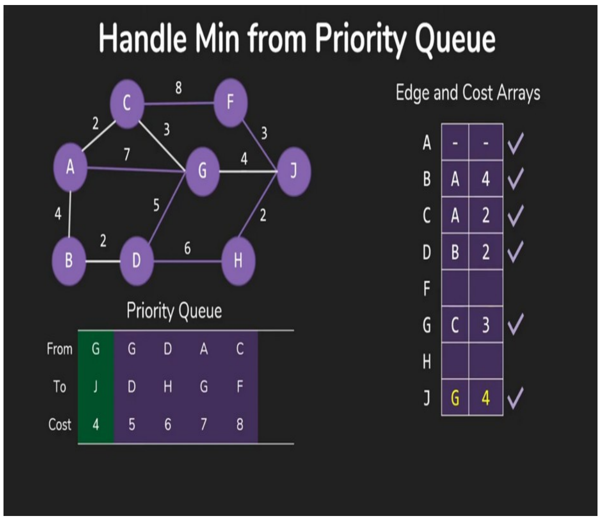
Handle Min from Priority Queue

Edge and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	2	✓
F			
G	C	3	✓
H			
J			

Priority Queue

From	B	G	G	A	C
To	D	J	D	G	F
Cost	2	4	5	7	8



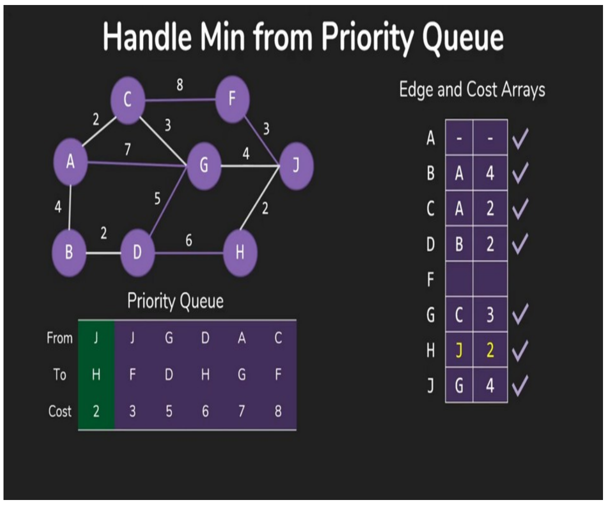
Handle Min from Priority Queue

Edge and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	2	✓
F			
G	C	3	✓
H			
J	G	4	✓

Priority Queue

From	G	G	D	A	C
To	J	D	H	G	F
Cost	4	5	6	7	8



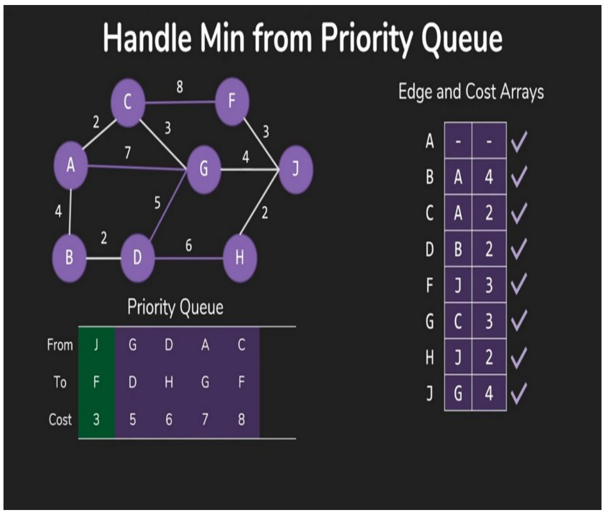
Handle Min from Priority Queue

Edge and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	2	✓
F			
G	C	3	✓
H	J	2	✓
J	G	4	✓

Priority Queue

From	J	J	G	D	A	C
To	H	F	D	H	G	F
Cost	2	3	5	6	7	8



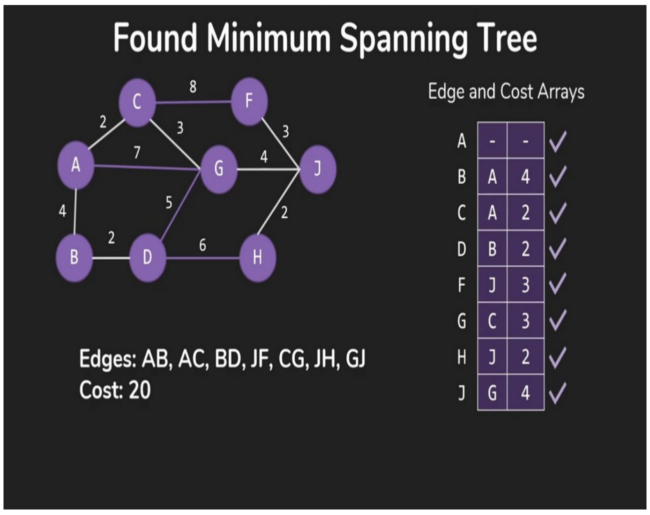
Handle Min from Priority Queue

Edge and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	2	✓
F	J	3	✓
G	C	3	✓
H	J	2	✓
J	G	4	✓

Priority Queue

From	J	G	D	A	C
To	F	D	H	G	F
Cost	3	5	6	7	8



Found Minimum Spanning Tree

Edge and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	2	✓
F	J	3	✓
G	C	3	✓
H	J	2	✓
J	G	4	✓

Edges: AB, AC, BD, JF, CG, JH, GJ
Cost: 20