Marwadi
University

01CE1301 – Data Structure

Unit - 4
Sorting & Searching Techniques

Outline

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Sequential/Linear Search
- Binary Search

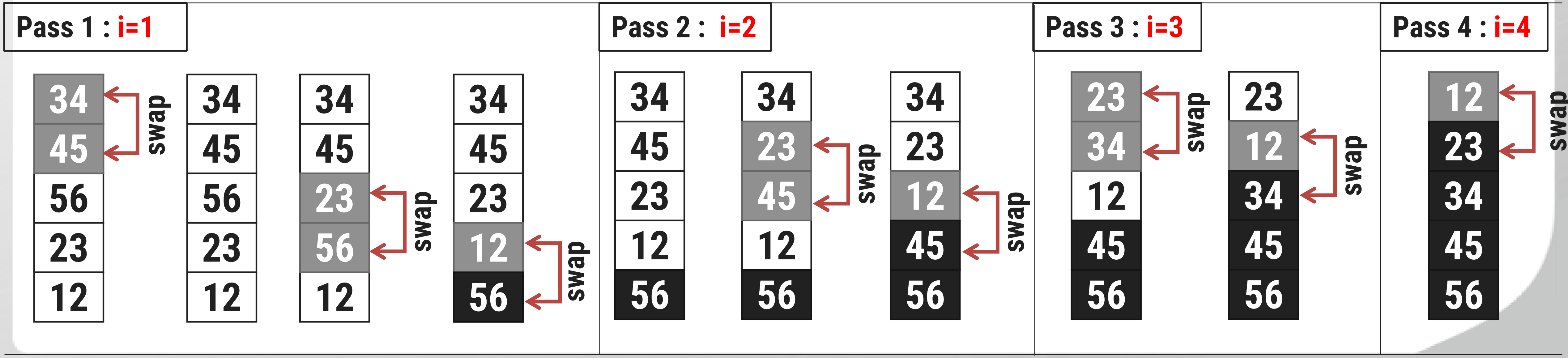
Bubble Sort

Bubble Sort

```
Procedure bubble (T[1…n])
for i ← 0 to n-1 do  //Decide Number of passes
    for j ← 0 to n-i do  // Check the index
        if T[j] > T[j+1] then // Comparison of elements
            T[j] ↔ T[j+1]
        end if
    end for
end for
```

Unsorted Array

45	34	56	23	12

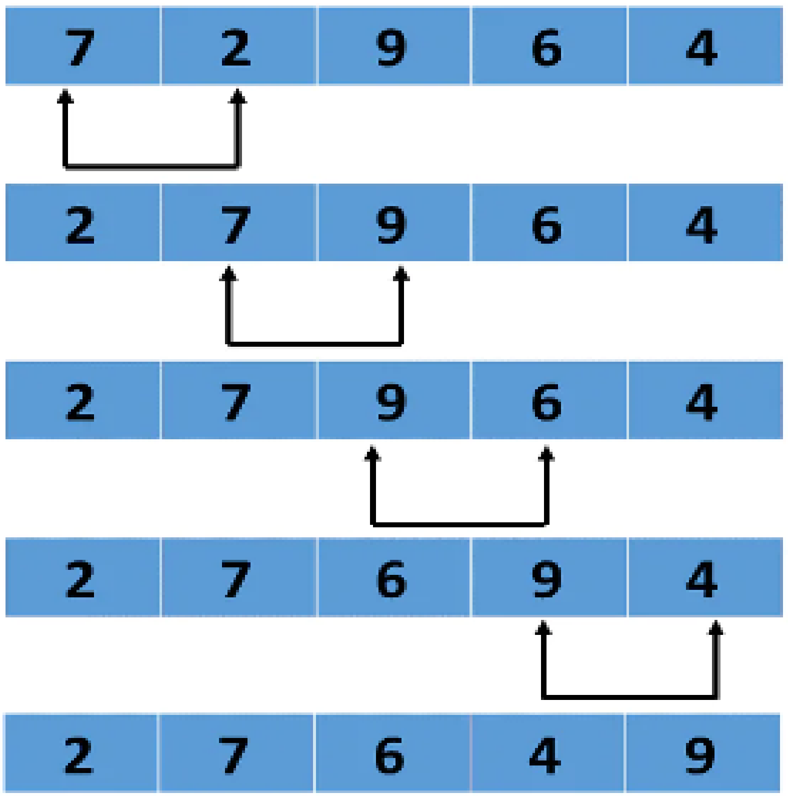


Bubble Sort

```
Procedure bubble (T[1…n])
for i ← 0 to n-1 do  //Decide Number of passes
      for j ← 0 to n-i do  // Check the index
            if T[j] > T[j+1] then // Comparison of elements
                  T[j] ↔ T[j+1]
            end if
      end for
end for
```

7	2	9	6	4

First Pass → i=1



Switch because 7 is greater than 2

No switch is required because 7 is less than 9

Switch because 9 is greater than 6

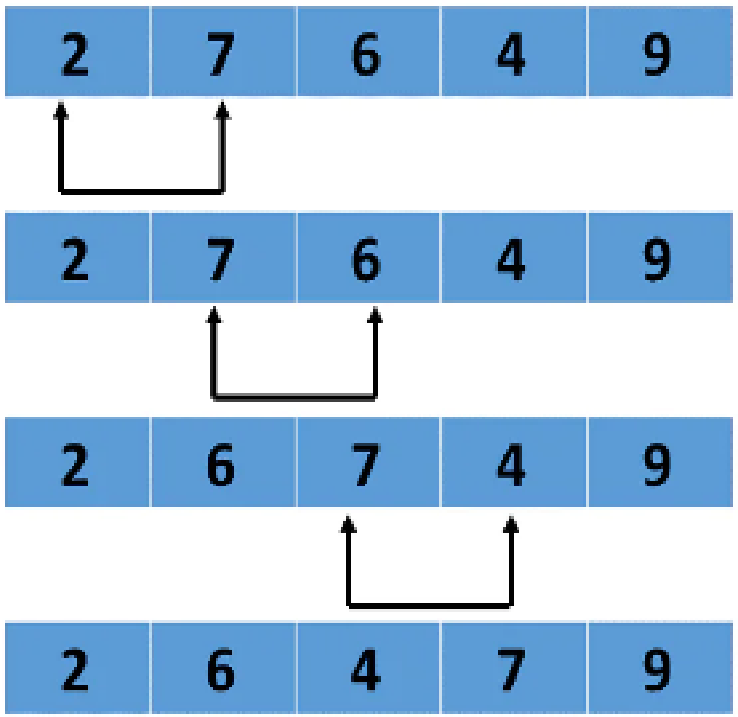
Switch because 9 is greater than 4

Bubble Sort

```
Procedure bubble (T[1…n])
for i ← 0 to n-1 do  //Decide Number of passes
    for j ← 0 to n-i do  // Check the index
        if T[j] > T[j+1] then // Comparison of elements
            T[j] ↔ T[j+1]
        end if
    end for
end for
```

7	2	9	6	4

Second Pass → i=2



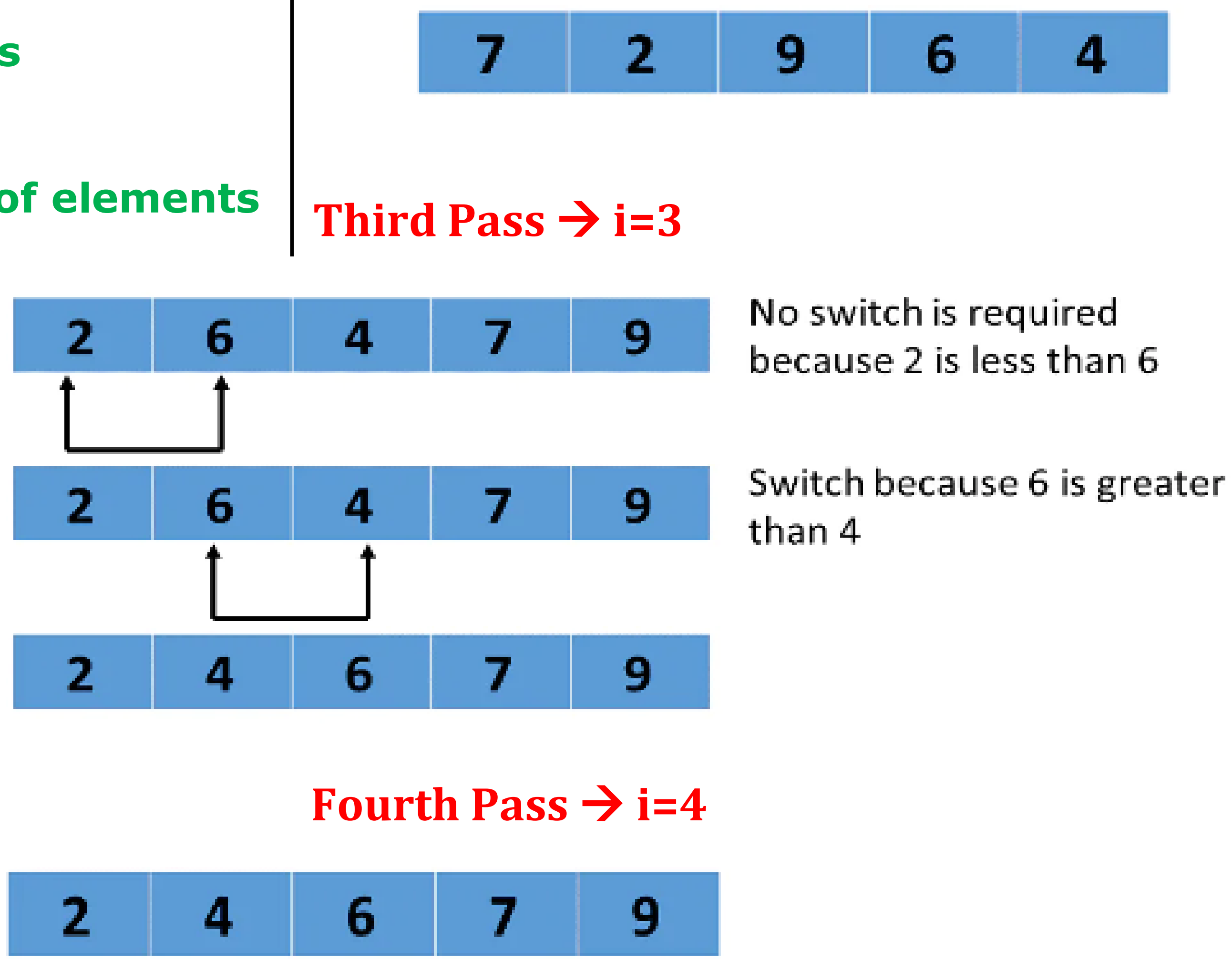
No switch is required because 2 is less than 7

