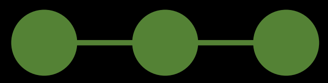
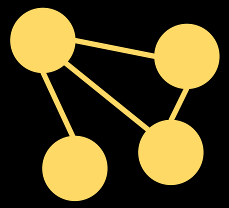
Sorting

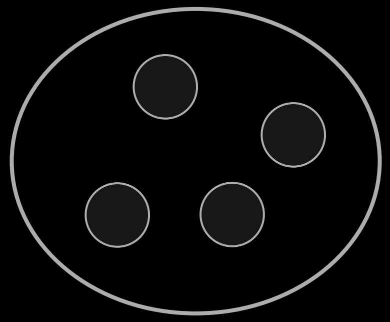
Data Structures & Algorithm

Categories of Data Structures

Linear Ordered	Non-linear Ordered	Not Ordered
Lists	Trees	Sets
Stacks	Graphs	Tables/Maps
Queues		







Problem (Sort)

Input: Unordered Collection of size n, $C_i\ [0.....n-1]$

Output: Ordered Collection of size n, $C_o\ [0.....n-1]$

Example: C_o for ascending sort

$$C_o[0] <= C_o[1] <= \ldots <= C_o[n-1]$$

Selection Sort

Selection Sort

Premise:

- Find the smallest/largest element, e_1 in a Collection, C_i
- Move this element, e_1 to its correct position
- Find the next smallest/largest element, e_2 in C_i
- Move this element, e_2 to its correct position

.
.
.
.

- Repeat this entire process C_i.size() – 1 times

Selection Sort

Example:

Initial array

29	10	14	37	13

Sorted

Selection Sort

Example:

Initial array	29	10	14	37	13
1st pass	10	29	14	37	13

Sorted

Selection Sort

Example:

Initial array	29	10	14	37	13
1st pass	10	29	14	37	13
2nd pass	10	13	14	37	29

Sorted

Selection Sort

Example:

Initial array	29	10	14	37	13
1st pass	10	29	14	37	13
2nd pass	10	13	14	37	29
3rd pass	10	13	14	37	29

Sorted

Selection Sort

Example:

Initial array	29	10	14	37	13
1st pass	10	29	14	37	13
2nd pass	10	13	14	37	29
3rd pass	10	13	14	37	29
4th pass	10	13	14	29	37

Sorted

Selection Sort

Example:

Initial array	29	10	14	37	13
1st pass	10	29	14	37	13
2nd pass	10	13	14	37	29
3rd pass	10	13	14	37	29
4th pass	10	13	14	29	37

Sorted

Selection Sort Pseudocode

```
for fill = 0 to n - 2 do
   Initialize posMin to fill
   for next = fill + 1 to n - 1 do
         if the item at next is less than the item at posMin
             Reset posMin to next
   Exchange the item at posMin with the one at fill
```

Selection Sort Pseudocode

```
for fill = 0 to n – 2 do
    Initialize posMin to fill
    for next = fill + 1 to n – 1 do
            if the item at next is less than the item at posMin
                Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Time Complexity

For very large n we can ignore all but the significant term in the expression, so the number of

- comparisons is $O(n^2)$
- exchanges is $O(n)$

An $O(n^2)$ sort is called a *quadratic sort*

Selection Sort Code

```
void selectionSort(int array[], int size)
{
    for (int i = 0; i < size - 1; i++)
    {
        int min_index = i;
        for (int j = i + 1; j < size; j++)
        {
            if (array[j] < array[min_index])
                min_index = j;
        }
        // put min at the correct position
        swap(&array[min_index], &array[i]);
    }
}
```

https://onlinegdb.com/HJaSC_4eD

Selection Sort Complexity

	Selection Sort	Bubble Sort	Insertion Sort
Worst Case	$O(n^2)$		
Average Case	$O(n^2)$		
Best Case	$O(n^2)$		
Space	$O(1)$		

Bubble Sort

Bubble Sort

Premise:

- Swap adjacent elements, e_i and e_{i+1} in a Collection, C_i if they are out of order
- Repeat swapping till you reach the end of the Collection to bubble up the largest element after each iteration

.
.
.
.

- Repeat this entire process C_i.size() – 1 times stopping at C_i.size() – i after i^{th} iteration

Bubble Sort

Example:

Initial array	29	10	14	37	13

Sorted

Bubble Sort

Example:

Initial array	29	10	14	37	13
1st pass	10	14	29	13	37

Sorted

Bubble Sort

Example:

Initial array	29	10	14	37	13
1st pass	10	14	29	13	37
2nd pass	10	14	13	29	37

Sorted

Bubble Sort

Example:

Initial array	29	10	14	37	13
1st pass	10	14	29	13	37
2nd pass	10	14	13	29	37
3rd pass	10	13	14	29	37

Sorted

Bubble Sort

Example:

Initial array	29	10	14	37	13
1st pass	10	14	29	13	37
2nd pass	10	14	13	29	37
3rd pass	10	13	14	29	37
4th pass	10	13	14	29	37

Sorted

Bubble Sort

Example:

Initial array	29	10	14	37	13
1st pass	10	14	29	13	37
2nd pass	10	14	13	29	37
3rd pass	10	13	14	29	37
4th pass	10	13	14	29	37
Final array	10	13	14	29	37

Sorted

Bubble Sort Pseudocode

```
For pass = 0 to n-1
  Sorted = true
  for each pair of adjacent array elements between pass and n
      if the values in a pair are out of order
          Exchange the values
          Sorted = false
while the array is not sorted
```

Bubble Sort Pseudocode

```
For pass = 0 to n-1
   Sorted = true
   for each pair of adjacent array elements between pass and n
         if the values in a pair are out of order
               Exchange the values
               Sorted = false
while the array is not sorted
```

Time Complexity

In the worst case,

- comparisons is $O(n^2)$
- exchanges is $O(n^2)$

Compared to selection sort with its $O(n^2)$ comparisons and $O(n)$ exchanges, bubble sort usually performs worse

Bubble Sort Code

```c
void bubbleSort(int array[], int size)
{
  for (int i = 0; i < size - 1; i++)
  {
    int swapped = 0;
    for (int j = 0; j < size - i - 1; ++j)
    {
      if (array[j] > array[j + 1])
      {
        int temp = array[j];
        array[j] = array[j + 1];
        array[j + 1] = temp;
        swapped = 1;
      }
    }
    // If there is no swapping in the last swap, then the array is already sorted.
    if (swapped == 0)
      break;
  }
}
```

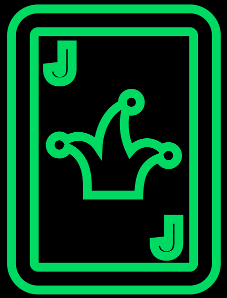
https://onlinegdb.com/H1thqETkv

Bubble Sort Complexity

	Selection Sort	Bubble Sort	Insertion Sort
Worst Case	$O(n^2)$	$O(n^2)$	
Average Case	$O(n^2)$	$O(n^2)$	
Best Case	$O(n^2)$	$O(n)$	
Space	$O(1)$	$O(1)$	

Insertion Sort

Insertion Sort



Premise:

- Keeps a track of two regions: Sorted and Unsorted
- Initially, the sorted region has one element
- Insert the first element in the unsorted region in the correct place in the sorted region
- Repeat this entire process till there are no more elements in unsorted region

Insertion Sort

Example:

Initial array	29	10	14	37	13

Sorted

Insertion Sort

Example:

Initial array	29	10	14	37	13
1st pass	29	10	14	37	13

Sorted   Unsorted

Sorted

Insertion Sort

Example:

Initial array	29	10	14	37	13
1st pass	**29**	10	14	37	13
2nd pass	**10**	**29**	14	37	13

Sorted

Insertion Sort

Example:

Initial array	29	10	14	37	13
1st pass	**29**	10	14	37	13
2nd pass	**10**	**29**	14	37	13
3rd pass	**10**	**14**	**29**	37	13

Sorted

Insertion Sort

Example:

Initial array	29	10	14	37	13
1st pass	29	10	14	37	13
2nd pass	10	29	14	37	13
3rd pass	10	14	29	37	13
4th pass	10	14	29	37	13

Sorted

Insertion Sort

Example:

Initial array	29	10	14	37	13
1st pass	29	10	14	37	13
2nd pass	10	29	14	37	13
3rd pass	10	14	29	37	13
4th pass	10	14	29	37	13
5th Pass/ Final array	10	13	14	29	37

Sorted

Insertion Sort Pseudocode

