MERGE SORT

Merge Sort

- Merge Sort applies the Divide and Conquer algorithm.
- It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.
- Has a runtime value of **O(nLogn)**

Divide and Conquer Algorithm

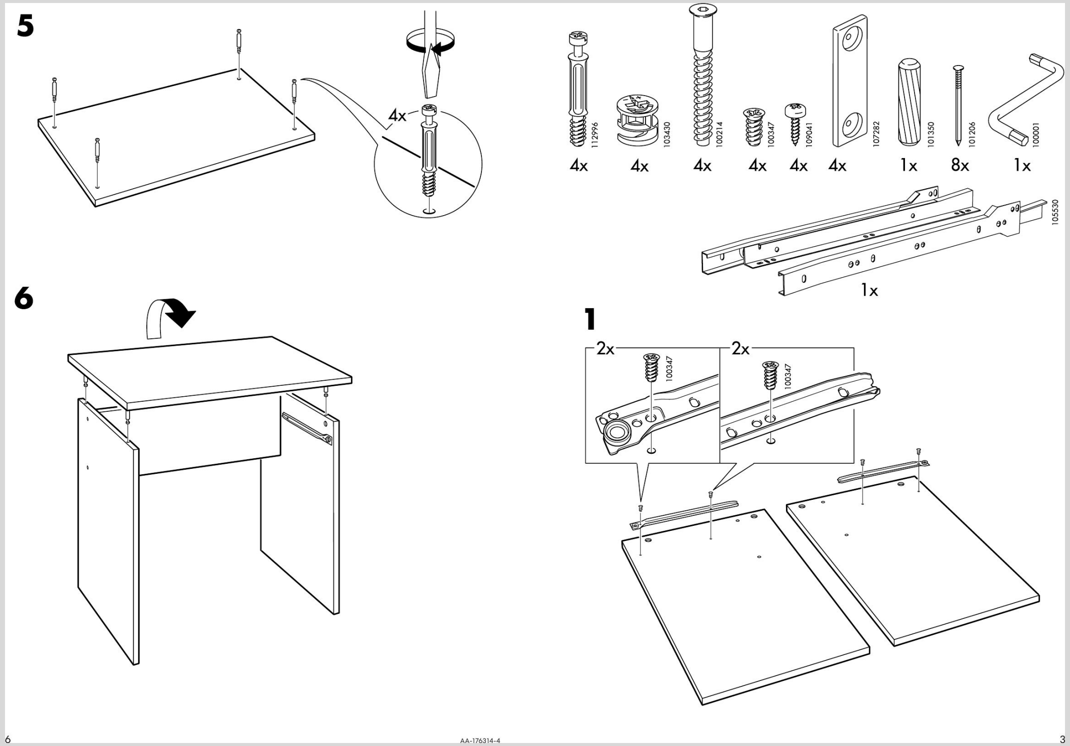
- **Divide**: This involves dividing the main problem into a series of smaller sub-problems via recursion
- **Conquer**: The sub-problems are then further divided via recursion until all sub-problems are in simple forms that allows it to be solved directly
- **Combine**: The solutions to each sub-problem are then combined and then returned in each recursion to find the solution to the main problem.

Merge Sort

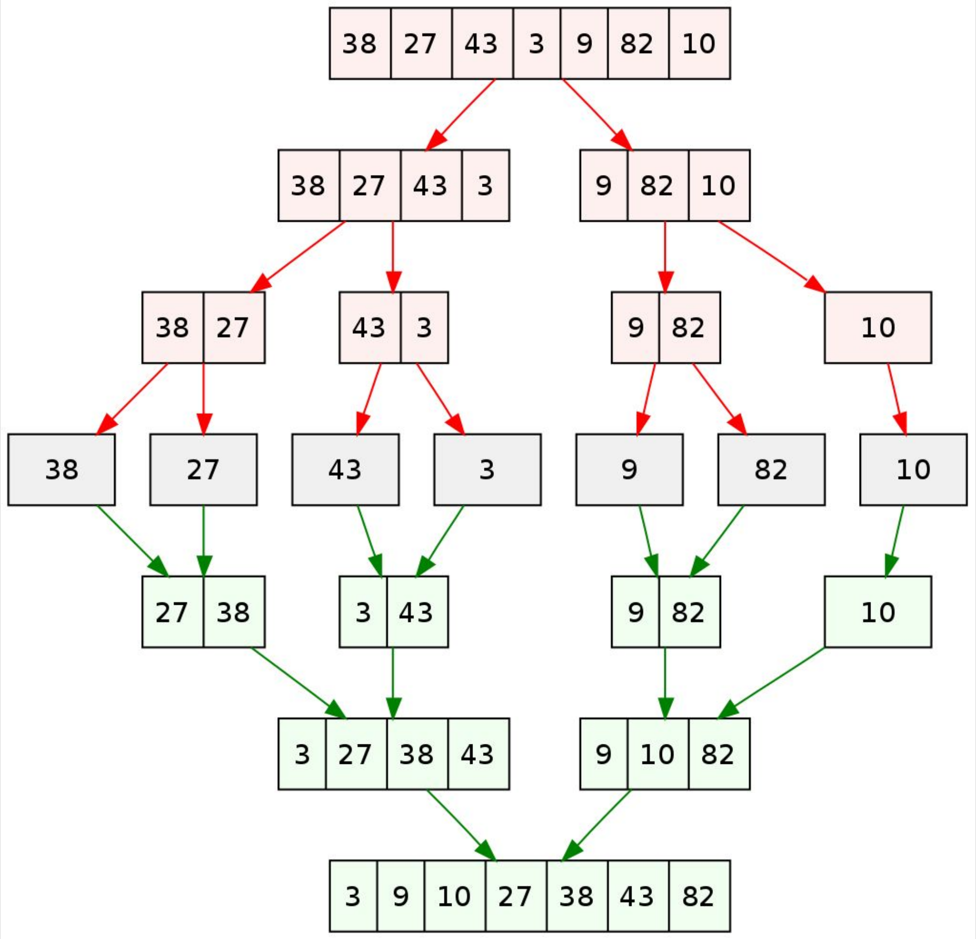
Divide and Conquer Algorithm

Almost like **assembling a piece of furniture**

- **Divide**: Main problem already divided (unassembled desk) into series of sub-problems (desk legs, tabletop)
- **Conquer**: Directly solve each sub-problem (attach screws and pins to tabletop and the desk legs)
- **Combine**: Combine each sub-problem solution in order to solve the main problem (connect the tabletop to the legs)



Illustration



Functions

- **mergeSort(int arr[], int l, int r)**
  - Divides the input array into halves

- **merge(int arr[], int l, int m, int r)**
  - Merges the two sorted halves

mergeSort function

```
void mergeSort(int arr[], int l, int r) {
   if (l < r) {
          int m = (l + r) / 2;

          mergeSort(arr, l, m);
          mergeSort(arr, m+1, r);

          merge(arr, l, m, r);
   }
}
```

mergeSort function

```
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
            int m = (l + r) / 2;

            mergeSort(arr, l, m);
            mergeSort(arr, m+1, r);

            merge(arr, l, m, r);
    }
}
```

mergeSort function

```
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}
```

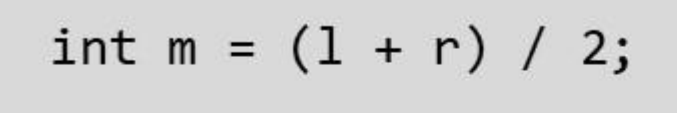
if (l < r) {

mergeSort function

```
void mergeSort(int arr[], int l, int r) {
   if (l < r) {
         int m

         mergeSort(arr, l, m);
         mergeSort(arr, m+1, r);

         merge(arr, l, m, r);
   }
}
```



```
int m = (l + r) / 2;
```

mergeSort function

```
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
            int m = (l + r) / 2;

            merge
            merge

            merge(arr, l, m, r);
    }
}
```

```
mergeSort(arr, l, m);
mergeSort(arr, m+1, r);
```

mergeSort function

```
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}
```

mergeSort function

```
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
          int m = (l + r) / 2;

          mergeSort(arr, l, m);
          mergeSort(arr, m+1, r);

          merge(arr, l, m, r);
    }
}
```

mergeSort function

```
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
          int m = (l + r) / 2;

          mergeSort(arr, l, m);
          mergeSort(arr, m+1, r);

          merge(arr, l, m, r);
    }
}
```

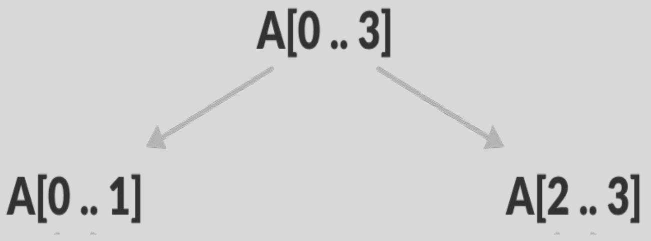
A[0 .. 3]

mergeSort function

```
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}
```