Switch because 7 is greater than 6

Switch because 7 is greater than 4

Bubble Sort

```
Procedure bubble (T[1…n])
for i ← 0 to n-1 do  //Decide Number of passes
    for j ← 0 to n-i do  // Check the index
        if T[j] > T[j+1] then // Comparison of elements
            T[j] ↔ T[j+1]
        end if
    end for
end for
```



Selection Sort

Selection Sort

```
Procedure selection (T[1…n])
for i ← 0 to n-1 do
    min ← i
    for j ← i+1 to n do
        if T[j] < T[min] then
            min ← j
        end if
    end for
    T[min] ↔ T[i]
end for
```

Unsorted Array

5	1	12	-5	16	2	12	14

Assign Index

Unsorted Array

5	1	12	-5	16	2	12	14
0	1	2	3	4	5	6	7

As per given array Min index = 0, value = 5

Step 1 :

Unsorted Array (elements 0 to 7)

-5	1	12	5	16	2	12	14
0	1	2	3	4	5	6	7

Swap

Find min value from Unsorted array

Index = 3, value = -5

Selection Sort

Marwadi University

```
Procedure selection (T[1…n])
for i ← 0 to n-1 do
    min ← i
    for j ← i+1 to n do
        if T[j] < T[min] then
            min ← j
        end if
    end for
    T[min] ↔ T[i]
end for
```

Step 2 :

Unsorted Array (elements 1 to 7)

-5	1	12	5	16	2	12	14
0	1	2	3	4	5	6	7

As per step -1:
Min index = 1, value = 1

Find min value from Unsorted array
Index = 1, value = 1

No Swapping as min value is already at right place

Step 3 :

Unsorted Array (elements 2 to 7)

-5	1	2	5	16	12	12	14
0	1	2	3	4	5	6	7

Swap

As per step-2:
Min index = 2, value = 12

Find min value from Unsorted array
Index = 5, value = 2

Selection Sort


```
Procedure selection (T[1…n])
for i ← 0 to n-1 do
    min ← i
    for j ← i+1 to n do
        if T[j] < T[min] then
            min ← j
        end if
    end for
    T[min] ↔ T[i]
end for
```

Step 4 :

Unsorted Array
(elements 3 to 7)

-5	1	2	5	16	12	12	14
0	1	2	3	4	5	6	7

As per step-3:
Min index = 3, value = 5

Find min value from
Unsorted array
Index = 3, value = 5

No Swapping as min value is already at right place

Step 5 :

Unsorted Array
(elements 5 to 7)

-5	1	2	5	12	16	12	14
0	1	2	3	4	5	6	7

Swap

As per step-4:
Min index = 4, value = 16

Find min value from
Unsorted array
Index = 5, value = 12

Selection Sort


```
Procedure selection (T[1…n])
for i ← 0 to n-1 do
    min ← i
    for j ← i+1 to n do
        if T[j] < T[min] then
            min ← j
        end if
    end for
    T[min] ↔ T[i]
end for
```

Step 6 :

Unsorted Array (elements 5 to 7)

-5	1	2	5	12	12	16	14
0	1	2	3	4	5	6	7

Swap

As per step-5:
Min index = 5, value = 16

Find min value from Unsorted array
Index = 6, value = 12

Step 7 :

Unsorted Array (elements 6 to 7)

-5	1	2	5	12	12	14	16
0	1	2	3	4	5	6	7

Swap

As per step-6:
Min index = 6, value = 16

Find min value from Unsorted array
Index = 7, value = 14

Selection Sort

```
Procedure selection (T[1…n])
for i ← 0 to n-1 do
    min ← i
    for j ← i+1 to n do
        if T[j] < T[min] then
            min ← j
        end if
    end for
    T[min] ↔ T[i]
end for
```

Step 8 :

-5	1	2	5	12	12	14	16
0	1	2	3	4	5	6	7

As per step -7:
Min index = 7, value = 16

No Swapping as min value is already at right place

Selection Sort

```
Procedure selection (T[1…n])
for i ← 0 to n-1 do
    min ← i
    for j ← i+1 to n do
        if T[j] < T[min] then
            min ← j
        end if
    end for
    T[min] ↔ T[i]
end for
```

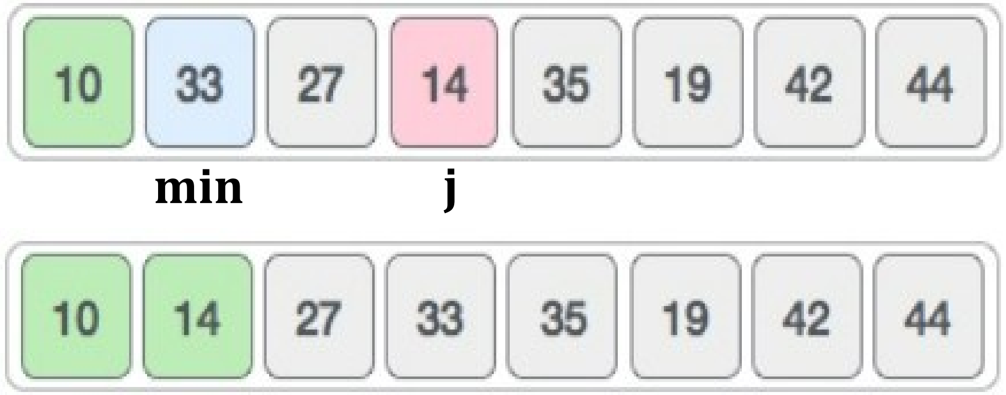
Unsorted Array

First Pass: i=0, min = 0

Selection Sort


```
Procedure selection (T[1…n])
for i ← 0 to n-1 do
    min ← i
    for j ← i+1 to n do
        if T[j] < T[min] then
            min ← j
        end if
    end for
    T[min] ↔ T[i]
end for
```

Second Pass: i=1, min = 1



Third Pass: i=2, min = 2

Selection Sort

```
Procedure selection (T[1…n])
for i ← 0 to n-1 do
      min ← i
      for j ← i+1 to n do
            if T[j] < T[min] then
                  min ← j
            end if
      end for
      T[min] ↔ T[i]
end for
```

Fourth Pass: i=3, min = 3

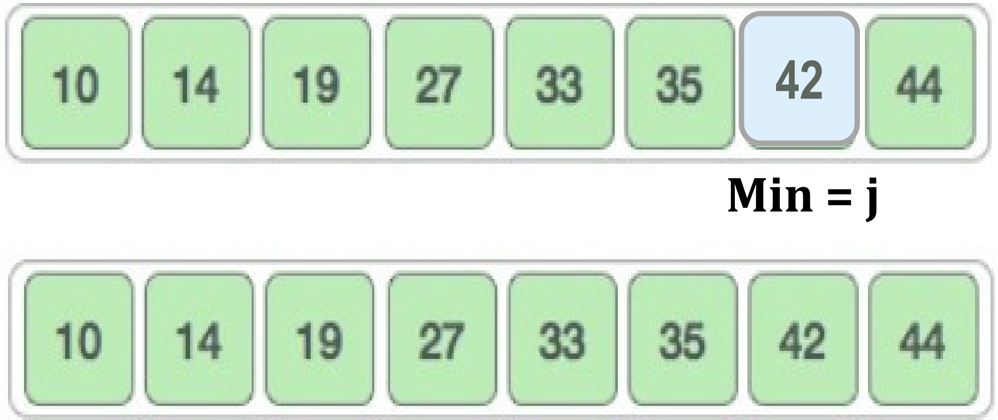
Fifth Pass: i=4, min = 4

Selection Sort

```
Procedure selection (T[1…n])
for i ← 0 to n-1 do
        min ← i
        for j ← i+1 to n do
                if T[j] < T[min] then
                        min ← j
                end if
        end for
        T[min] ↔ T[i]
end for
```