```
for each array element from the second (nextPos = 1) to the last
    nextPos  is the position of the element to insert
    Save the value of the element to insert in nextVal
    while nextPos > 0 and the element at nextPos – 1 > nextVal
          Shift the element at nextPos – 1 to position nextPos
          Decrement nextPos by 1
    Insert nextVal at nextPos
```

Insertion Sort Pseudocode

```
for each array element from the second (nextPos = 1) to the last
    nextPos  is the position of the element to insert
    Save the value of the element to insert in nextVal
    while nextPos > 0 and the element at nextPos - 1 > nextVal
          Shift the element at nextPos - 1 to position nextPos
          Decrement nextPos by 1
    Insert nextVal at nextPos
```

Time Complexity

In the worst case,

- comparisons is $O(n^2)$
- exchanges is $O(n^2)$

Insertion Sort Code

```
void insertionSort(int array[], int size)
{
  for (int i = 1; i < size; i++)
  {
    int key = array[i];
    int j = i-1;

    // Compare key with each element in sorted till smaller value is found
    while (key < array[j] && j >= 0)
    {
      array[j+1] = array[j];
      j--;
    }
    array[j+1] = key;
  }
}
```

https://onlinegdb.com/rJgA3LAkw

Insertion Sort Complexity

	Selection Sort	Bubble Sort	Insertion Sort
Worst Case	$O(n^2)$	$O(n^2)$	$O(n^2)$
Average Case	$O(n^2)$	$O(n^2)$	$O(n^2)$
Best Case	$O(n^2)$	$O(n)$	$O(n)$
Space	$O(1)$	$O(1)$	$O(1)$

Resources

- https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
- https://www.programiz.com/dsa
- https://www.youtube.com/user/AlgoRythmics/videos
- https://www.toptal.com/developers/sorting-algorithms

Questions

Shell Sort

Shell Sort

Premise:

- A Shell sort is a type of insertion sort, but with $O(n^{3/2})$ or better performance than the $O(n^2)$ sorts
- Instead of sorting the entire array, Shell sort sorts many smaller subarrays using insertion sort before sorting the entire array
- Named after Donald Shell

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

`Gap = n/2`

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

`Gap = n/2`

7	2
4	8
9	6
3	5

Shell Sort

Example:

Initial array	7	4	9	3	2	8	6	5	Gap = n/2
	2				7				
		4				8			
			6				9		
				3				5	

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

`Gap = n/2`

2	4	6	3	7	8	9	5

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	4	6	3	7	8	9	5

Gap = n/4

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	4	6	3	7	8	9	5

Gap = n/4

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	4	6	3	7	8	9	5

Gap = n/4

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	8	9	5

Gap = n/4

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5	Gap = n/2
2	3	6	4	7	8	9	5	Gap = n/4

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	8	9	5

Gap = n/4

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	8	9	5

Gap = n/4

Shell Sort

Example:

Initial array	7	4	9	3	2	8	6	5	Gap = n/2
	2	3	6	4	7	8	9	5	Gap = n/4

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	8	9	5

Gap = n/4

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	5	9	8

Gap = n/4

Shell Sort

Example:

Initial array	7	4	9	3	2	8	6	5	Gap = n/2
	2	3	6	4	7	5	9	8	Gap = n/4

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	5	9	8

Gap = n/4

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5	Gap = n/2

2	3	6	4	7	5	9	8	Gap = n/4

2	3	6	4	7	5	9	8	Gap = n/8

Shell Sort

Example:

Initial array	7	4	9	3	2	8	6	5	Gap = n/2
	2	3	6	4	7	5	9	8	Gap = n/4
	2	3	6	4	7	5	9	8	Gap = n/8

Shell Sort

Example:

Initial array									
Initial array	7	4	9	3	2	8	6	5	Gap = n/2
	2	3	6	4	7	5	9	8	Gap = n/4
	2	3	6	4	7	5	9	8	Gap = n/8

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	5	9	8

Gap = n/4

2	3	6	4	7	5	9	8

Gap = n/8

Shell Sort

Example:

Initial array	7	4	9	3	2	8	6	5	Gap = n/2
	2	3	6	4	7	5	9	8	Gap = n/4
	2	3	4	6	7	5	9	8	Gap = n/8

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	5	9	8

Gap = n/4

2	3	4	6	7	5	9	8

Gap = n/8

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	5	9	8

Gap = n/4

2	3	4	6	7	5	9	8

Gap = n/8

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5	Gap = n/2
2	3	6	4	7	5	9	8	Gap = n/4
2	3	4	6	5	7	9	8	Gap = n/8

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	5	9	8

Gap = n/4

2	3	4	6	5	7	9	8

Gap = n/8

Shell Sort

Example:

Initial array	7	4	9	3	2	8	6	5	Gap = n/2
	2	3	6	4	7	5	9	8	Gap = n/4
	2	3	4	5	6	7	9	8	Gap = n/8

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	5	9	8

Gap = n/4

2	3	4	5	6	7	9	8

Gap = n/8

Shell Sort

Example:

Initial array	7	4	9	3	2	8	6	5	Gap = n/2
	2	3	6	4	7	5	9	8	Gap = n/4
	2	3	4	5	6	7	9	8	Gap = n/8

Shell Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Gap = n/2

2	3	6	4	7	5	9	8

Gap = n/4

2	3	4	5	6	7	8	9

Gap = n/8

Shell Sort

Example:

Initial array

Initial array	7	4	9	3	2	8	6	5
	2	3	6	4	7	5	9	8
	2	3	4	5	6	7	8	9

Gap = n/2

Gap = n/4

Gap = n/8

Shell Sort Pseudocode