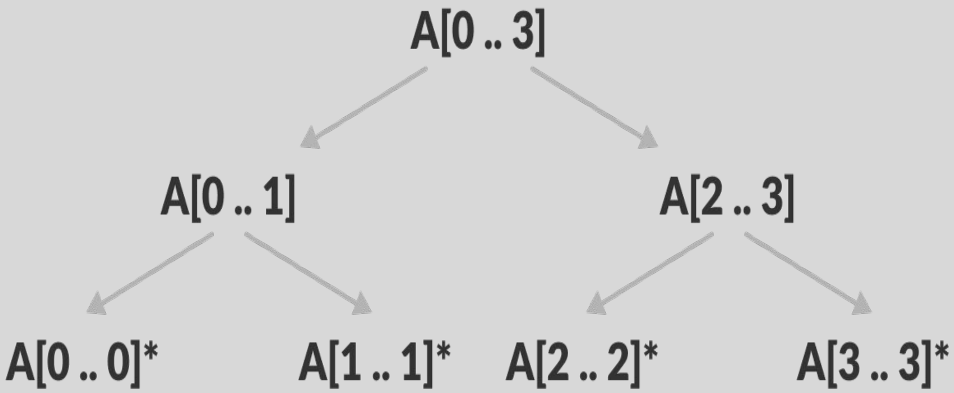


mergeSort function

```
void mergeSort(int arr[], int l, int r) {
   if (l < r) {
         int m = (l + r) / 2;

         mergeSort(arr, l, m);
         mergeSort(arr, m+1, r);

         merge(arr, l, m, r);
   }
}
```



A[0 .. 3]

A[0 .. 1]

A[2 .. 3]

A[0 .. 0]*

A[1 .. 1]*

A[2 .. 2]*

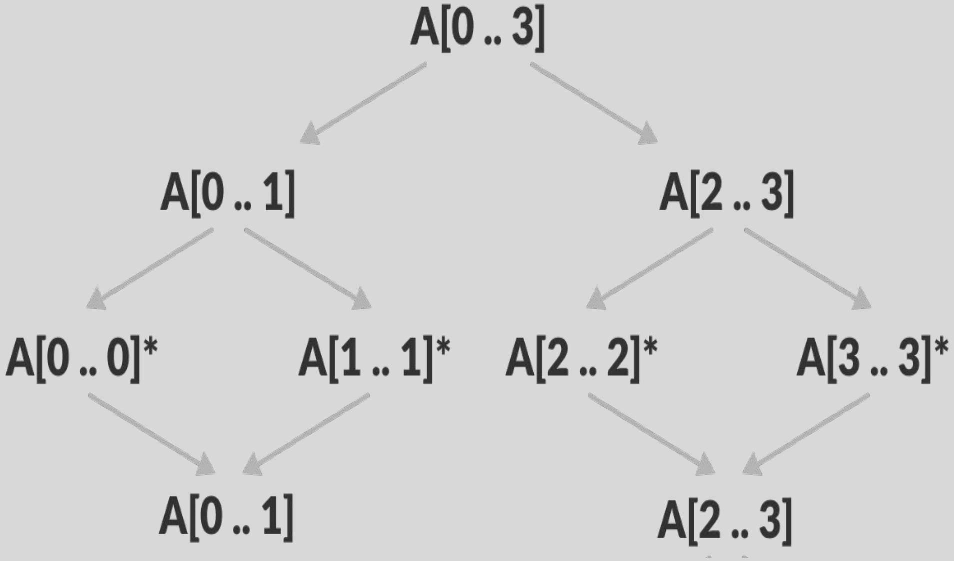
A[3 .. 3]*

mergeSort function

```
void mergeSort(int arr[], int l, int r) {
   if (l < r) {
         int m = (l + r) / 2;

         mergeSort(arr, l, m);
         mergeSort(arr, m+1, r);

         merge(arr, l, m, r);
   }
}
```

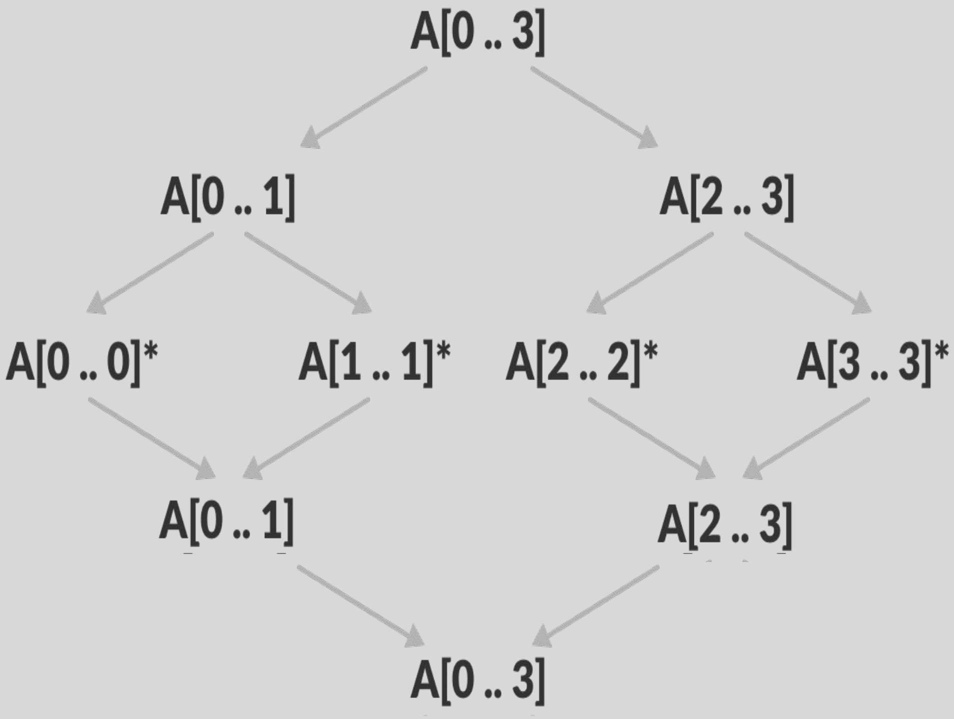


mergeSort function

```
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
            int m = (l + r) / 2;

            mergeSort(arr, l, m);
            mergeSort(arr, m+1, r);

            merge(arr, l, m, r);
    }
}
```



merge function

```
void merge(int arr[], int l, int m, int r) {
   int i, j, k, size1, size2;

   size1 = m-l+1;
   size2 = r-m;

   int L[size1], R[size2];

   for(i=0; i<size1; i++) {
      L[i] = arr[l+i];
   }
   for(i=0; i<size2; i++) {
      R[i] = arr[m+1+i];
   }
```

merge function

```
void merge(int arr[], int l, int m, int r) {
   int i, j, k, size1, size2;
   size1 = m-l+1;
   size2 = r-m;
   int L[size1], R[size2];
   for(i=0; i<size1; i++) {
      L[i] = arr[l+i];
   }
   for(i=0; i<size2; i++) {
      R[i] = arr[m+1+i];
   }
```

merge function

```
void merge(int arr[], int l, int m, int r) {
      int i, j, k, size1, size2;
   size1 = m-l+1;
   size2 = r-m;

   int L[size1], R[size2];

   for(i=0; i<size1; i++) {
      L[i] = arr[l+i];
   }
   for(i=0; i<size2; i++) {
      R[i] = arr[m+1+i];
   }
```