Sixth Pass: i=5, min = 5

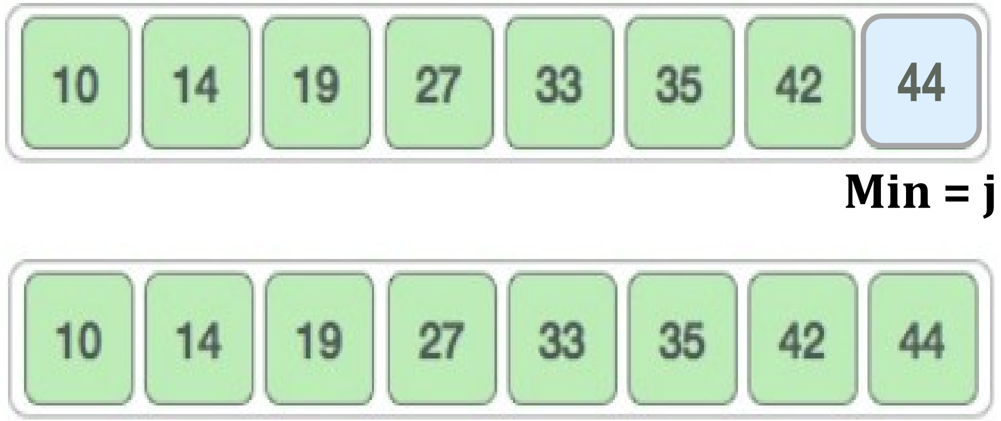
Seventh Pass: i=6, min = 6



Selection Sort

```
Procedure selection (T[1…n])
for i ← 0 to n-1 do
    min ← i
    for j ← i+1 to n do
        if T[j] < T[min] then
            min ← j
        end if
    end for
    T[min] ↔ T[i]
end for
```

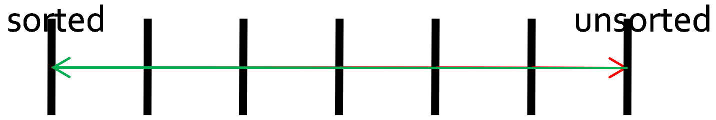
Eighth Pass: i=7, min = 7

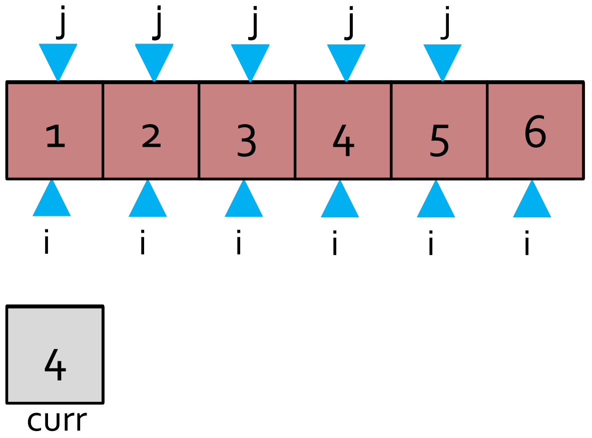


Insertion Sort

Insertion Sort

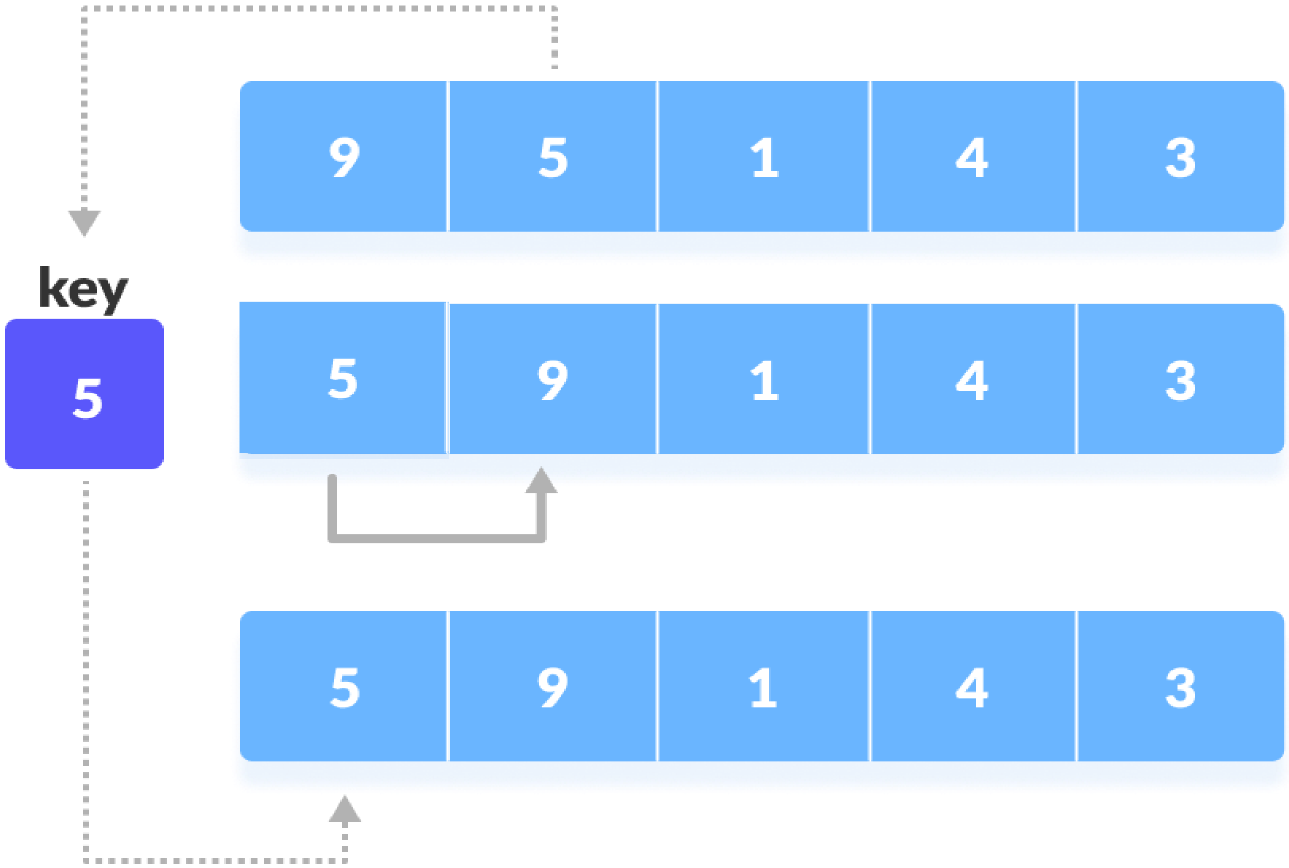

```
Procedure insertion(T[1…n])
for i ← 1 to n do
    curr ← T[i]
    j ← i-1
    while j >= 0 and T[j] > curr do
        T[j+1] ← T[j]
        j ← j-1
    end while
    T[j+1] ← curr;
end for
```