```
Shell Sort Algorithm

1.     Set the initial value of gap to n / 2
2.     while gap > 0
3.       for each array element from position gap to the last element
4.                 Insert this element where it belongs in its subarray.
5.       if gap is 2, set it to 1
6.       else gap = gap / 2.2. // chosen by experimentation
```

Shell Sort

Time Complexity:

- A general analysis of Shell sort is an open research problem in computer science
- Performance depends on how the decreasing sequence of values for gap is chosen
- If successive powers of 2 are used for gap, performance is $O(n^2)$
- If successive values for gap are based on Hibbard's sequence,

  $2^k - 1$ (i.e. 31, 15, 7, 3, 1)

  it can be proven that the performance is $O(n^{3/2})$
- Other sequences give similar or better performance

Shell Sort Time Complexity

	Shell Sort
Worst Case	$O(n^2)$
Average Case	$O(n^{5/4})$
Best Case	$O(n^{7/6})$
Space	$O(1)$

Merge Sort

Merge Sort

Premise:

- Merge sort splits the array in half, sorts the two smaller halves, then merges the two sorted halves together.
- Divide the array to be sorted into smaller subarrays till you reach a size of 1
- In the Conquer step, sort the two subarrays
- In the Combine step, combine two sorted arrays

.
.
.

- Repeat this till you merge all elements in one array

Merge Sort

Example:

Initial array

6	5	22	10	9	1

Sorted

Merge Sort

Example:

Divide

Initial array

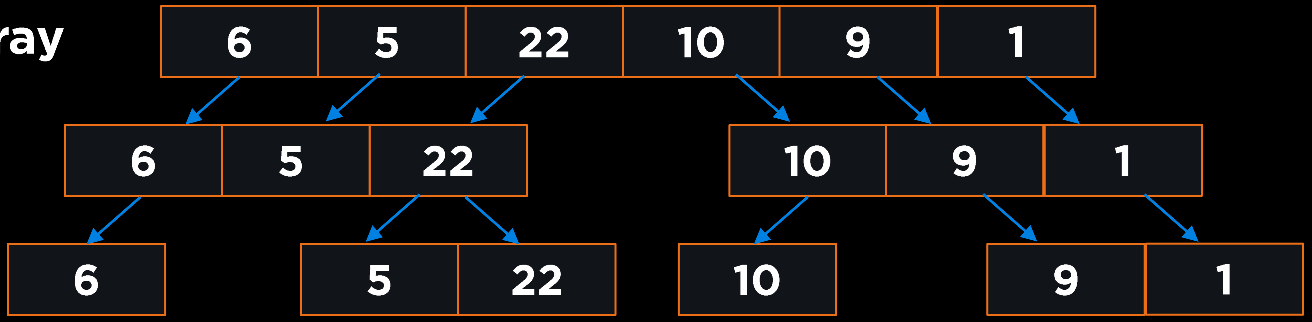
Sorted

Merge Sort

Example:

Divide

Initial array



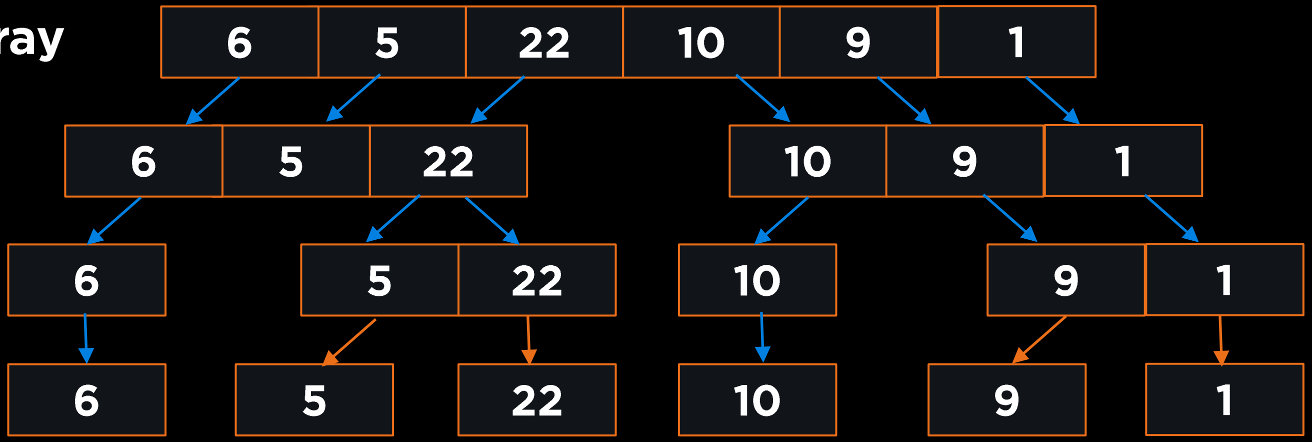
Sorted

Merge Sort

Example:

Divide

Initial array

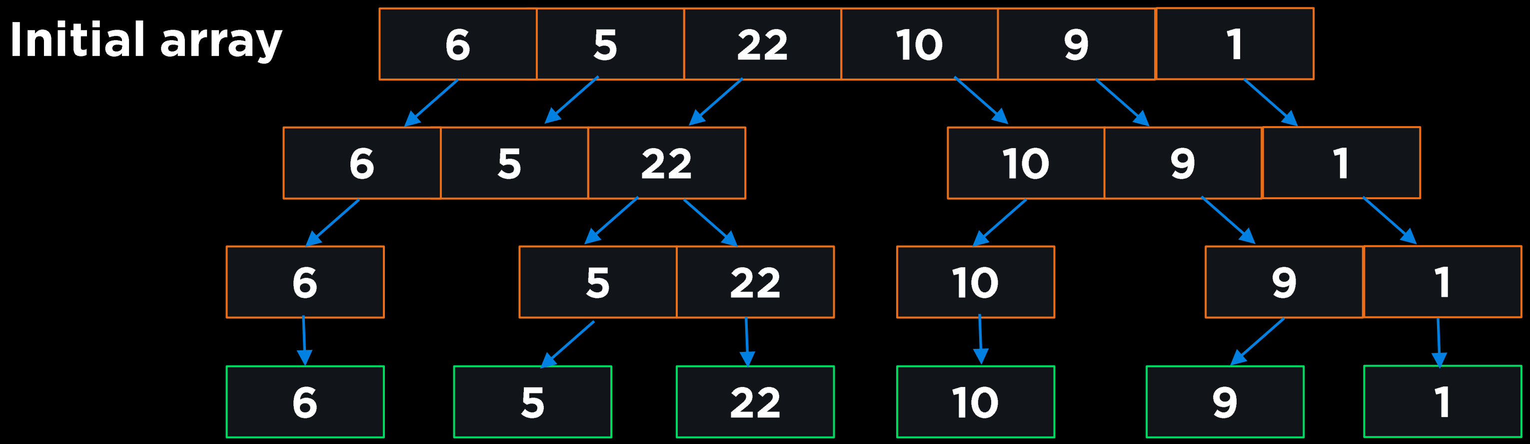


Sorted

Merge Sort

Example:

Conquer

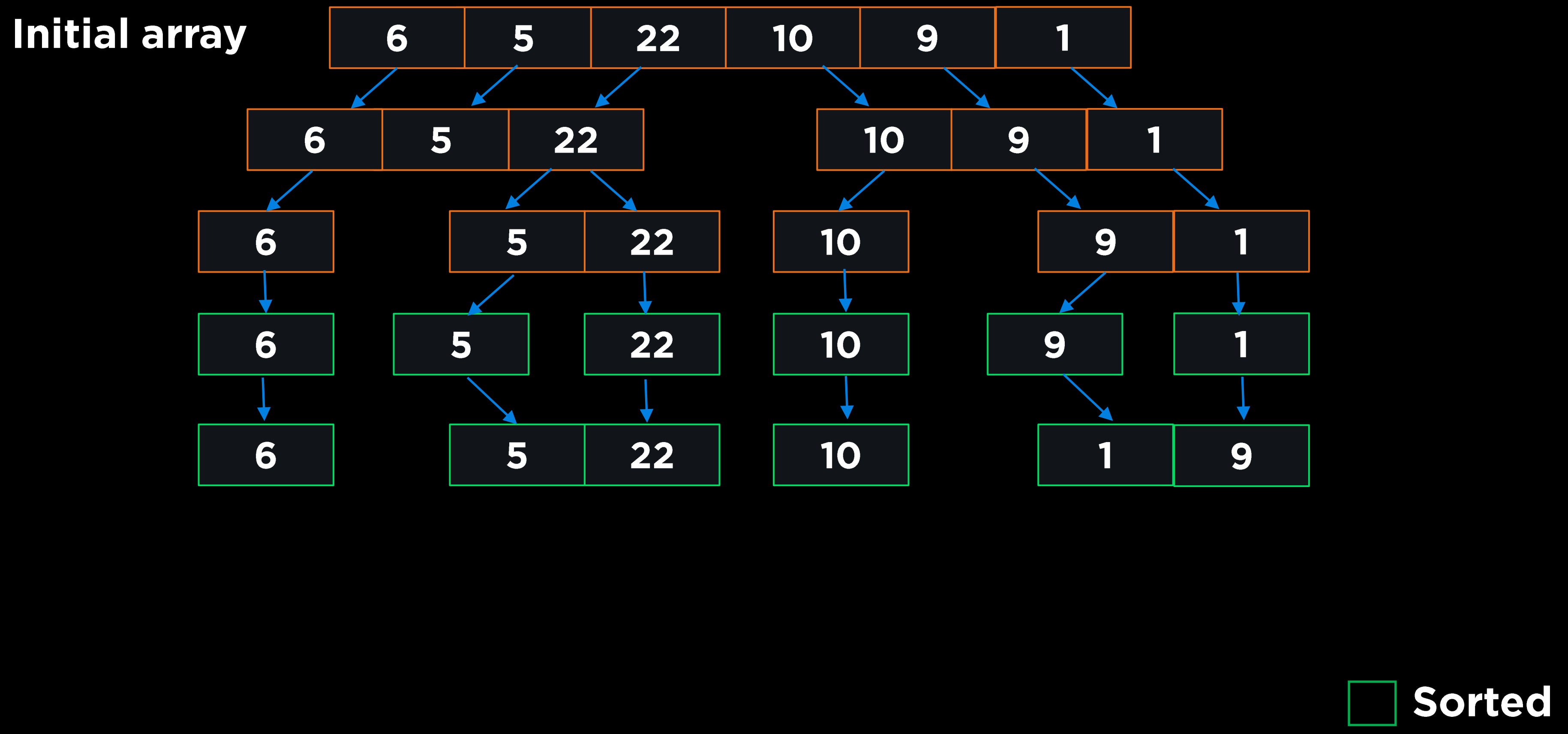


Sorted

Merge Sort

Example:

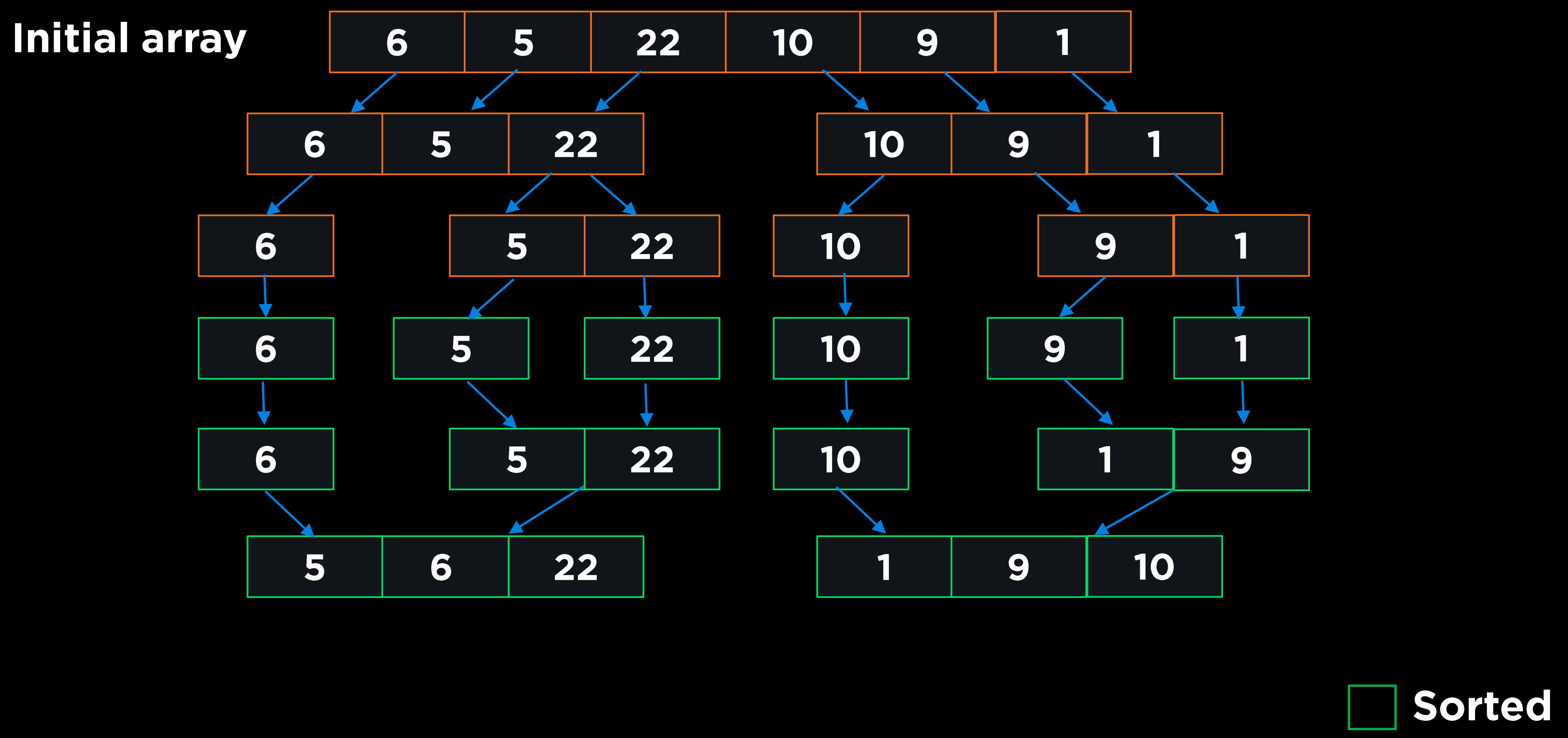
Combine



Merge Sort

Example:

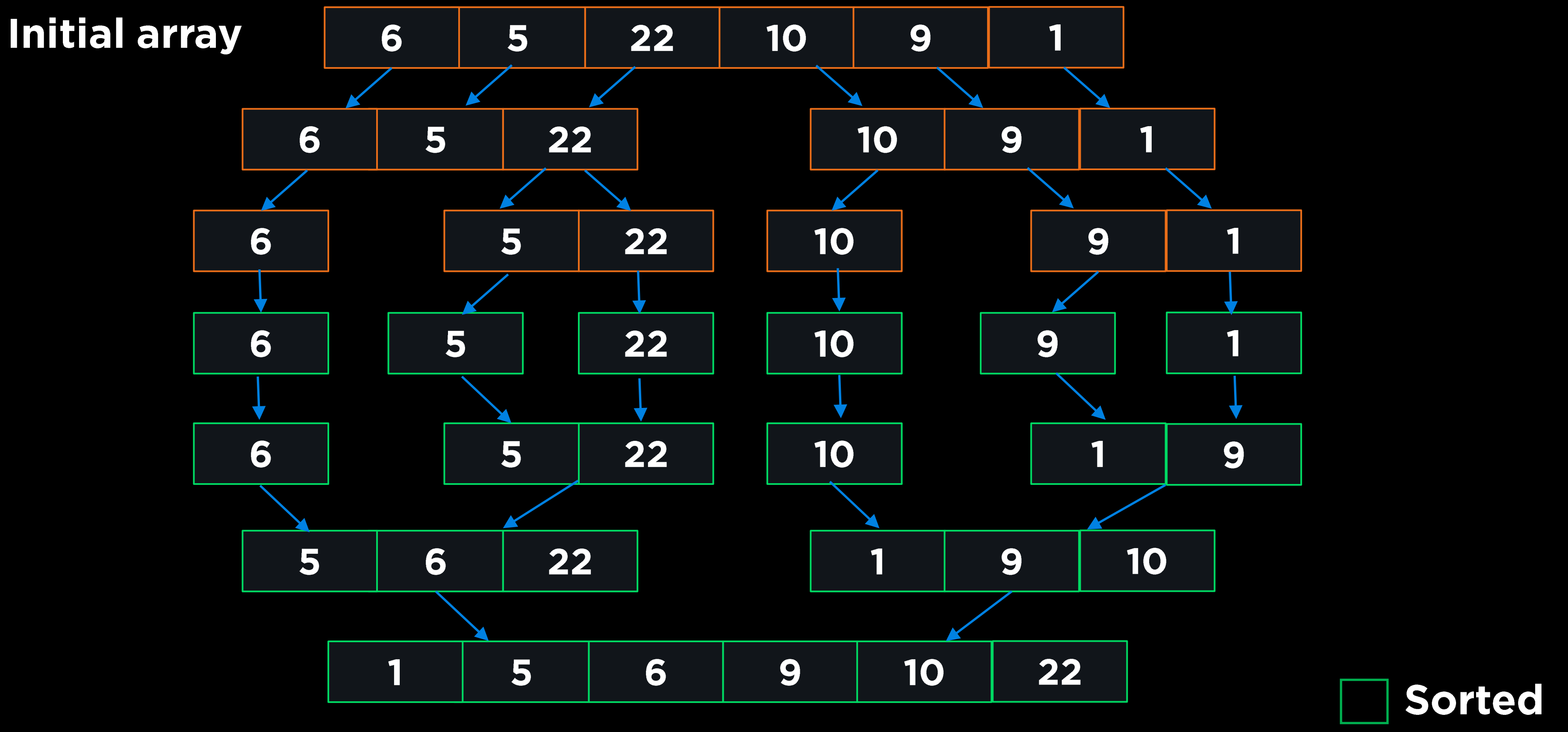
Combine



Merge Sort

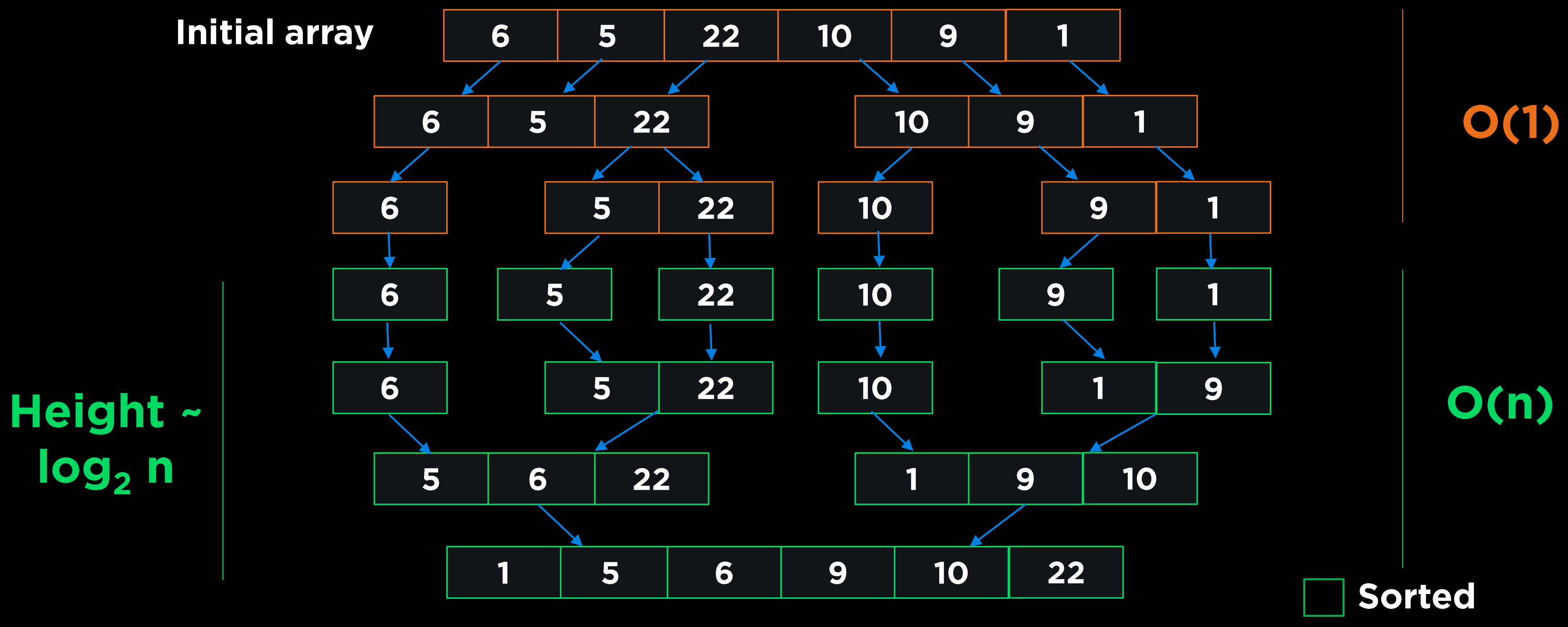
Example:

Combine



Merge Sort

Example:



Merge Sort Pseudocode