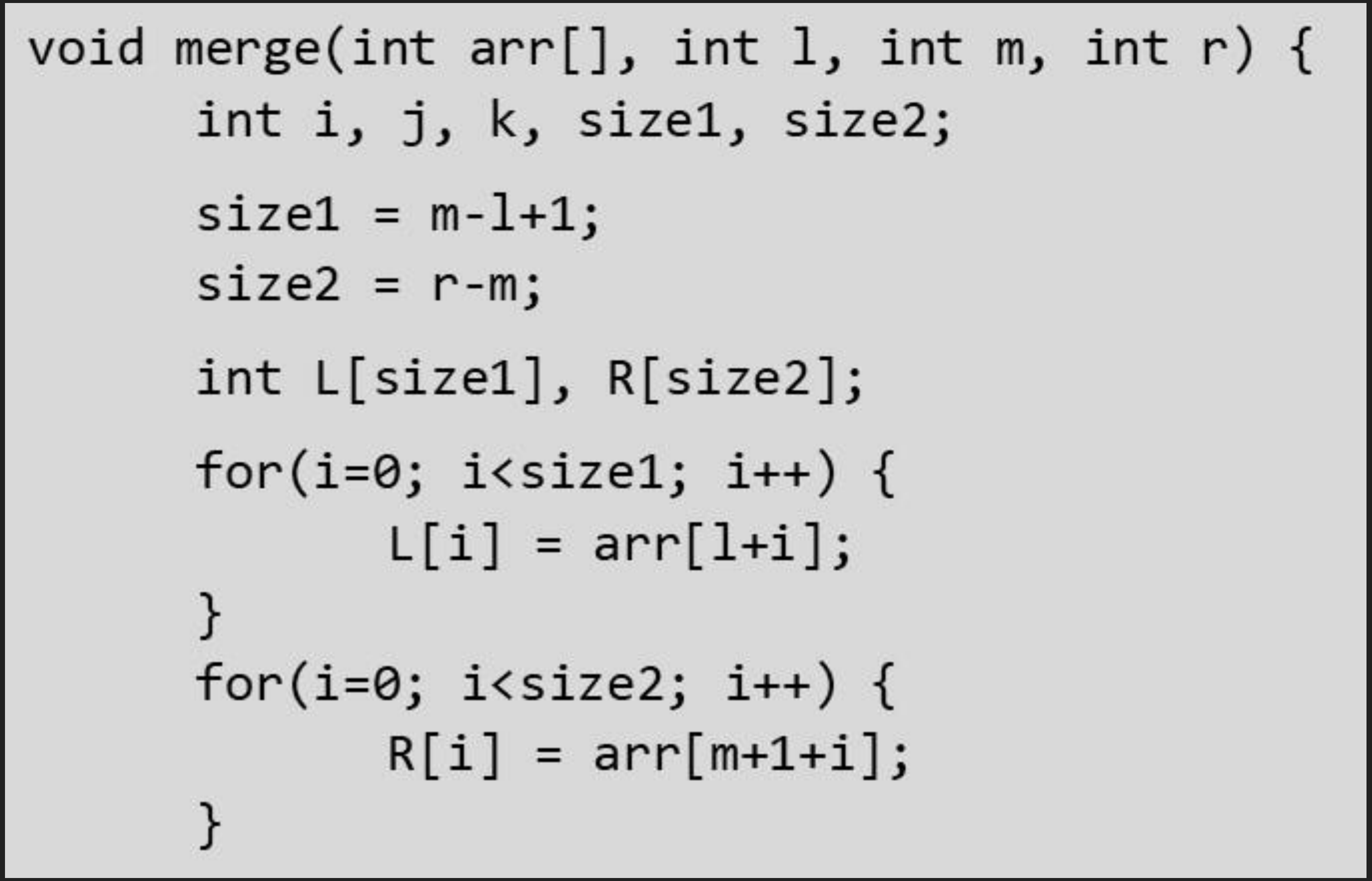
merge function

```
void merge(int arr[], int l, int m, int r) {
      int i, j, k, size1, size2;

      size1 = m-l+1;
      size2 = r-m;

      int L[size1], R[size2];
  for(i=0; i<size1; i++) {
      L[i] = arr[l+i];
  }
  for(i=0; i<size2; i++) {
      R[i] = arr[m+1+i];
  }
```

merge function



```
void merge(int arr[], int l, int m, int r) {
      int i, j, k, size1, size2;
      size1 = m-l+1;
      size2 = r-m;
      int L[size1], R[size2];
      for(i=0; i<size1; i++) {
            L[i] = arr[l+i];
      }
      for(i=0; i<size2; i++) {
            R[i] = arr[m+1+i];
      }
```

merge function

```
i = 0;
j = 0;
k = l;

while(i<size1 && j<size2) {
   if(L[i] < R[j]) {
      arr[k] = L[i];
      i++;
   } else {
      arr[k] = R[j];
      j++;
   }
   k++;
}
```

```
  while (i<size1) {
     arr[k] = L[i];
     k++;
     i++;
  }

  while(j<size2) {
     arr[k] = R[j];
     k++;
     j++;
  }
}
```

merge function

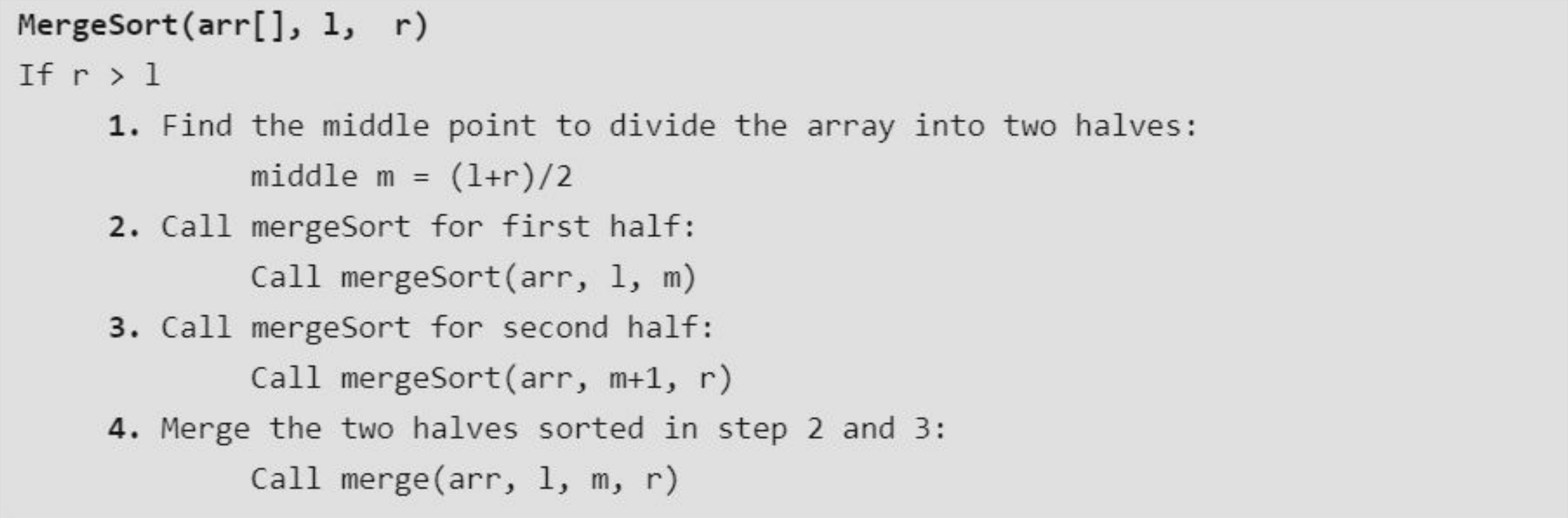
```
i = 0;
j = 0;
k = l;

while(i<size1 && j<size2) {
    if(L[i] < R[j]) {
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = R[j];
        j++;
    }
    k++;
}
```

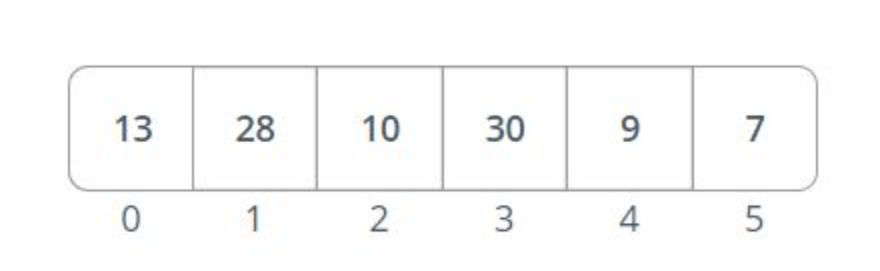
```
    while (i<size1) {
        arr[k] = L[i];
        k++;
        i++;
    }

    while(j<size2) {
        arr[k] = R[j];
        k++;
        j++;
    }
}
```