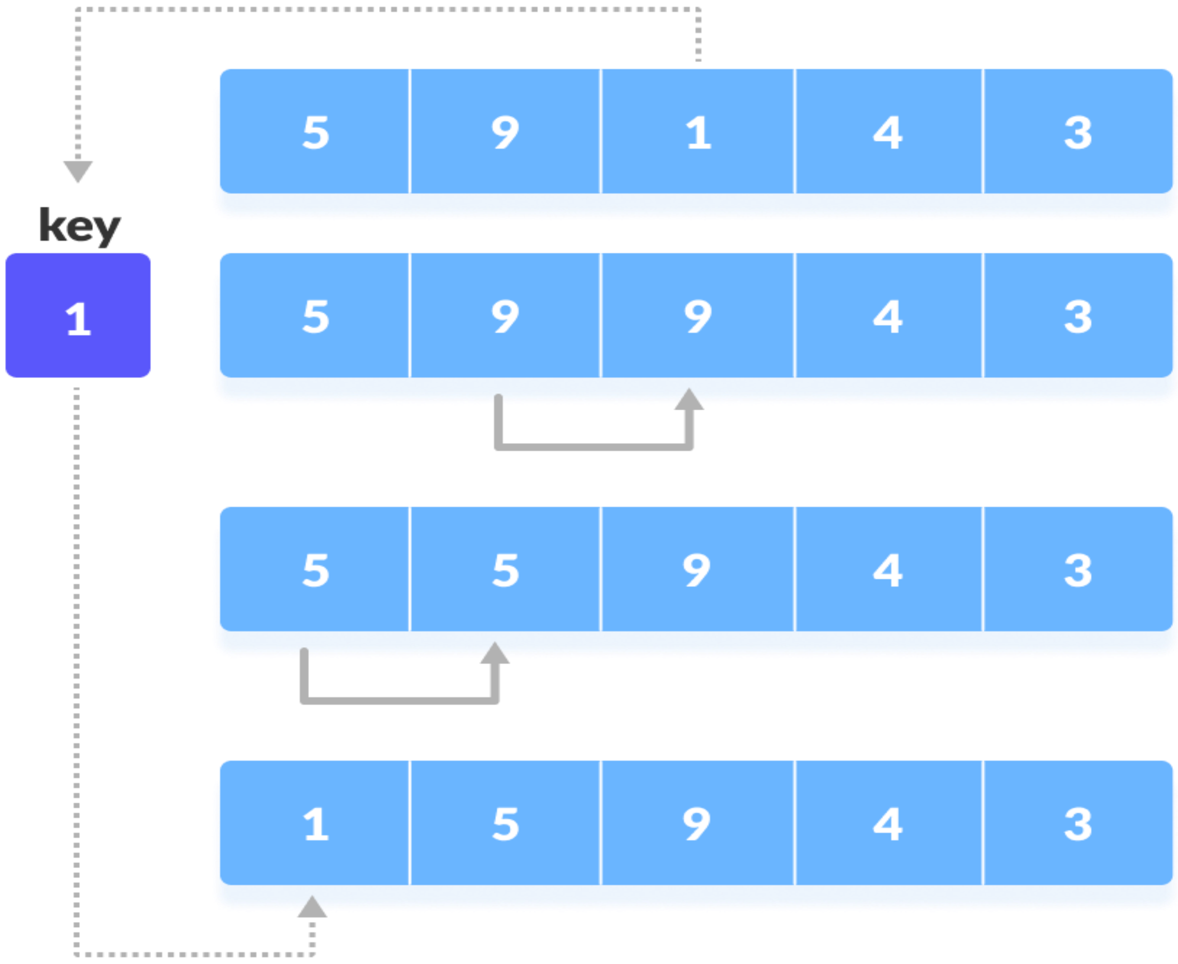
Insertion Sort

```
Procedure insertion(T[1…n])
for i ← 1 to n do
        curr ← T[i]
        j ← i-1
        while j >= 0 and T[j] > curr do
                T[j+1] ← T[j]
                j ← j-1
        end while
        T[j+1] ← curr;
end for
```



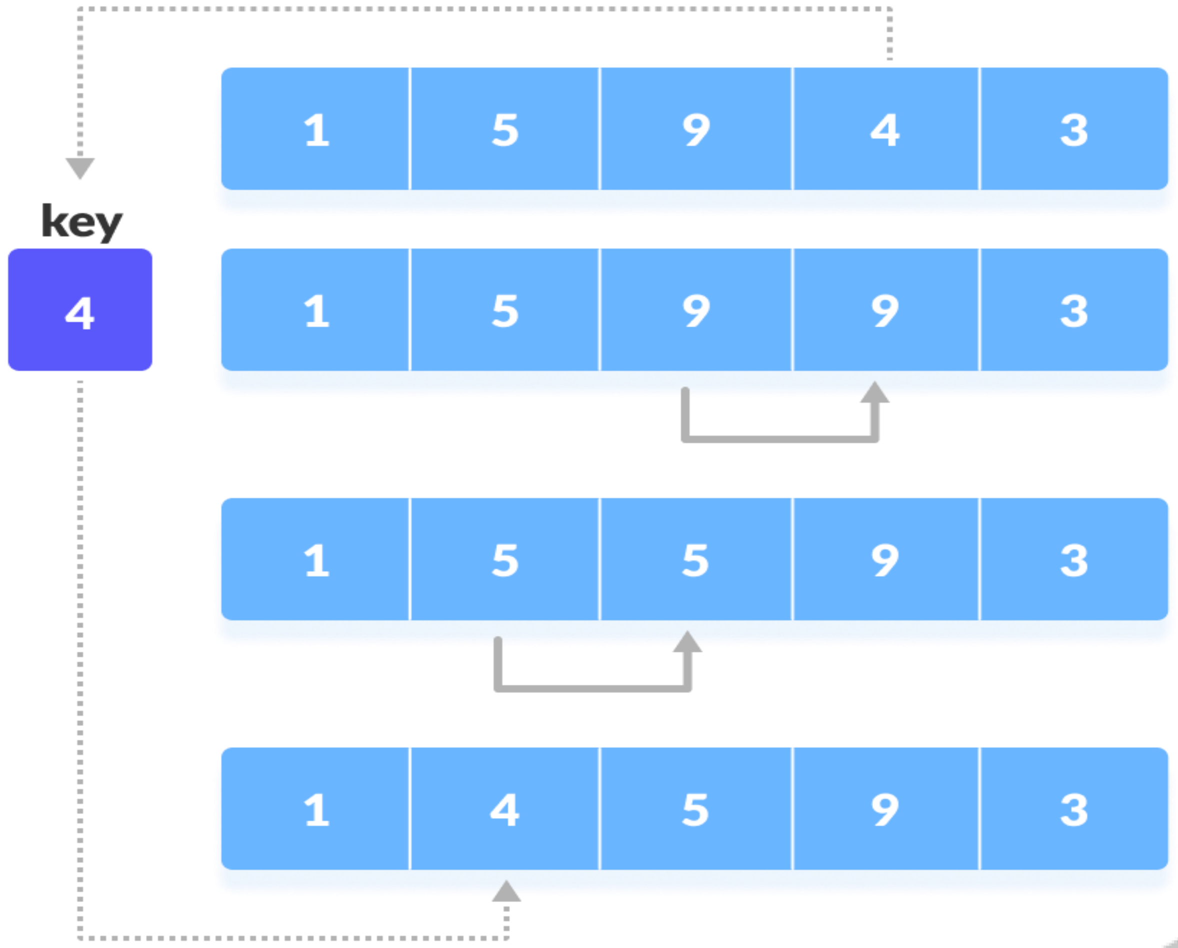
Insertion Sort

```
Procedure insertion(T[1…n])
for i ← 1 to n do
      curr ← T[i]
      j ← i-1
      while j >= 0 and T[j] > curr do
            T[j+1] ← T[j]
            j ← j-1
      end while
      T[j+1] ← curr;
end for
```



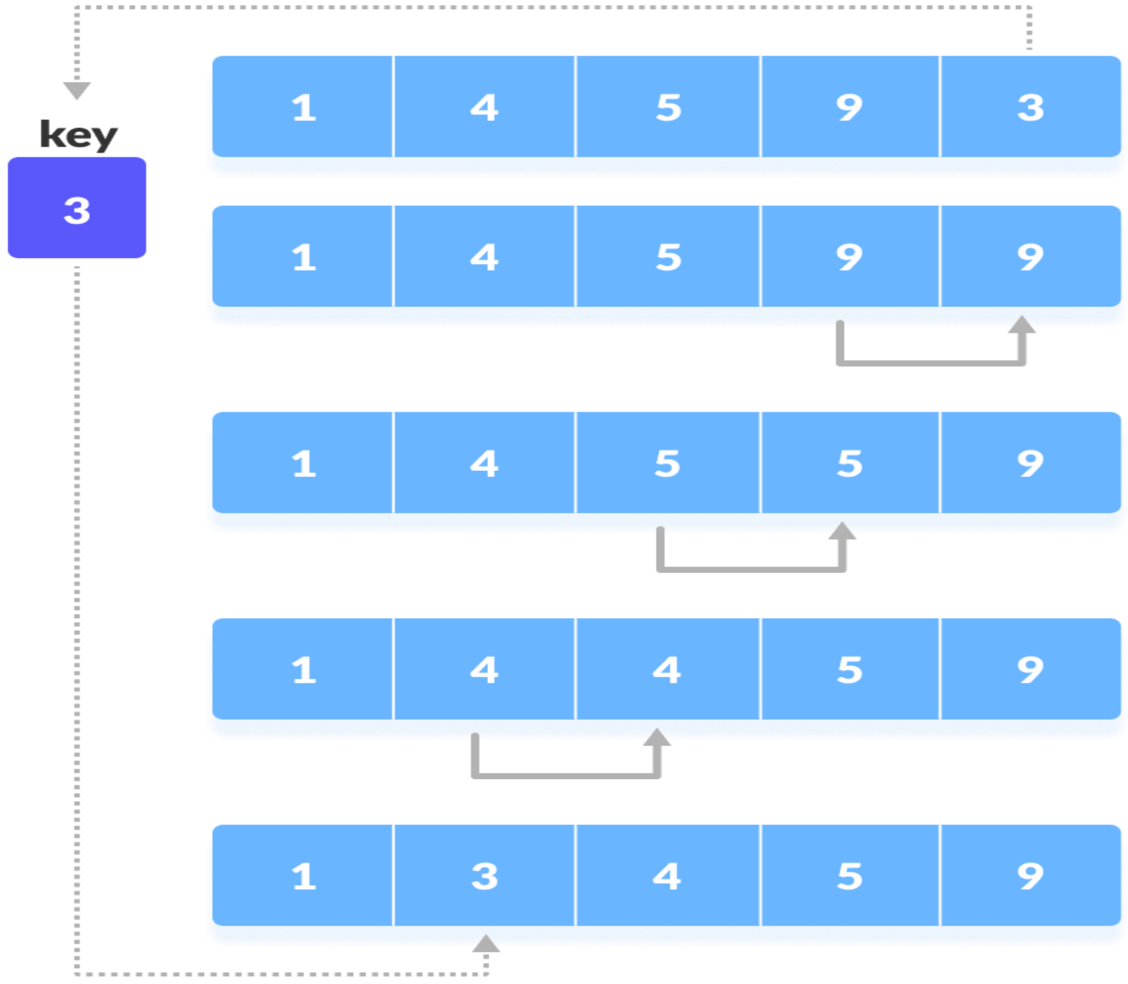
Insertion Sort

```
Procedure insertion(T[1…n])
for i ← 1 to n do
        curr ← T[i]
        j ← i-1
        while j >= 0 and T[j] > curr do
                T[j+1] ← T[j]
                j ← j-1
        end while
        T[j+1] ← curr;
end for
```