```
mergeSort(int [] numbers, int start, int end)
{
     if (start<end)  //base case is start = end and sorting an array of 1
     {
           middle = (start+end)/2;
           mergeSort(numbers, start, middle);
           mergeSort(numbers, middle+1, end);
           merge(numbers, start, middle, end);
      }
}
```

```
mergeSort(A, 0, length(A)-1)
```

```
Merge Algorithm
1.     Access the first item from both sequences.
2.     while not finished with either sequence
3.          Compare the current items from the two sequences, copy the smaller
            current item to the output sequence and access the next item from the input sequence whose item was copied.
4.     Copy any remaining items from the first sequence to the output sequence.
5.     Copy any remaining items from the second sequence to the output sequence.
```

Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    // m is the point where the array is divided into two subarrays
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```

```
mergeSort(A, 0, length(A)-1)
```

https://onlinegdb.com/HJT6FfMlv

Merge Sort Code

```
// Merge two subarrays from arr
void merge(int arr[], int left, int mid, int right)
{
  // Create X ← arr[left..mid] & Y ← arr[mid+1..right]
  int n1 = mid - left + 1;
  int n2 = right - mid;
  int X[n1], Y[n2];

  for (int i = 0; i < n1; i++)
    X[i] = arr[left + i];
  for (int j = 0; j < n2; j++)
    Y[j] = arr[mid + 1 + j];

// Merge the arrays X and Y into arr
  int i, j, k;
  i = 0;
  j = 0;
  k = left;
```

```
while (i < n1 && j < n2)
{
  if (X[i] <= Y[j])
  {
    arr[k] = X[i];
    i++;
  }
  else
  {
    arr[k] = Y[j];
    j++;
  }
  k++;
}
```