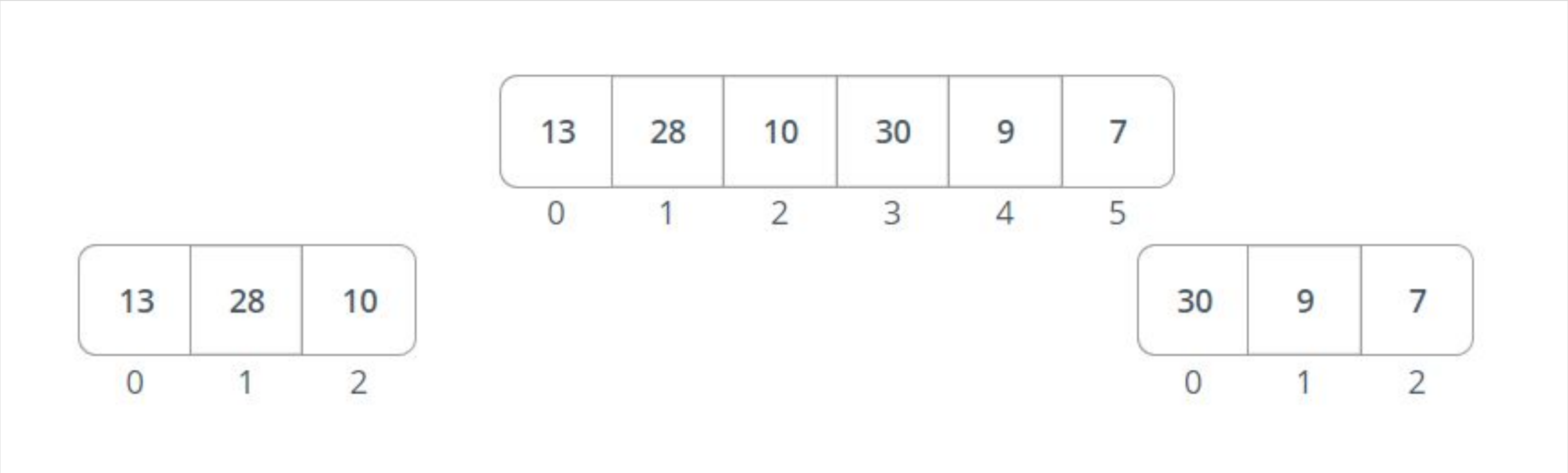
PSEUDOCODE



```
MergeSort(arr[], l,  r)
If r > l
     1. Find the middle point to divide the array into two halves:
             middle m = (l+r)/2
     2. Call mergeSort for first half:
             Call mergeSort(arr, l, m)
     3. Call mergeSort for second half:
             Call mergeSort(arr, m+1, r)
     4. Merge the two halves sorted in step 2 and 3:
             Call merge(arr, l, m, r)
```