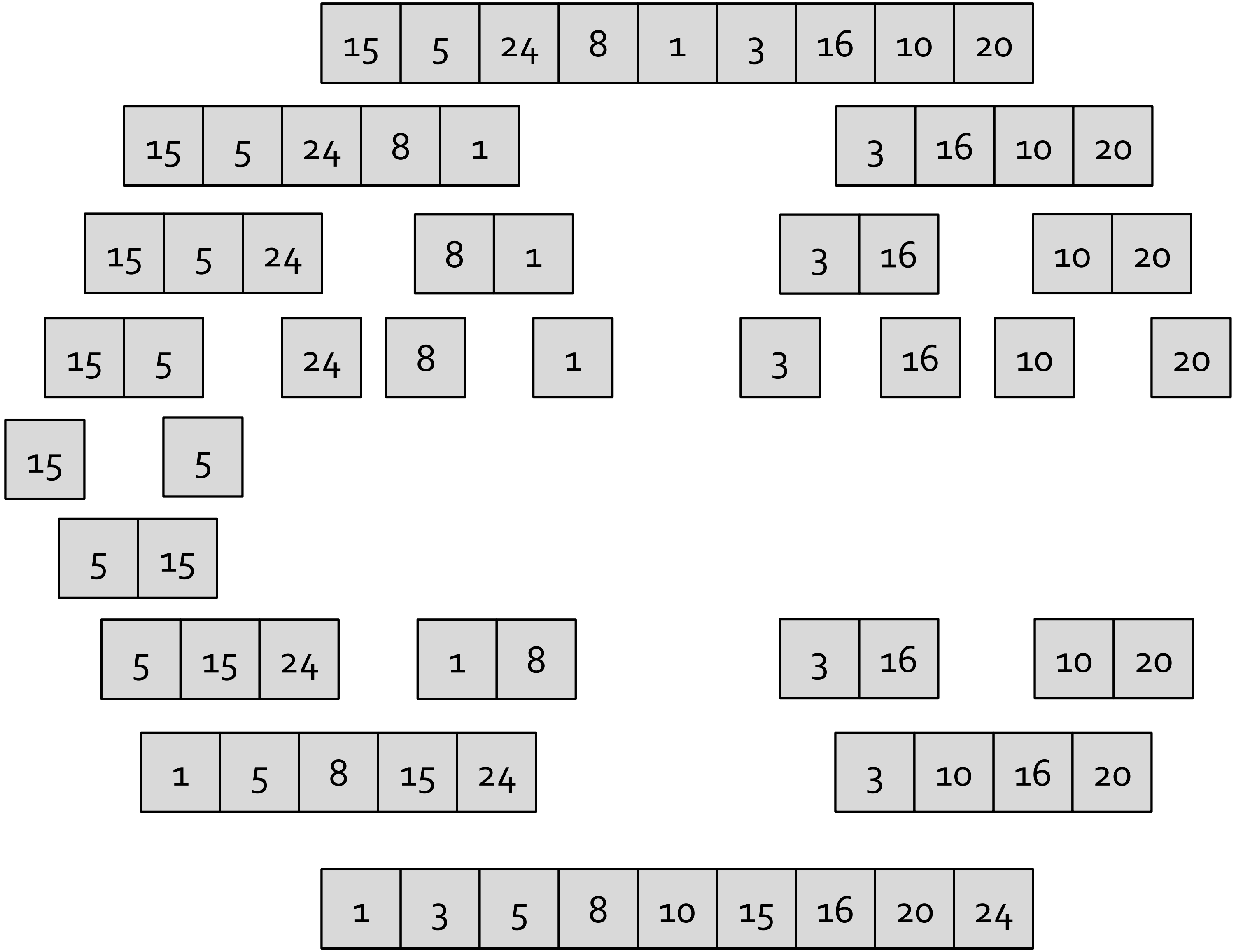
Insertion Sort

Procedure insertion(T[1…n])
for i ← 1 **to** n **do**
    curr ← T[i]
    j ← i-1
    while j >= 0 **and** T[j] > curr **do**
        T[j+1] ← T[j]
        j ← j-1
    end while
    T[j+1] ← curr;
end for



Merge Sort

Merge Sort


15 5 24 8 1 3 16 10 20
15 5 24 8 1
3 16 10 20
15 5 24
8 1
3 16
10 20
15 5
24
8
1
3
16
10
20
15
5
5 15
5 15 24
1 8
3 16
10 20
1 5 8 15 24
3 10 16 20
1 3 5 8 10 15 16 20 24

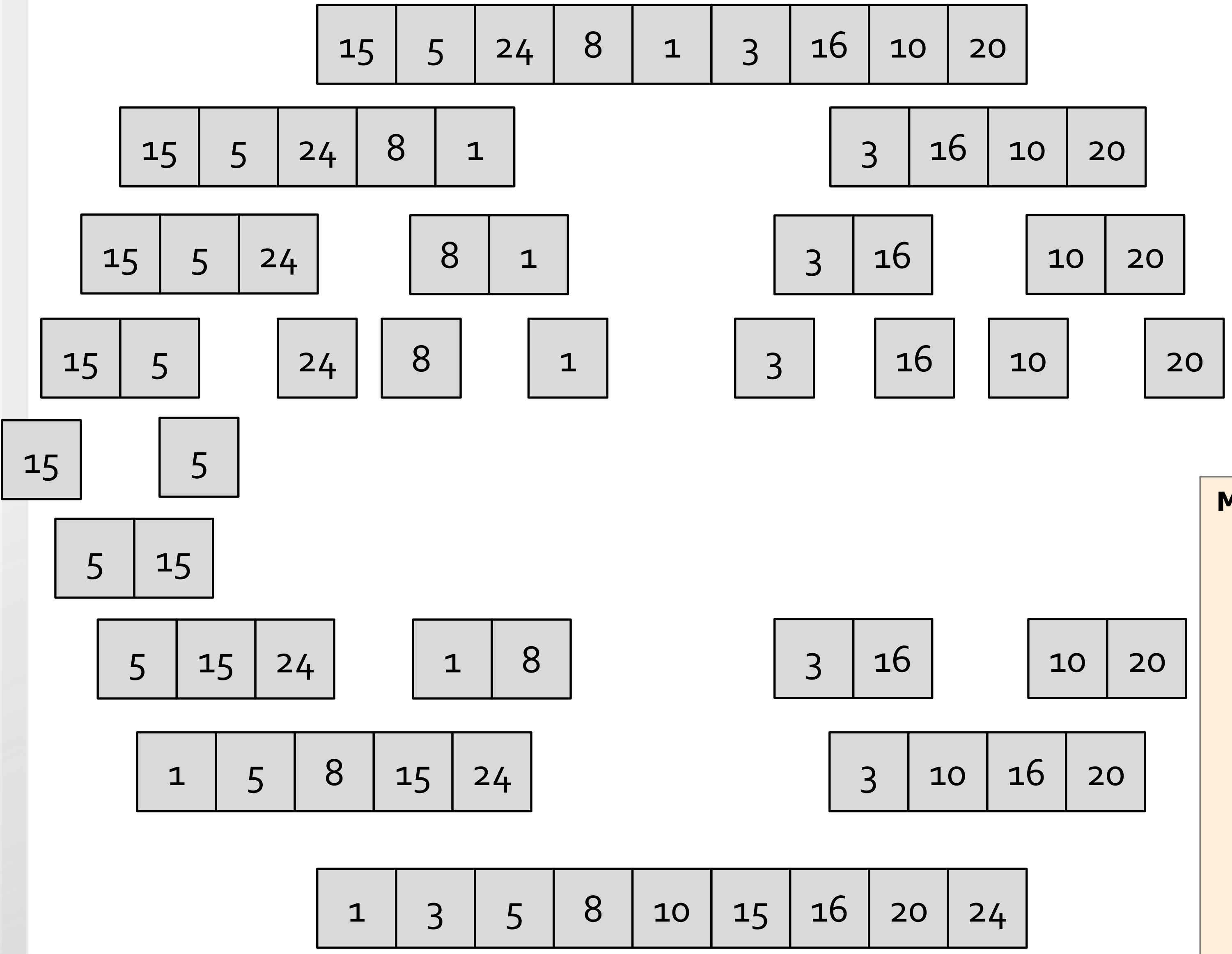
Merge Sort

```
MergeSort (A, LB, UB){
    If(LB < UB){
        mid = (LB+UB)/2;
        MergeSort (A, LB, mid);
        MergeSort (A, mid+1, UB);
        Merge (A, LB, mid, UB);
     }
}
```

```
Merge(A, LB, mid, UB){
    i = LB;
    j = mid+1;
    k = LB;
    while(i <= mid && j <= UB){
        if(a[i] <= a[j]){
            b[k]=a[i];
            i++;
        }
        else{
            b[k]=a[j];
            j++;
        }
        k++;
    }
```

```
    while(i <= mid){
            b[k]=a[i];
            i++;
            k++;
    }
    while(j <= UB){
            b[k]=a[j];
            j++;
            k++;
    }
    for(k=LB; k<=UB; k++){
        a[k]=b[k];
    }
```