```
  // When we run out of elements in either X or Y append the remaining elements
  while (i < n1)
  {
    arr[k] = X[i];
    i++;
    k++;
  }
  while (j < n2)
  {
    arr[k] = Y[j];
    j++;
    k++;
  }
}
```

https://onlinegdb.com/HJT6FfMlv

Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```

arr

15	2	7	0
0	1	2	3

`mergeSort(arr, 0, 3)`

https://onlinegdb.com/HJT6FfMlv

Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```

arr

15	2	7	0
0	1	2	3

`mergeSort(arr, 0, 3)`

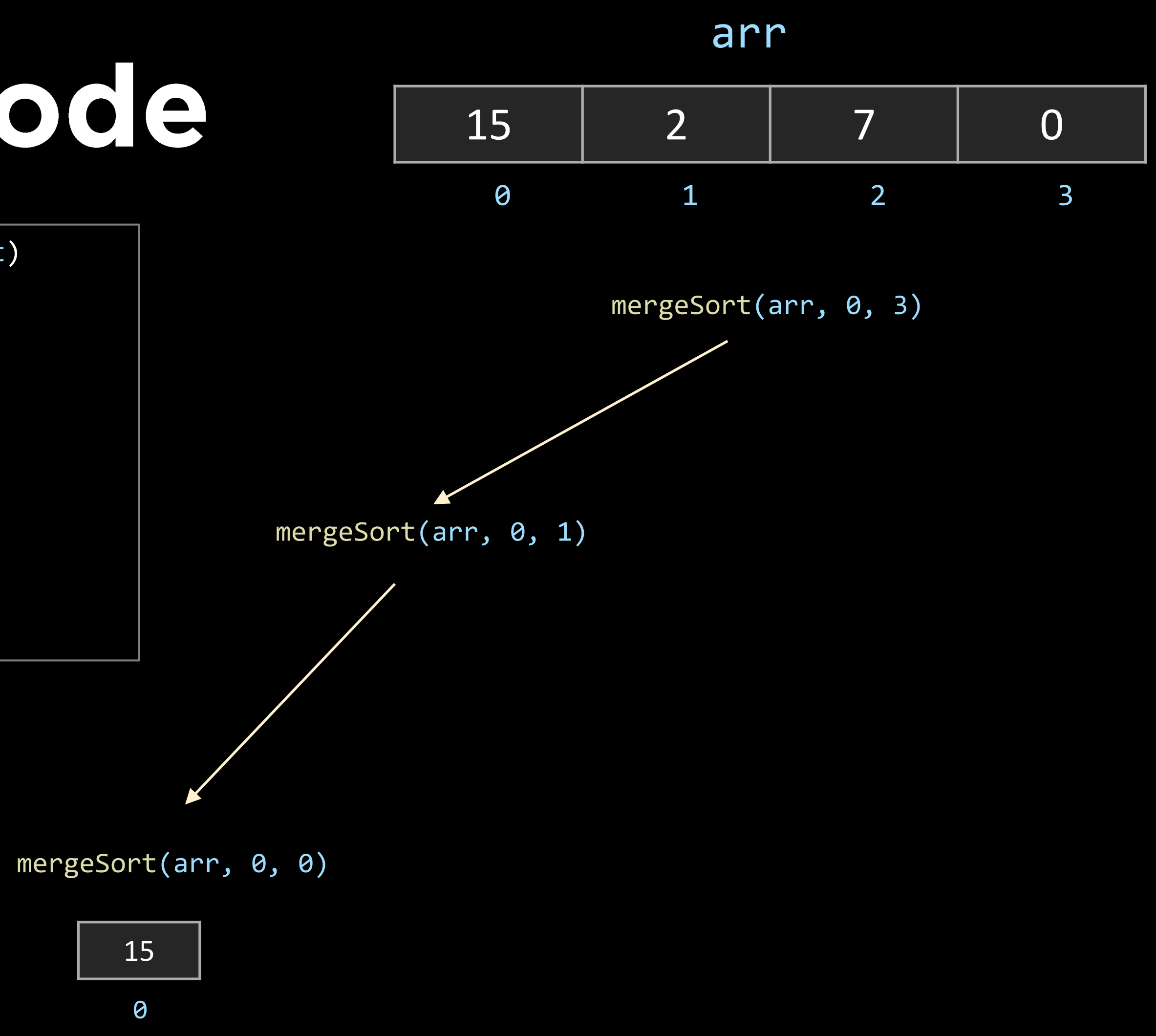