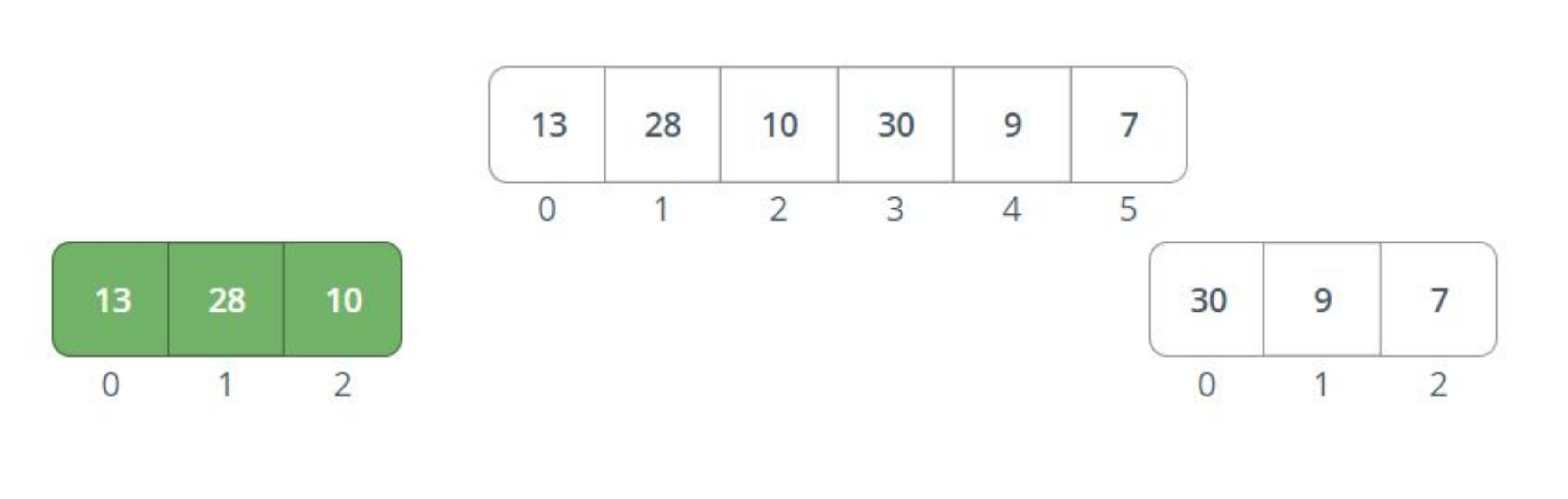
13	28	10	30	9	7
0	1	2	3	4	5



13	28	10	30	9	7
0	1	2	3	4	5

13	28	10
0	1	2

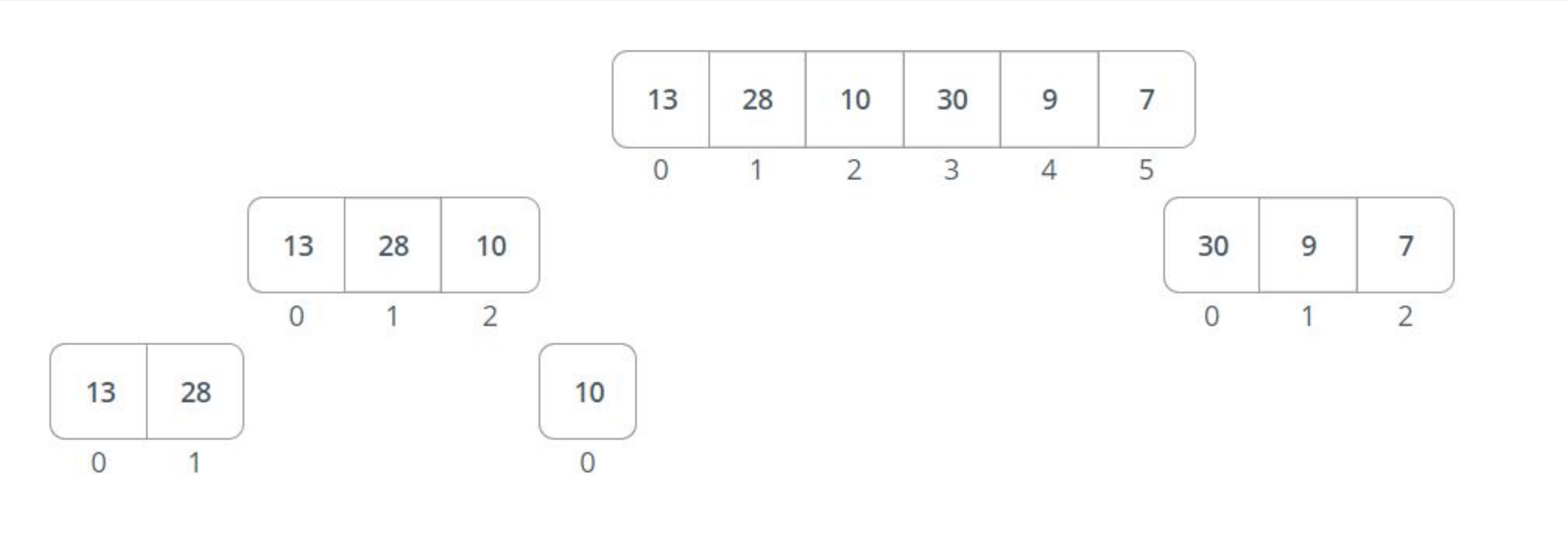
30	9	7
0	1	2



13	28	10	30	9	7
0	1	2	3	4	5

13	28	10
0	1	2

30	9	7
0	1	2



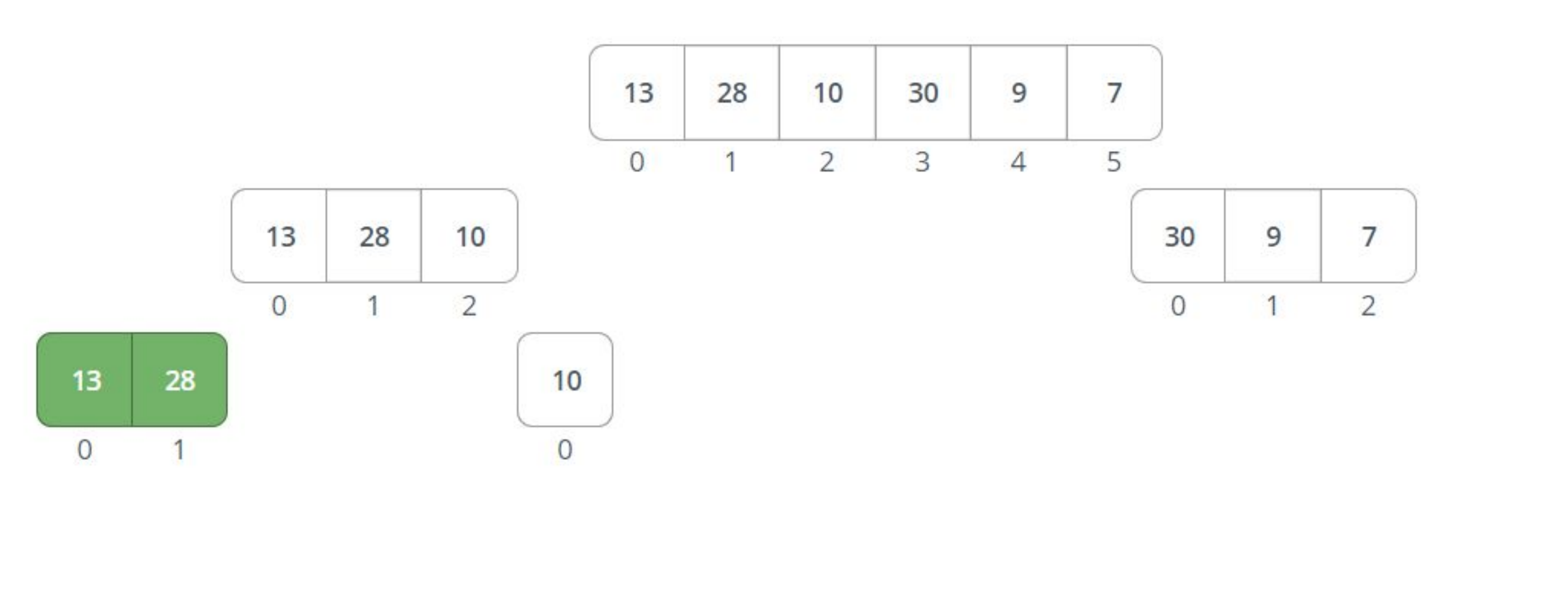
13	28	10	30	9	7
0	1	2	3	4	5

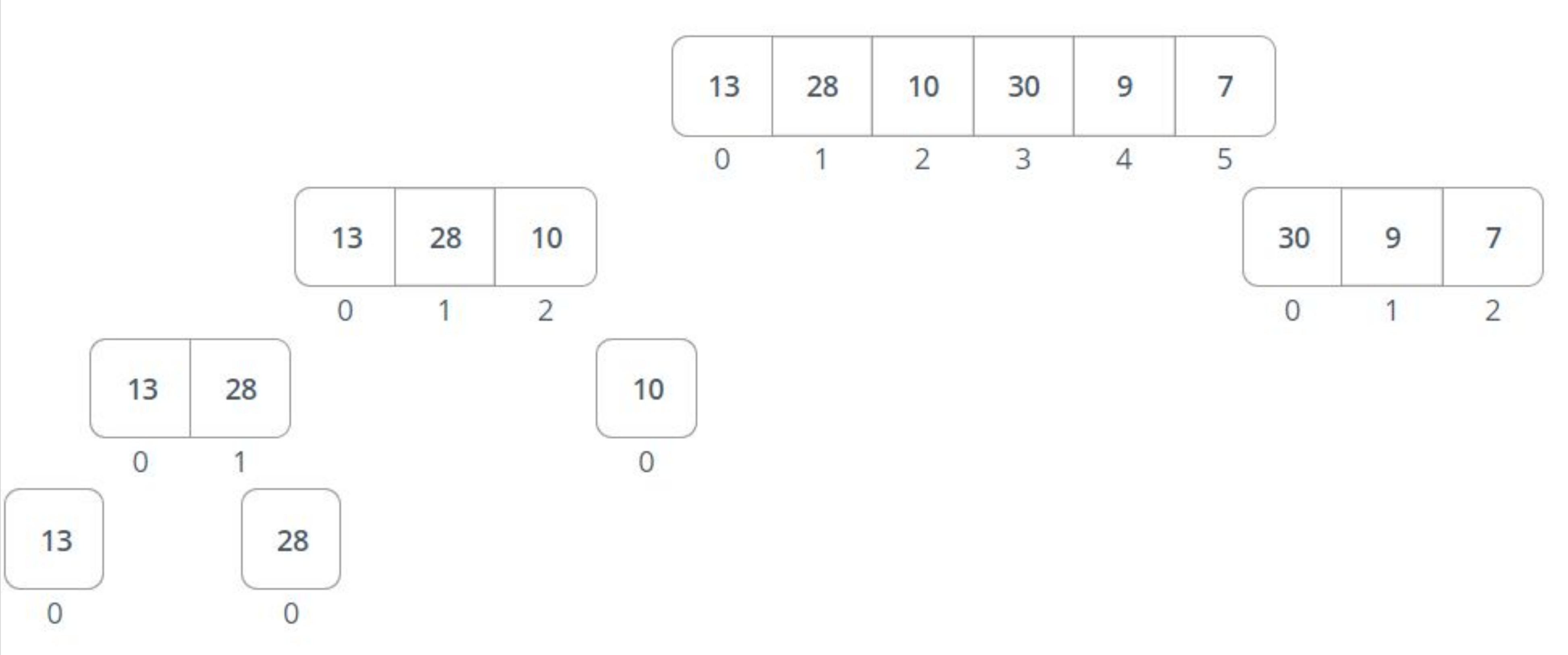
13	28	10
0	1	2

30	9	7
0	1	2

13	28
0	1

10
0


13 28 10 30 9 7
0 1 2 3 4 5
13 28 10
0 1 2
30 9 7