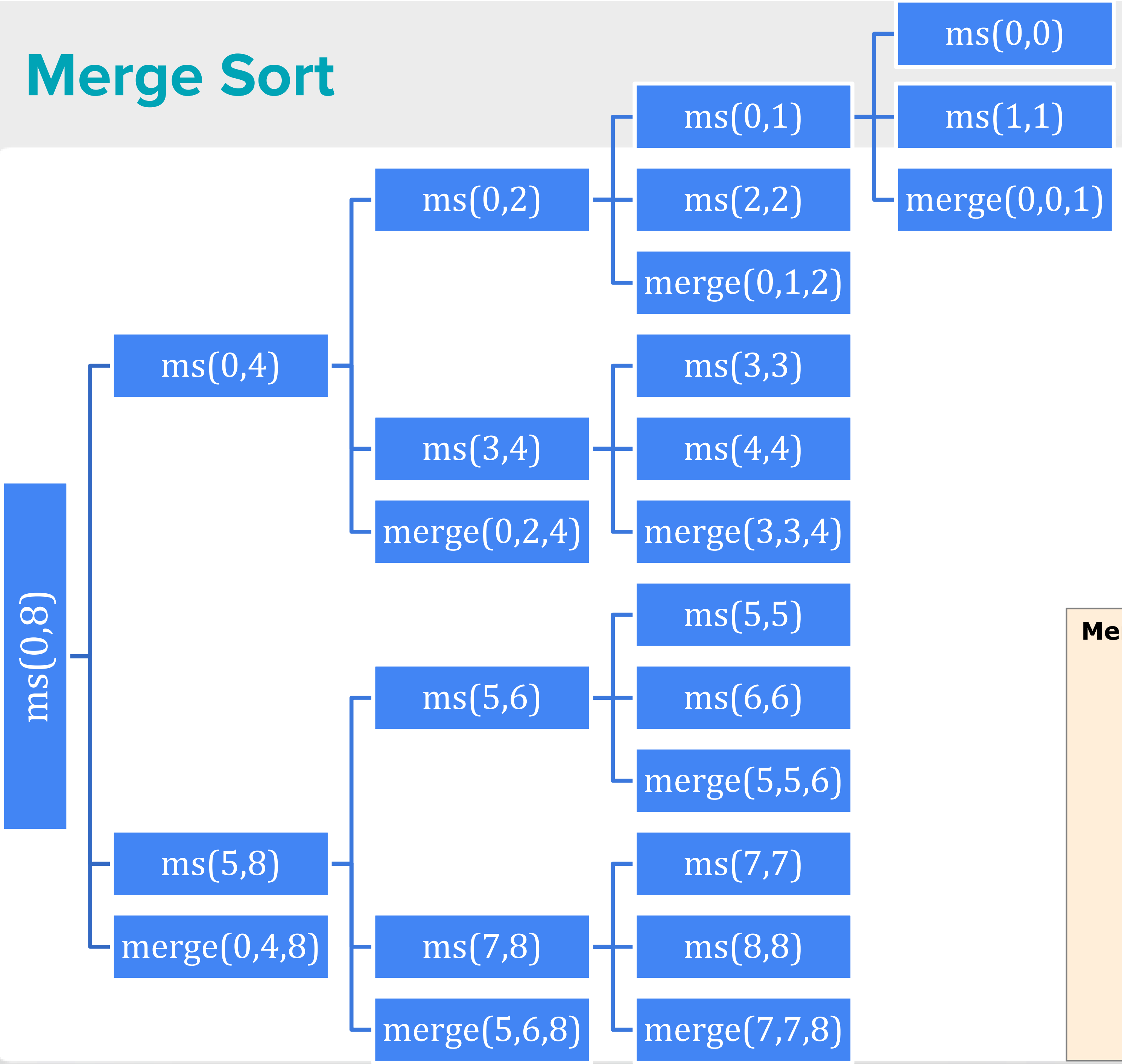
Merge Sort




```
MergeSort (A, LB, UB){
    If(LB < UB){
        mid = (LB+UB)/2;
        MergeSort (A, LB, mid);
        MergeSort (A, mid+1, UB);
        Merge (A, LB, mid, UB);
    }
}
```

```
Merge(A, LB, mid, UB){
    i = LB;
    j = mid+1;
    k = LB;
    while(i <= mid && j <= UB){
        if(a[i] <= a[j]){
            b[k]=a[i];
            i++;
        }
        else{
            b[k]=a[j];
            j++;
        }
        k++;
    }
    while(i <= mid){
            b[k]=a[i];
            i++;
            k++;
    while(j <= UB){
            b[k]=a[j];
            j++;
            k++;
    }
    for(k=LB; k<=UB; k++){
        a[k]=b[k];
    }
```

Merge Sort

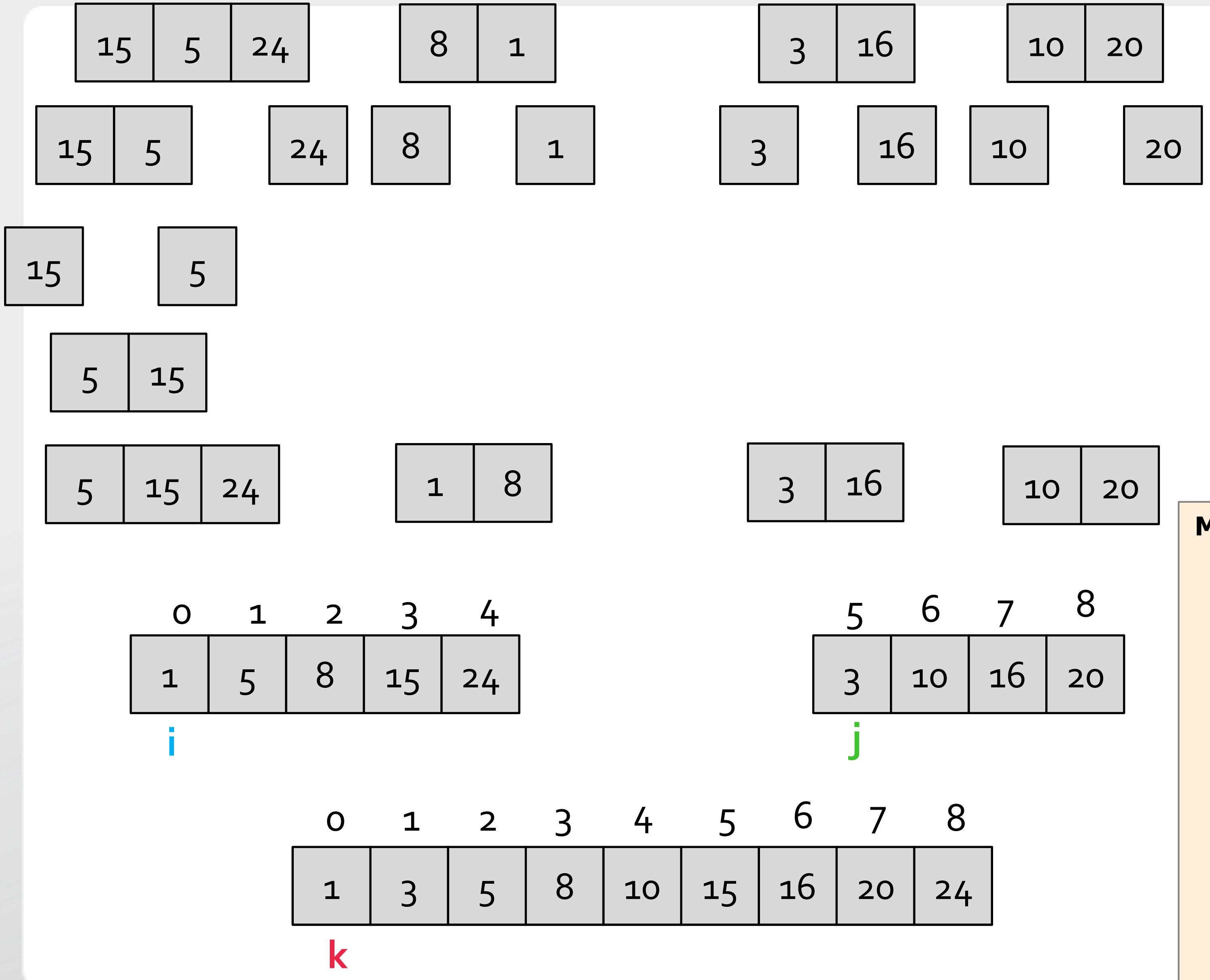



```
MergeSort (A, LB, UB){
    If(LB < UB){
        mid = (LB+UB)/2;
        MergeSort (A, LB, mid);
        MergeSort (A, mid+1, UB);
        Merge (A, LB, mid, UB);
    }
}
```

```
Merge(A, LB, mid, UB){
    i = LB;
    j = mid+1;
    k = LB;
    while(i <= mid && j <= UB){
        if(a[i] <= a[j]){
            b[k]=a[i];
            i++;
        }
        else{
            b[k]=a[j];
            j++;
        }
        k++;
    }
```

```
while(i <= mid){
        b[k]=a[i];
        i++;
        k++;
while(j <= UB){
        b[k]=a[j];
        j++;
        k++;
}
for(k=LB; k<=UB; k++){
    a[k]=b[k];
}
```