`mergeSort(arr, 0, 1)`

https://onlinegdb.com/HJT6FfMlv

Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```

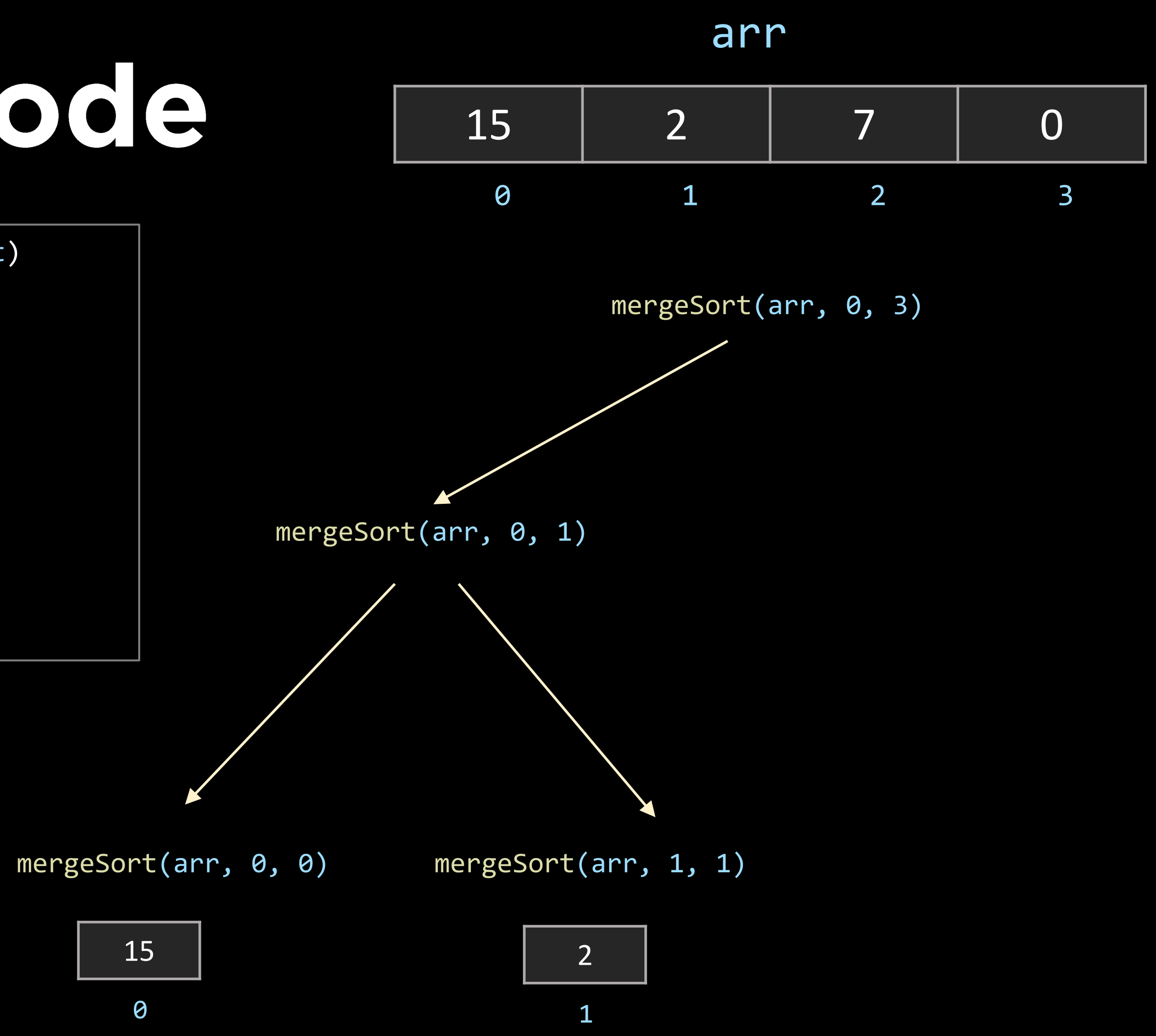


https://onlinegdb.com/HJT6FfMlv

Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```

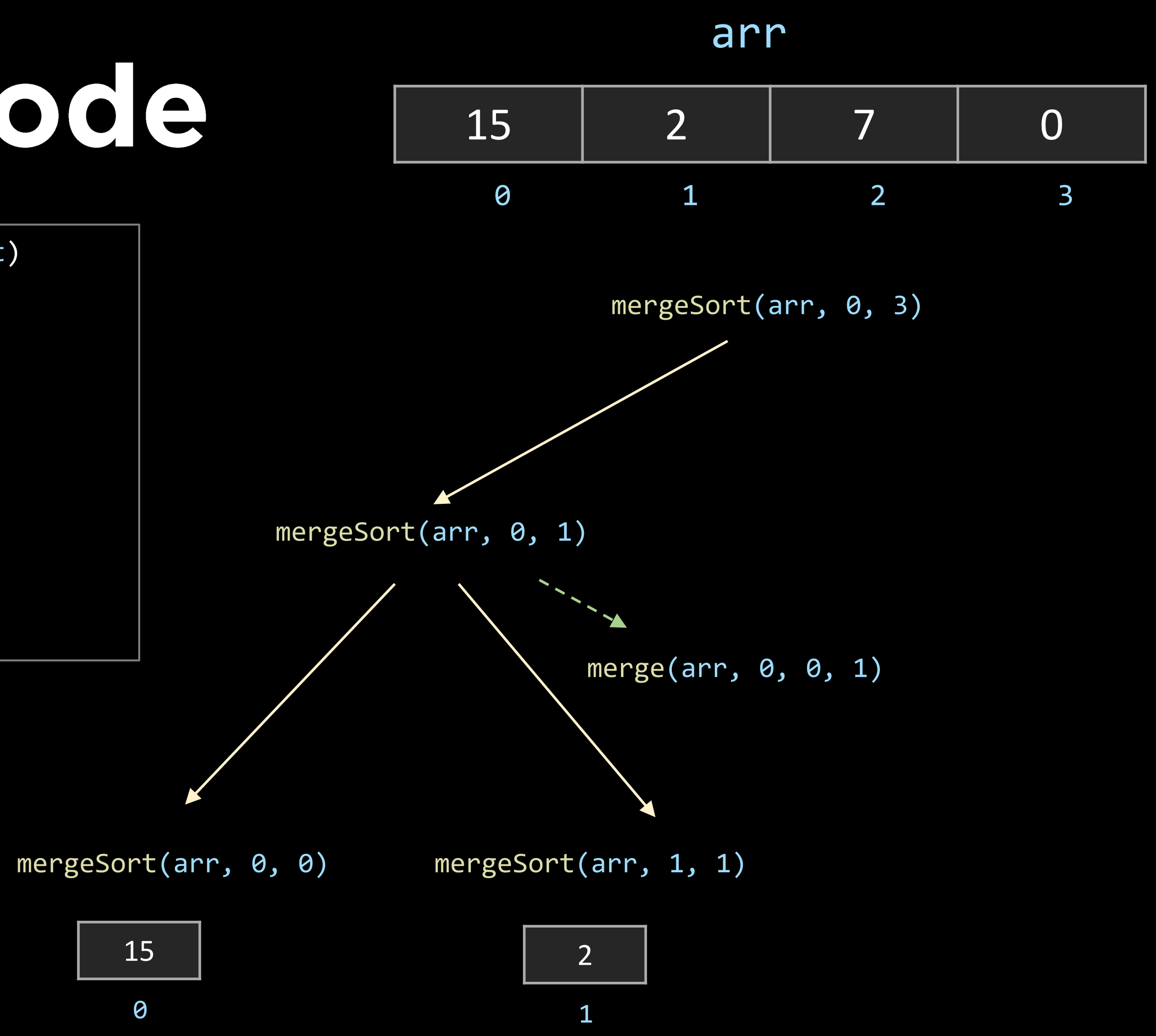


https://onlinegdb.com/HJT6FfMlv

Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```

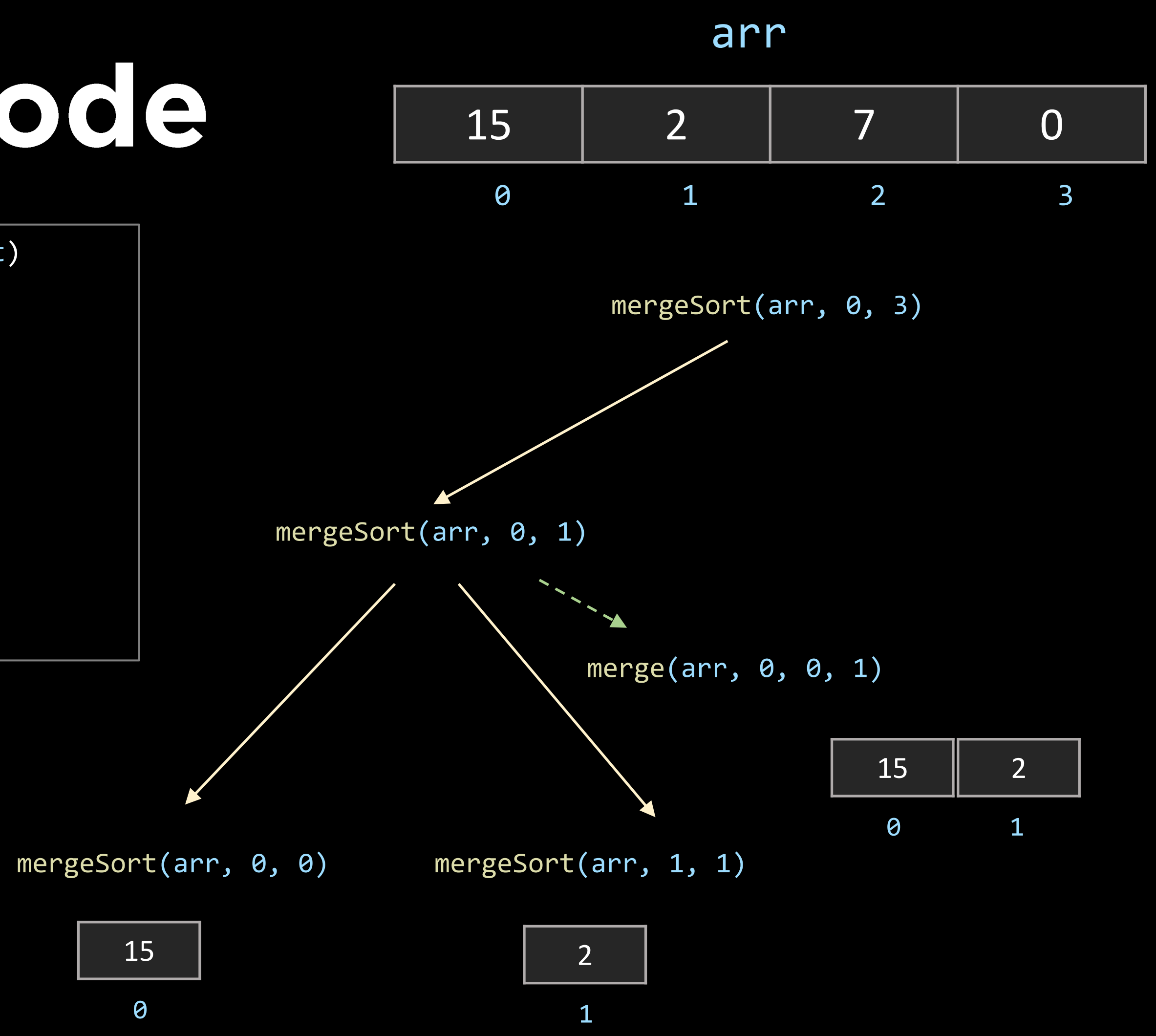


https://onlinegdb.com/HJT6FfMlv

Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```

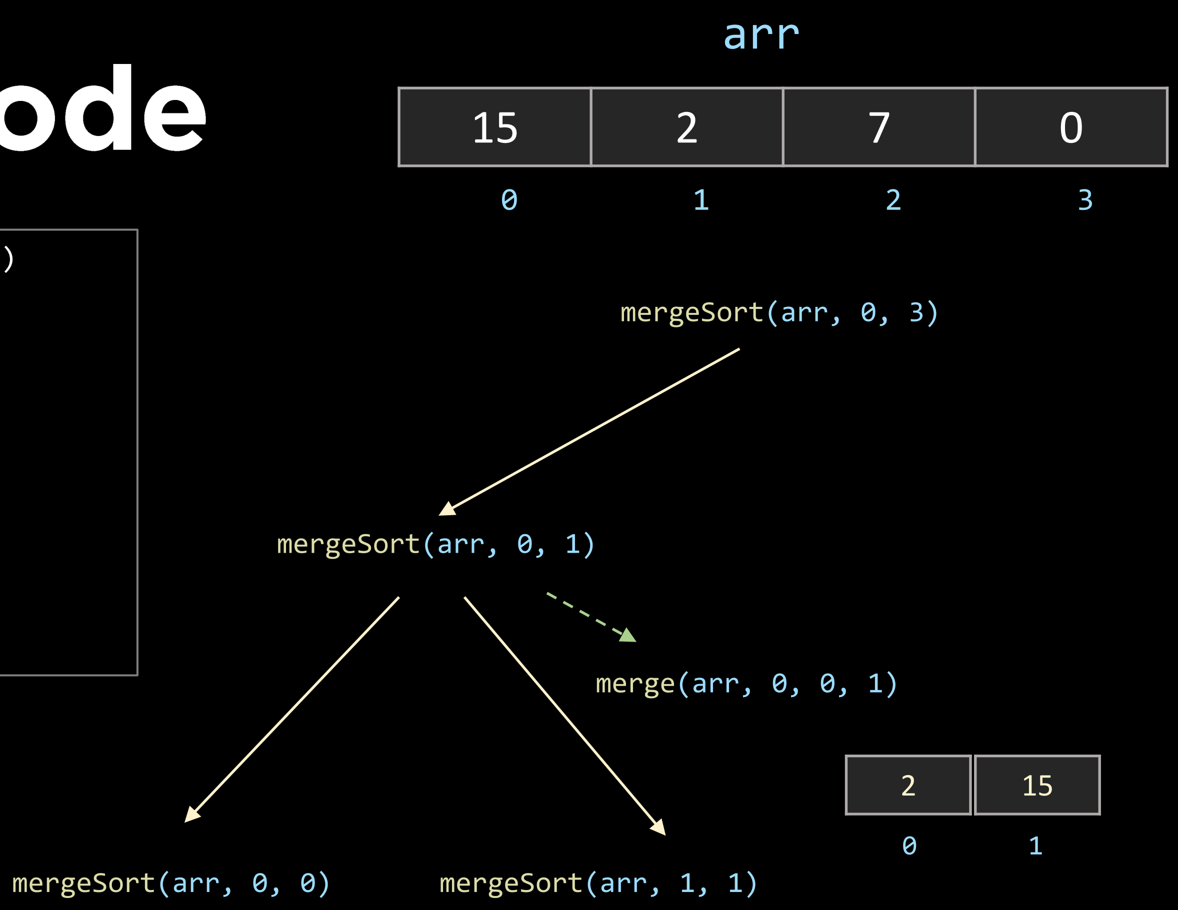