0 1 2
13 28
0 1
10
0



13	28	10	30	9	7
0	1	2	3	4	5

13	28	10
0	1	2

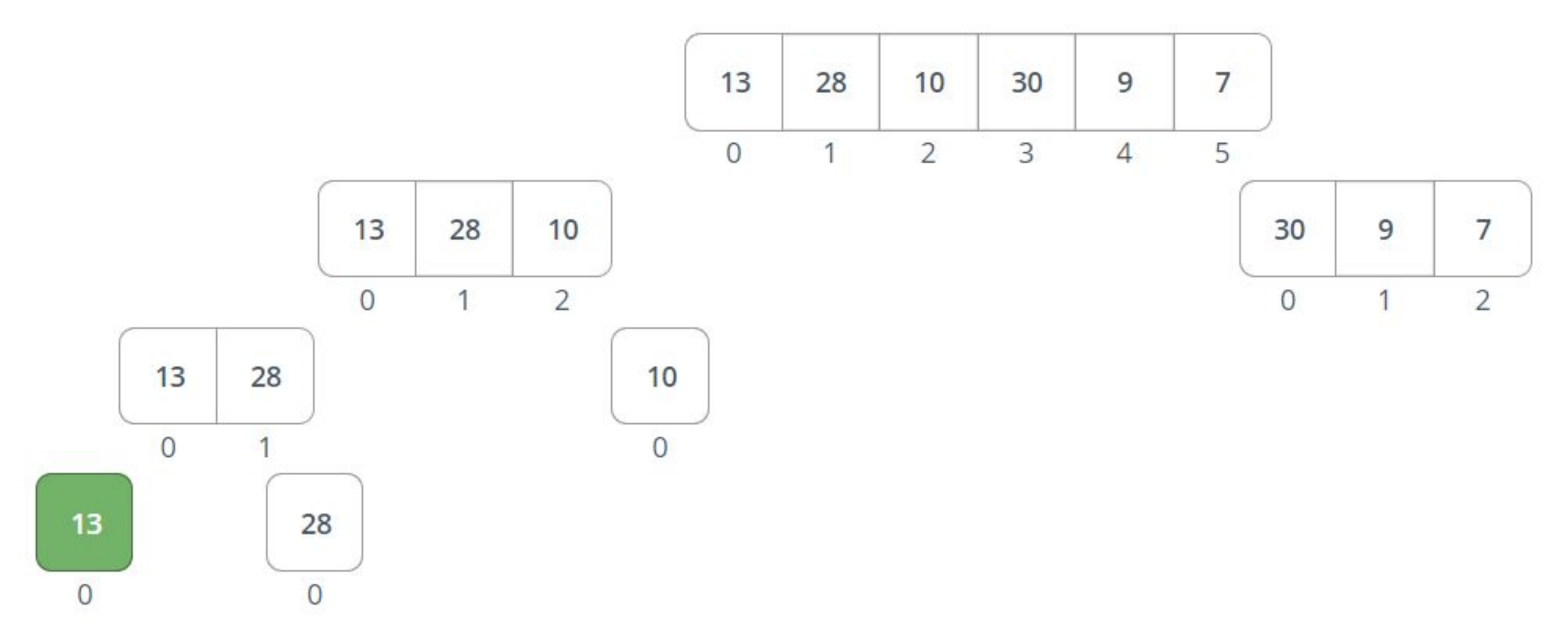
30	9	7
0	1	2

13	28
0	1

10
0

13
0

28
0



13	28	10	30	9	7
0	1	2	3	4	5

13	28	10
0	1	2

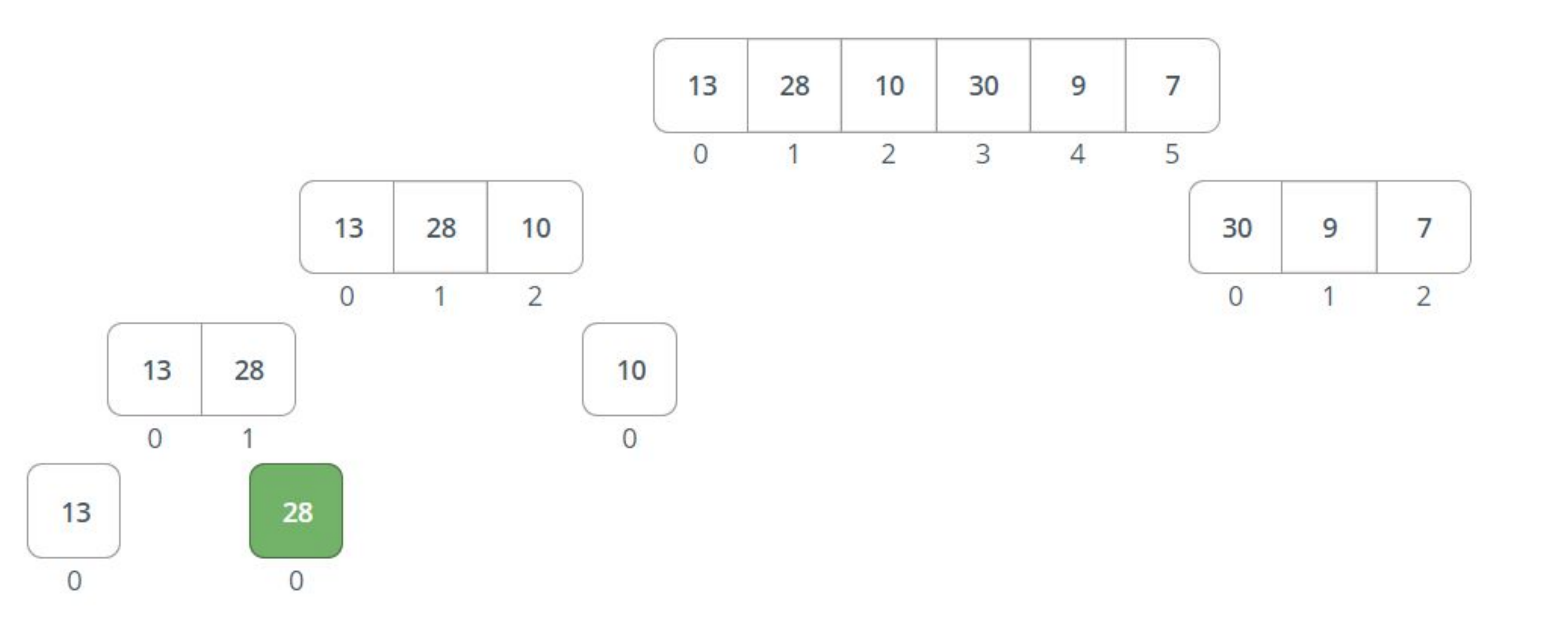
30	9	7
0	1	2

13	28
0	1

10
0

13
0

28
0



13	28	10	30	9	7
0	1	2	3	4	5

13	28	10
0	1	2

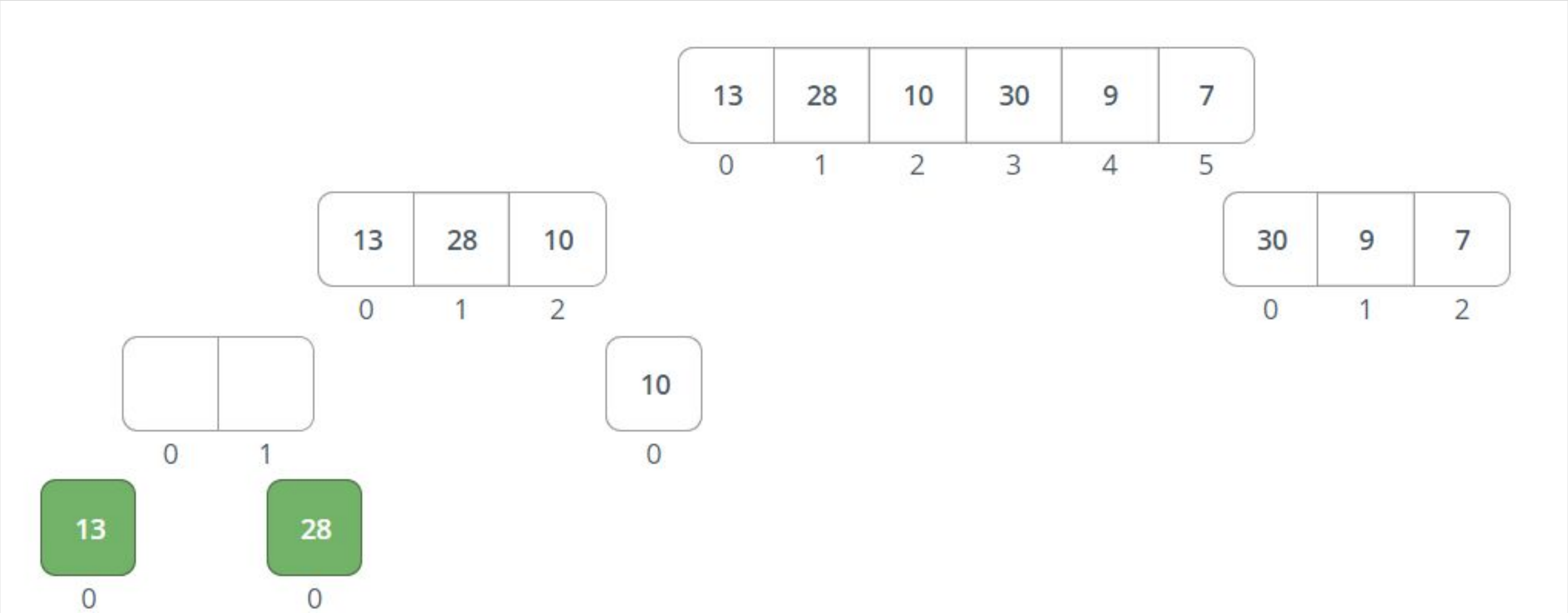
30	9	7
0	1	2

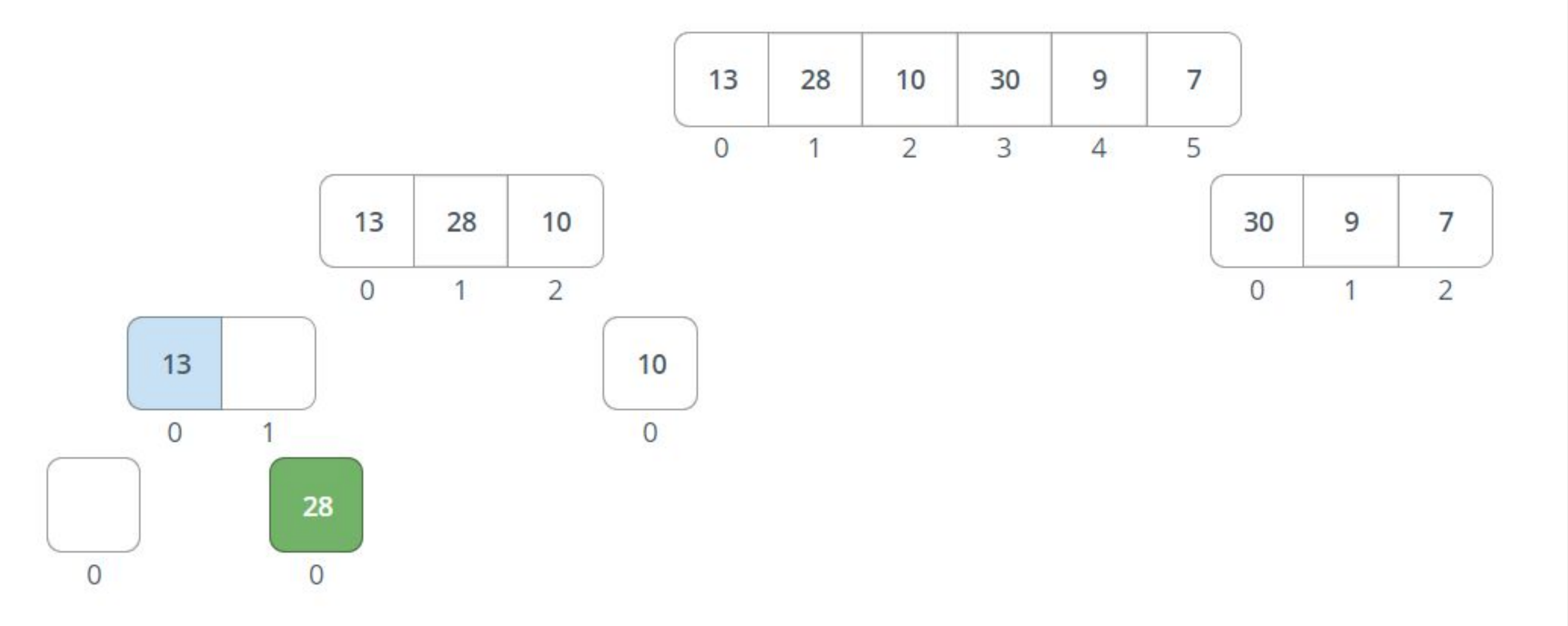
13	28
0	1