Merge Sort




```
MergeSort (A, LB, UB){
    If(LB < UB){
        mid = (LB+UB)/2;
        MergeSort (A, LB, mid);
        MergeSort (A, mid+1, UB);
        Merge (A, LB, mid, UB);
    }
}
```

```
Merge(A, LB, mid, UB){
    i = LB;
    j = mid+1;
    k = LB;
    while(i <= mid && j <= UB){
        if(a[i] <= a[j]){
            b[k]=a[i];
            i++;
        }
        else{
            b[k]=a[j];
            j++;
        }
        k++;
    }
    while(i <= mid){
            b[k]=a[i];
            i++;
            k++;
    while(j <= UB){
            b[k]=a[j];
            j++;
            k++;
    }
    for(k=LB; k<=UB; k++){
        a[k]=b[k];
    }
```

Quick Sort

Quick Sort

```
QuickSort (A, low, high){
    if(low < high){
        loc = Partition(A, low, high);
        QuickSort (A, low, loc-1);
        QuickSort (A, loc+1, high);
    }
}
```

```
Partition (A, LB, UB){
    pivot = low
    i = low;
    j = high;
    while(i < j){
        while(A[i] <= A[pivot] && i<high){
            i++;
        }
        while(A[j] > A[pivot] && j>low){
            j--;
        }
        if(i < j){
            swap(A[i], A[j]);
        }
    }
    swap(A[pivot], A[j]);
    return j;
}
```

Quick Sort

low								high
0	1	2	3	4	5	6	7	8
7	6	10	5	9	2	1	15	7
i	i	i						j

pivot=7

0	1	2	3	4	5	6	7	8
7	6	7	5	9	2	1	15	10
		i	i	i		j	j	j

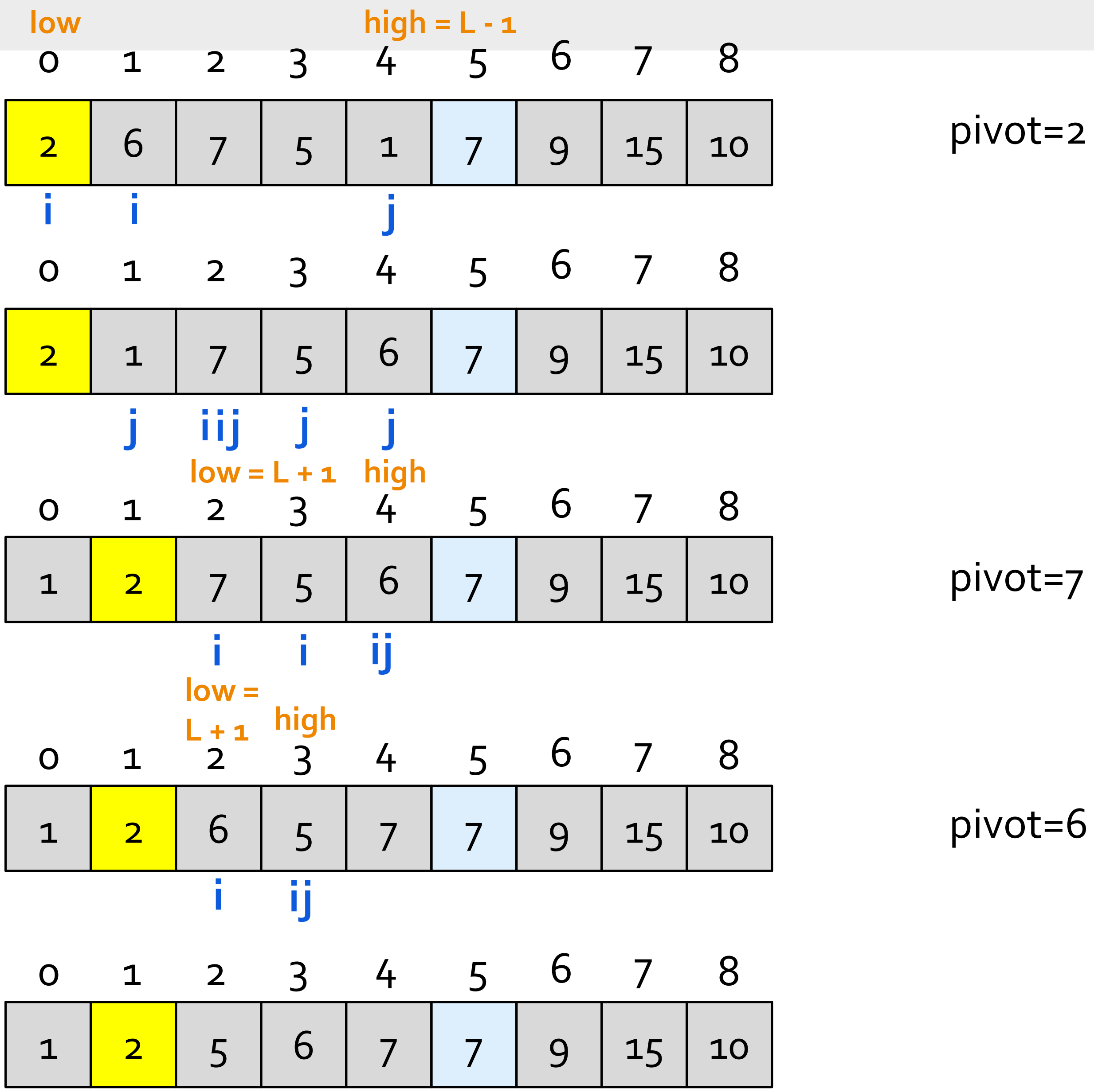
0	1	2	3	4	5	6	7	8
7	6	7	5	1	2	9	15	10
				i	j	iij		

0	1	2	3	4	5	6	7	8
2	6	7	5	1	7	9	15	10
					j	i		

```
QuickSort (A, low, high){
    if(low < high){
        loc = Partition(A, low, high);
        QuickSort (A, low, loc-1);
        QuickSort (A, loc+1, high);
    }
}
```

```
Partition (A, LB, UB){
    pivot = low
    i = low;
    j = high;
    while(i < j){
        while(A[i] <= A[pivot] && i<high){
            i++;
        }
        while(A[j] > A[pivot] && j>low){
            j--;
        }
        if(i < j){
            swap(A[i], A[j]);
        }
    }
    swap(A[pivot], A[j]);
    return j;
}
```

Quick Sort




```
QuickSort (A, low, high){
    if(low < high){
        loc = Partition(A, low, high);
        QuickSort (A, low, loc-1);
        QuickSort (A, loc+1, high);
    }
}
```

```
Partition (A, LB, UB){
    pivot = low
    i = low;
    j = high;
    while(i < j){
        while(A[i] <= A[pivot] && i<high){
            i++;
        }
        while(A[j] > A[pivot] && j>low){
            j--;
        }
        if(i < j){
            swap(A[i], A[j]);
        }
    }
    swap(A[pivot], A[j]);
    return j;
}
```

Quick Sort

low = L + 1 high

0	1	2	3	4	5	6	7	8
1	2	5	6	7	7	9	15	10
						j	i j	j

pivot=9

low = L + 1 high

0	1	2	3	4	5	6	7	8
1	2	5	6	7	7	9	15	10
							i	ij

pivot=15

0	1	2	3	4	5	6	7	8
1	2	5	6	7	7	9	10	15

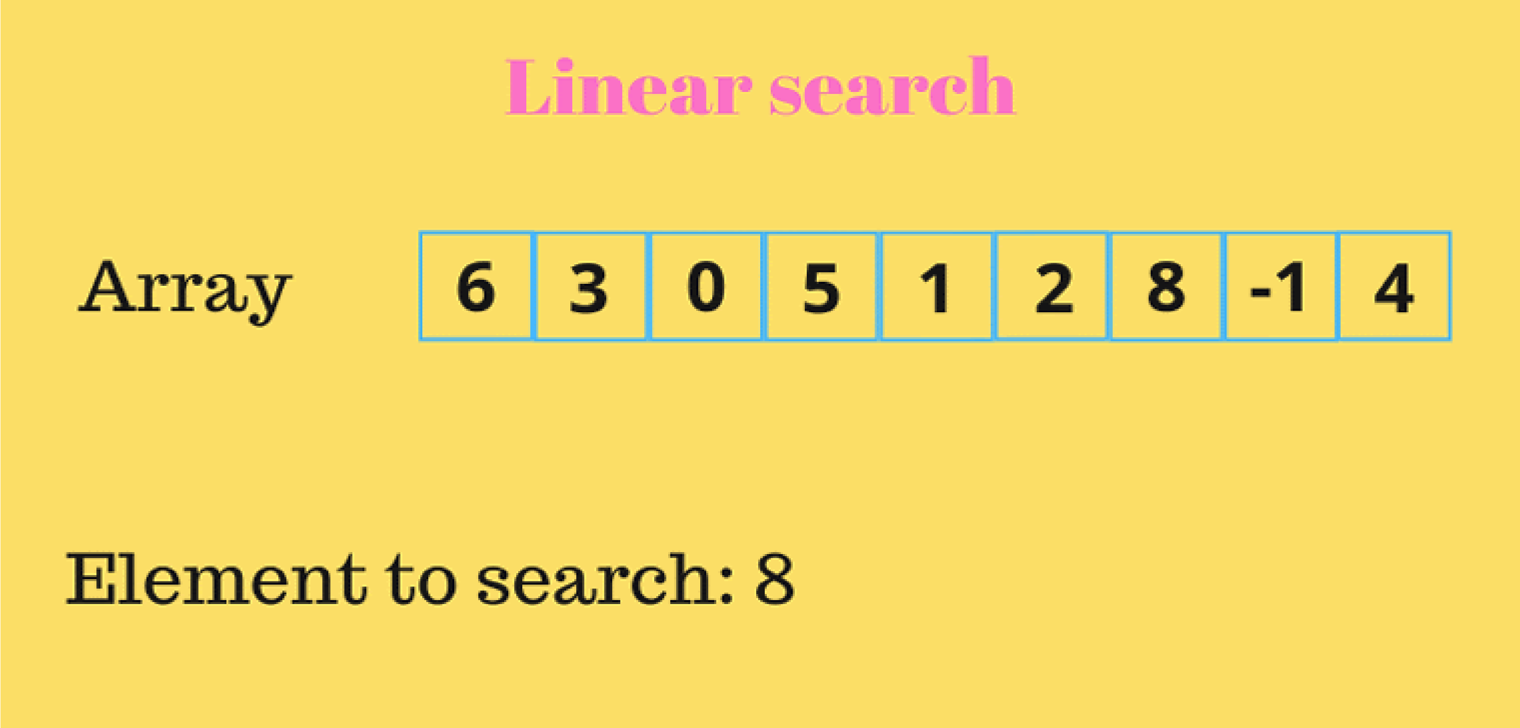
```
QuickSort (A, low, high){
    if(low < high){
        loc = Partition(A, low, high);
        QuickSort (A, low, loc-1);
        QuickSort (A, loc+1, high);
    }
}
```

```
Partition (A, LB, UB){
    pivot = low
    i = low;
    j = high;
    while(i < j){
        while(A[i] <= A[pivot] && i<high){
            i++;
        }
        while(A[j] > A[pivot] && j>low){
            j--;
        }
        if(i < j){
            swap(A[i], A[j]);
        }
    }
    swap(A[pivot], A[j]);
    return j;
}
```

Linear Search

Linear Search

- Linear search in C to find whether a number is present in an array. If it's present, then at what location it occurs. It is also known as a **sequential search**.
- we **compare** each element with the element **to search** until we find it or the list ends.



Linear Search


```
for(i=0; i<n; i++)
{
    if(a[i] == data)
    {
        Printf("element found at location: %d", i);
        Break;
    }
}
If(i >= n){
    printf("element not found");
}
```

Binary Search

Binary Search

- Binary search will take less time than linear search.
- **Precondition:** Array must be sorted. If array is not sorted we cannot apply algorithm.

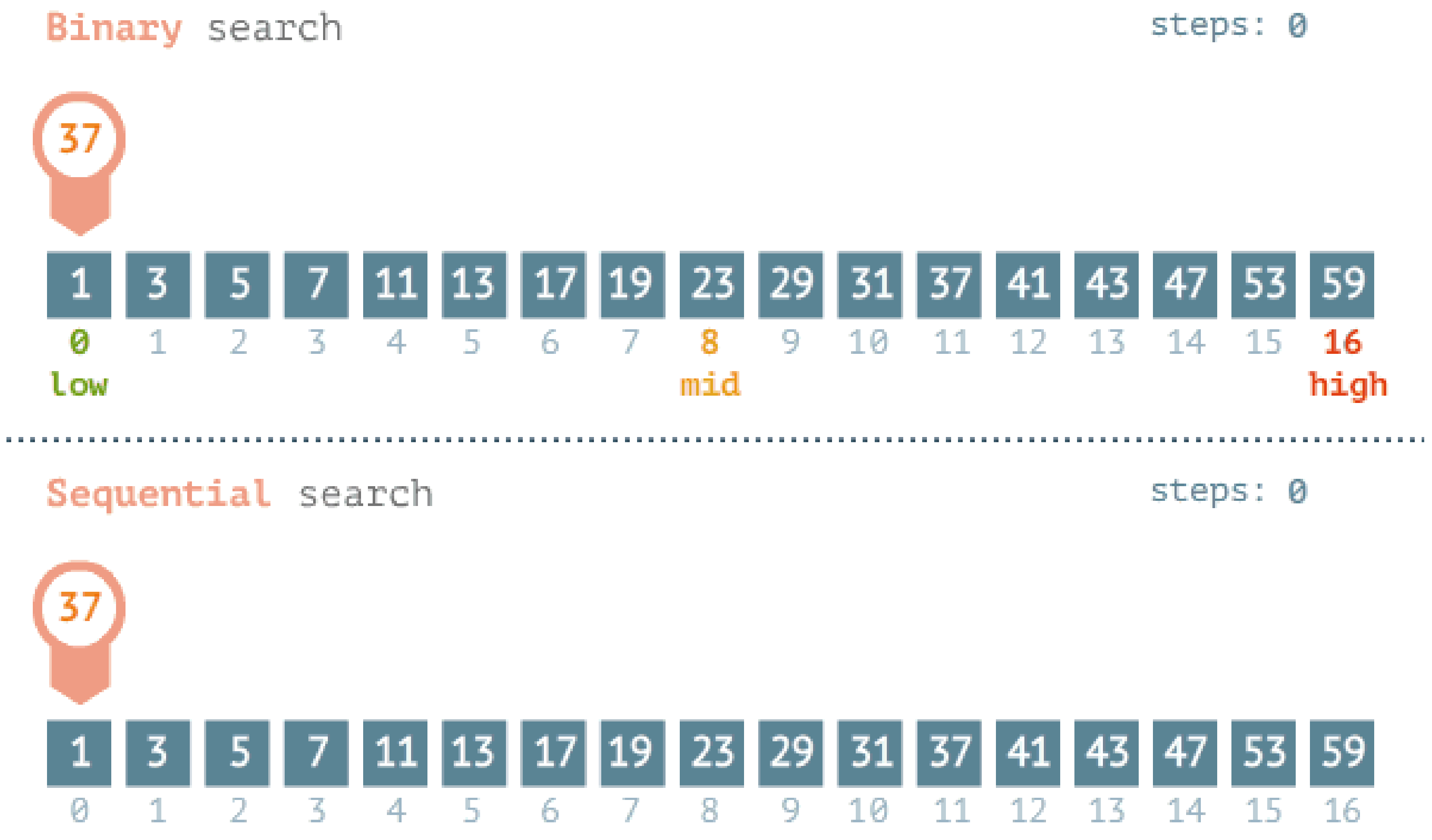
- **Working Principle:**
- Search a sorted array by repeatedly dividing the search interval in half.
- We basically ignore half of the elements just after one comparison.

1. Compare **data** with the **middle** element.
2. If **data** matches with **middle** element, we return the **mid** index.
3. Else If **data > mid** element, then **data** can only lie in **right** half subarray after the **mid** element. So we trace for **right** half.
4. Else (**data** is smaller) trace for the **left half.**

Binary Search

```
BINARY-SEARCH(A, low, high, data){
        if(low <= high){
                mid = floor((start + end)/2)
        if (A[mid]== data){
                return mid
        }
        if (A[mid]>data){
                return BINARY-SEARCH(A, low, mid-1, data)
        }
        if( A[mid]<data){
                return BINARY-SEARCH(A, mid+1, high, data)
        }
        return FALSE // in case, element is not in the array
}
```

Binary Search vs Linear Search



www.penjee.com


Thank
You