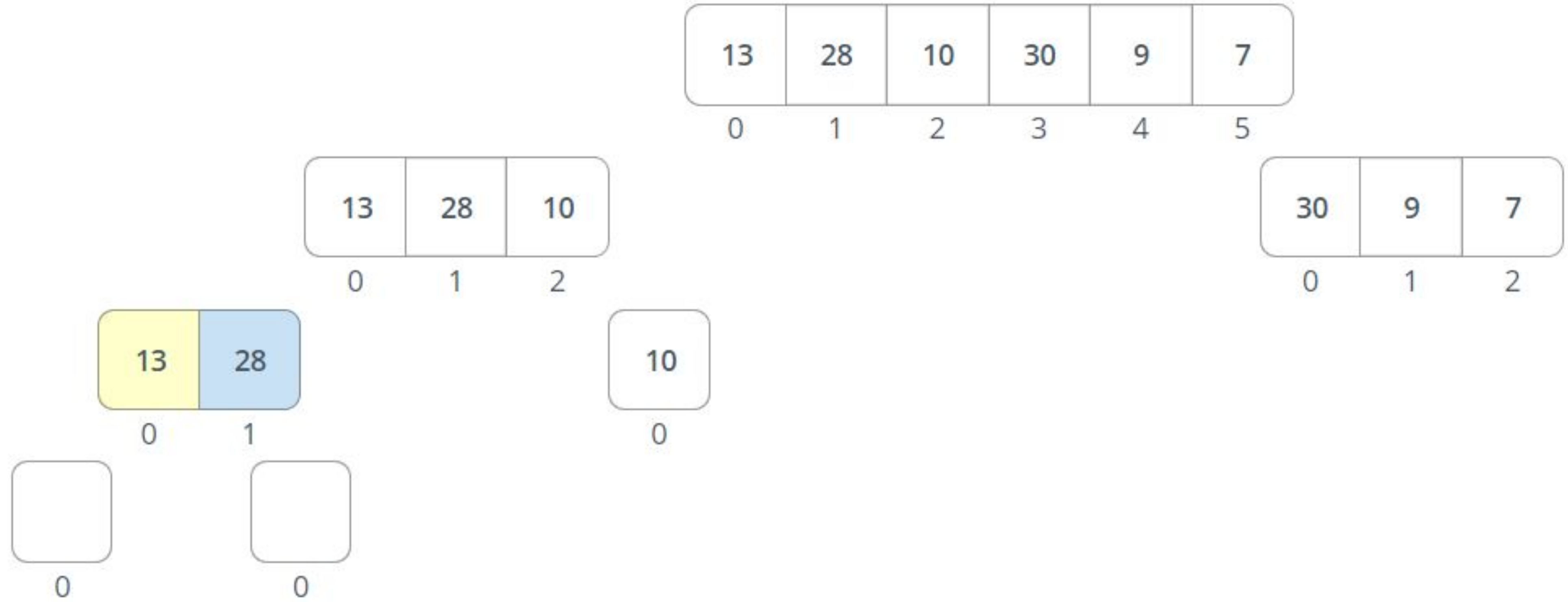
10
0

13
0

28
0


13 28 10 30 9 7
0 1 2 3 4 5
13 28 10
0 1 2
30 9 7
0 1 2
0 1
10
0
13
0
28
0


13 28 10 30 9 7
0 1 2 3 4 5
13 28 10
0 1 2
30 9 7
0 1 2
13
0 1
10
0
0
28
0



13	28	10	30	9	7
0	1	2	3	4	5

13	28	10
0	1	2

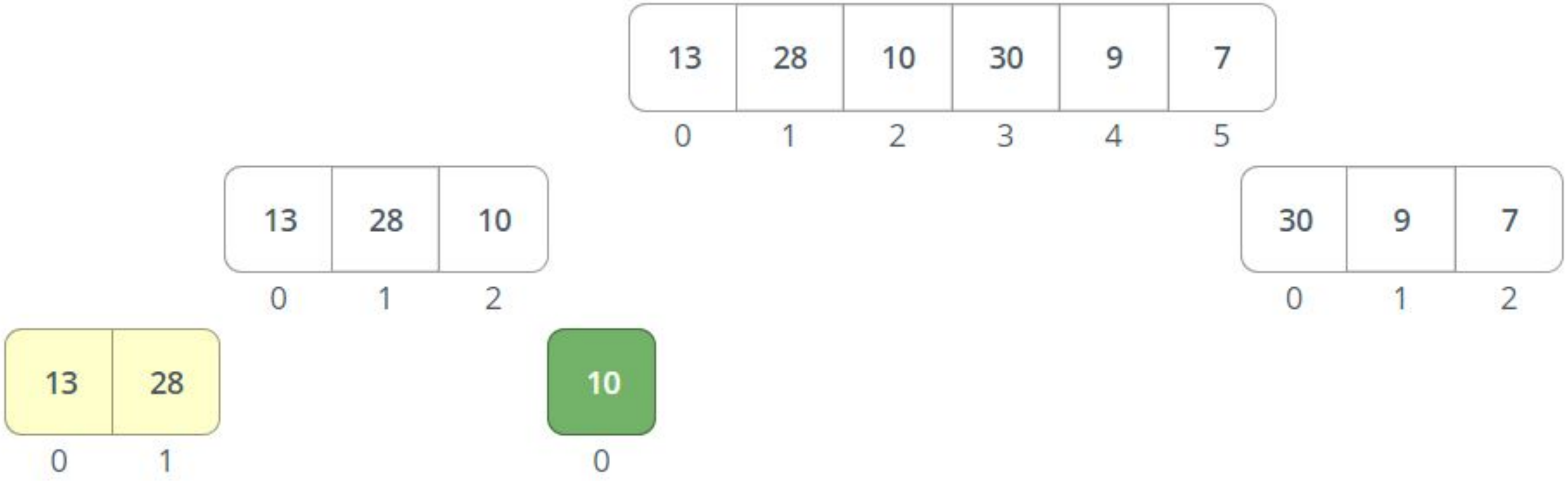
30	9	7
0	1	2

13	28
0	1

10
0

0

0



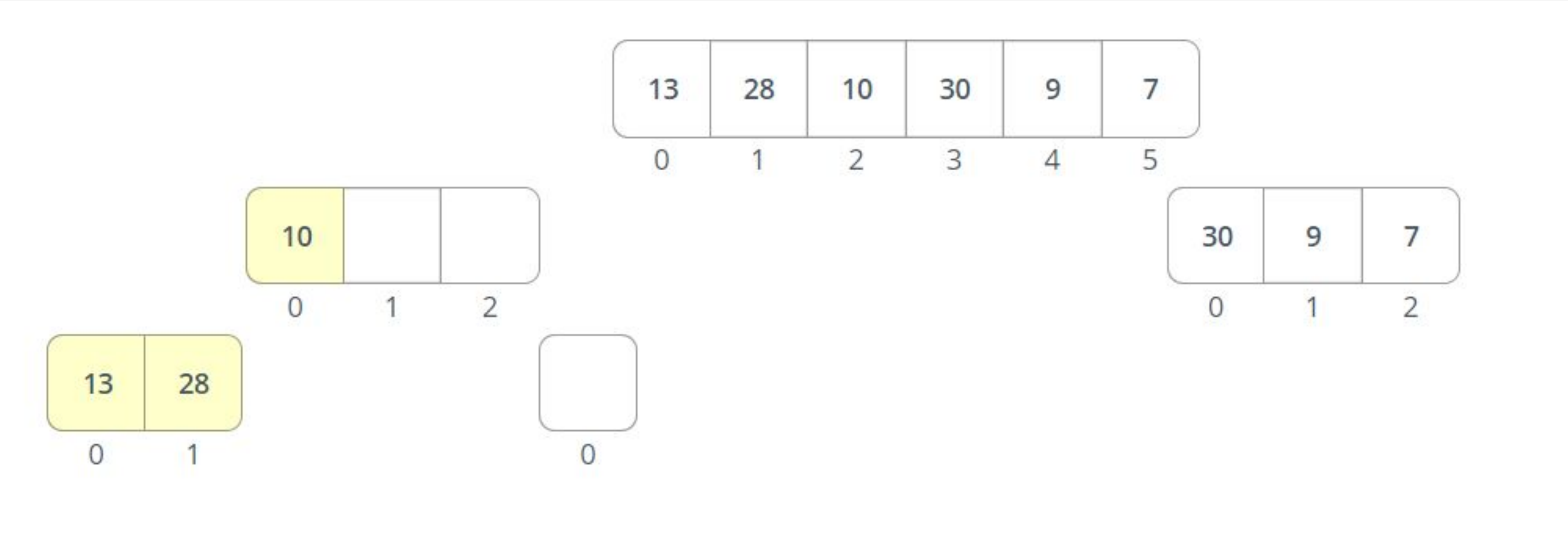
13	28	10	30	9	7
0	1	2	3	4	5

13	28	10
0	1	2

30	9	7
0	1	2

13	28
0	1

10
0



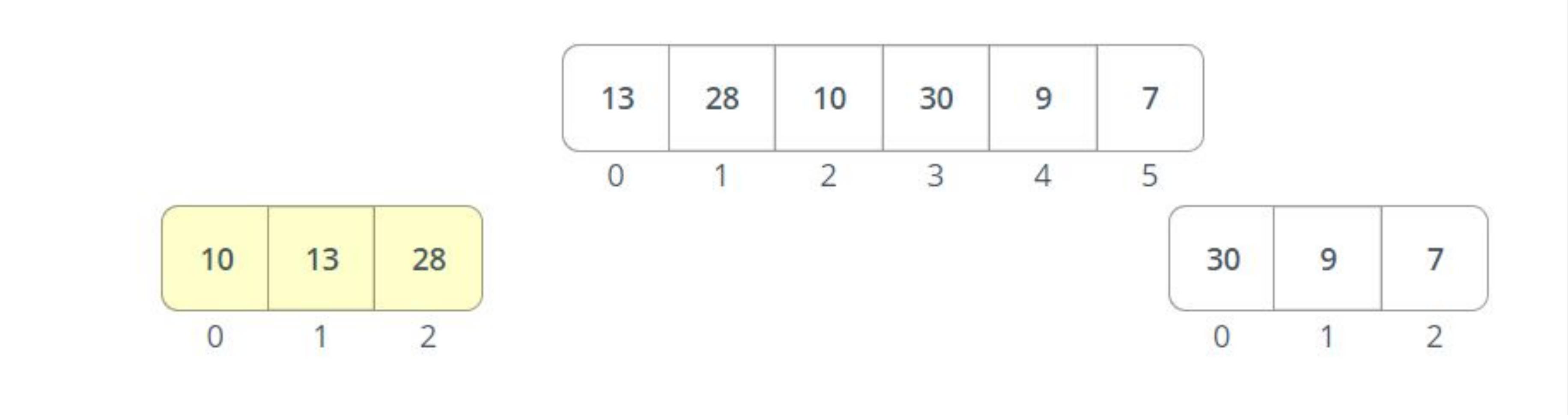
13	28	10	30	9	7
0	1	2	3	4	5

10		
0	1	2

30	9	7
0	1	2

13	28
0	1

0

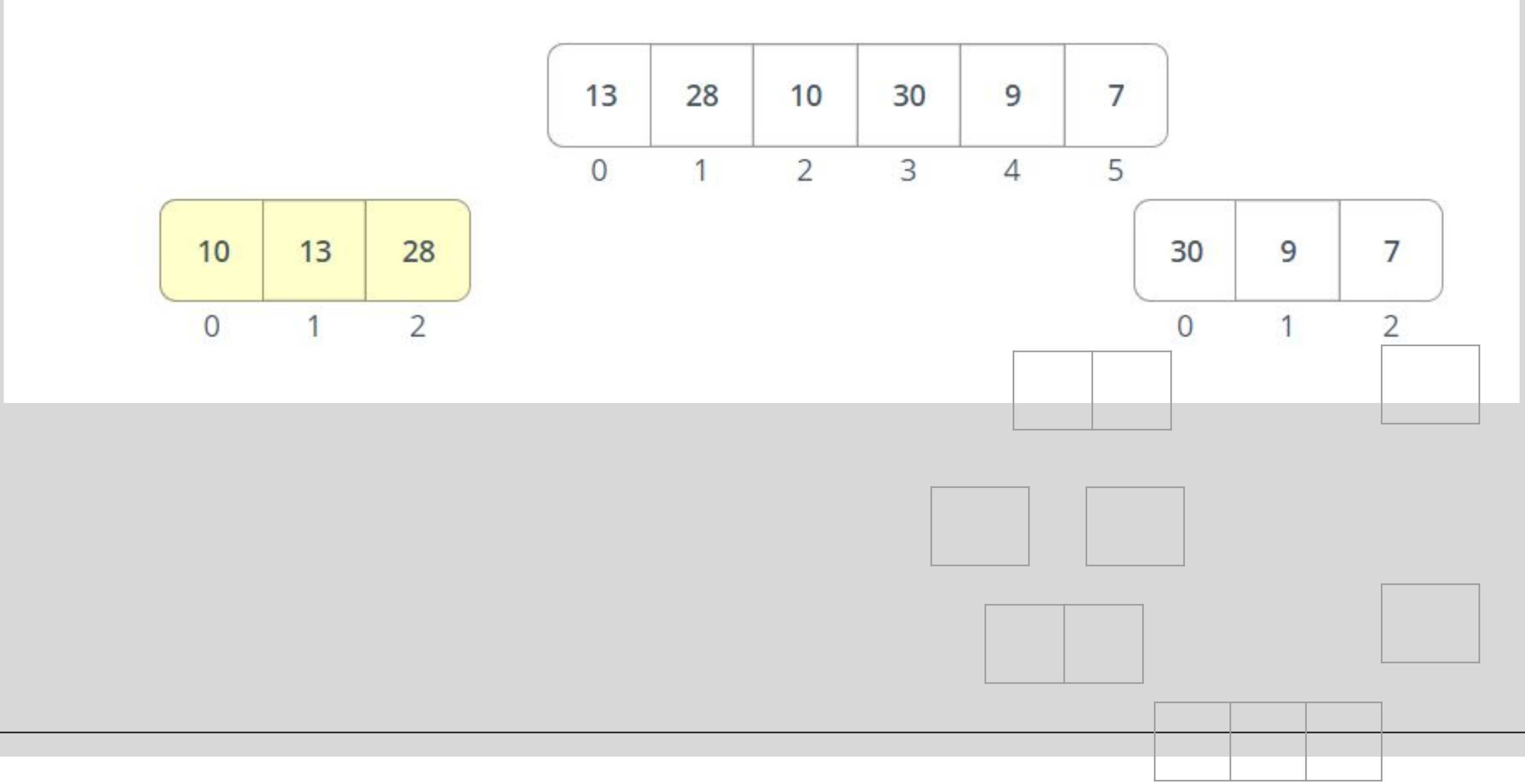


13	28	10	30	9	7
0	1	2	3	4	5

10	13	28
0	1	2

30	9	7
0	1	2

Activity: Simulate for the right sub-array

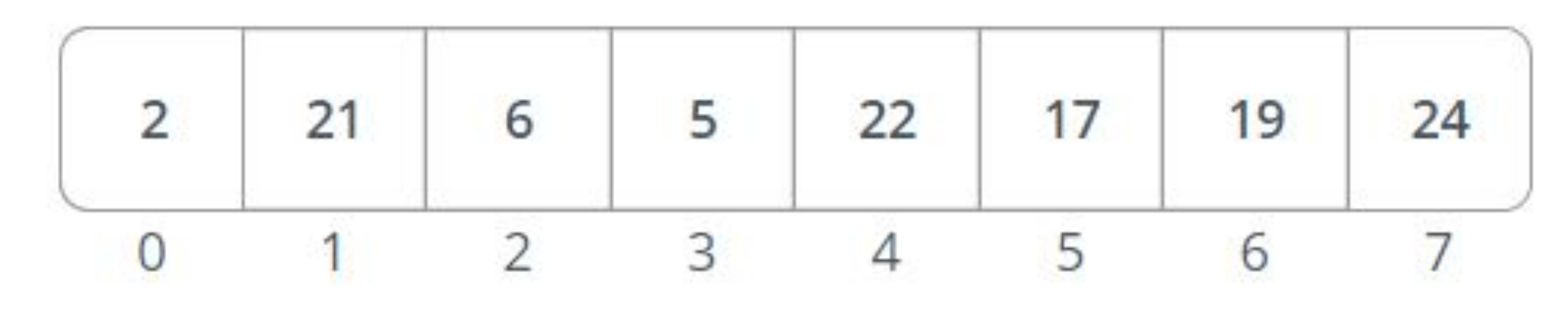


13	28	10	30	9	7
0	1	2	3	4	5

10	13	28
0	1	2

30	9	7
0	1	2

Additional Activity for simulation



2	21	6	5	22	17	19	24
0	1	2	3	4	5	6	7