https://onlinegdb.com/HJT6FfMlv

Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```



https://onlinegdb.com/HJT6FfMlv

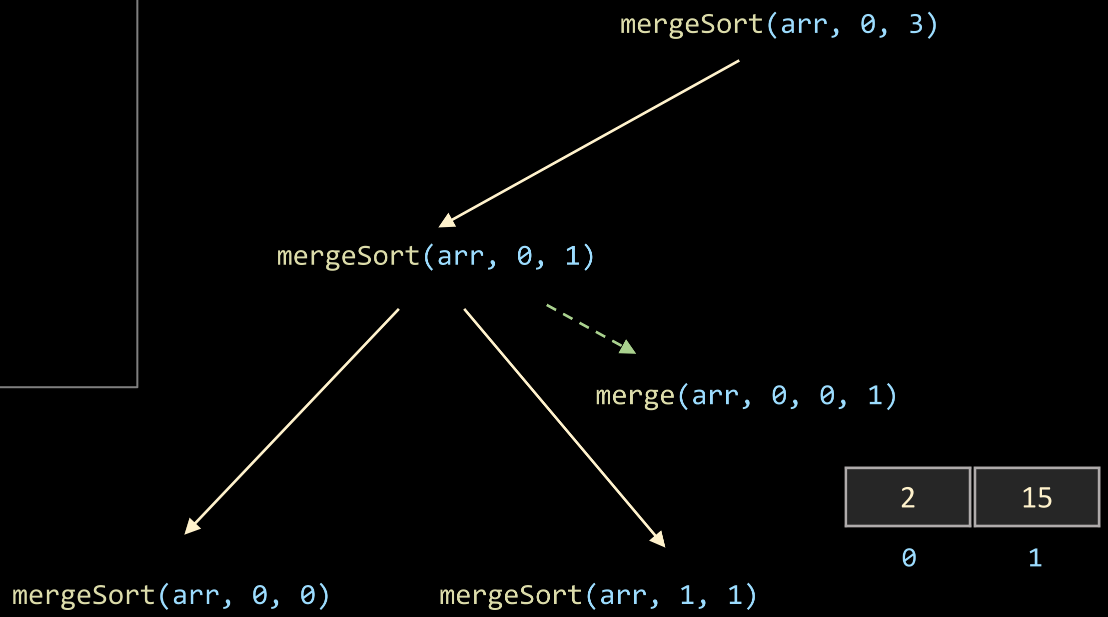
Merge Sort Code

arr

15	2	7	0
0	1	2	3

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```



arr

2	15	7	0
0	1	2	3

https://onlinegdb.com/HJT6FfMlv

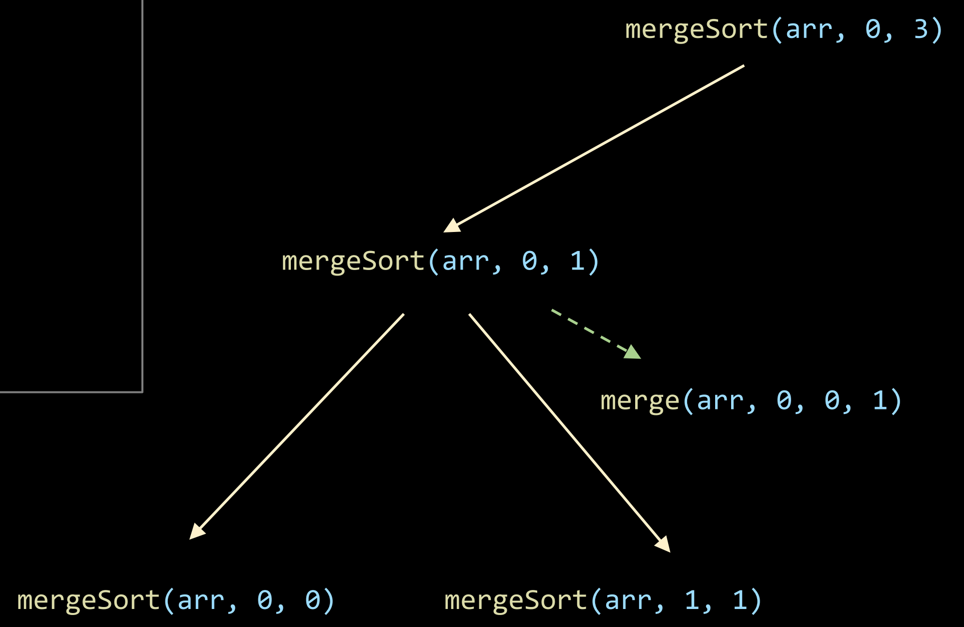
Merge Sort Code

arr

15	2	7	0
0	1	2	3

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```



arr

2	15	7	0
0	1	2	3

https://onlinegdb.com/HJT6FfMlv

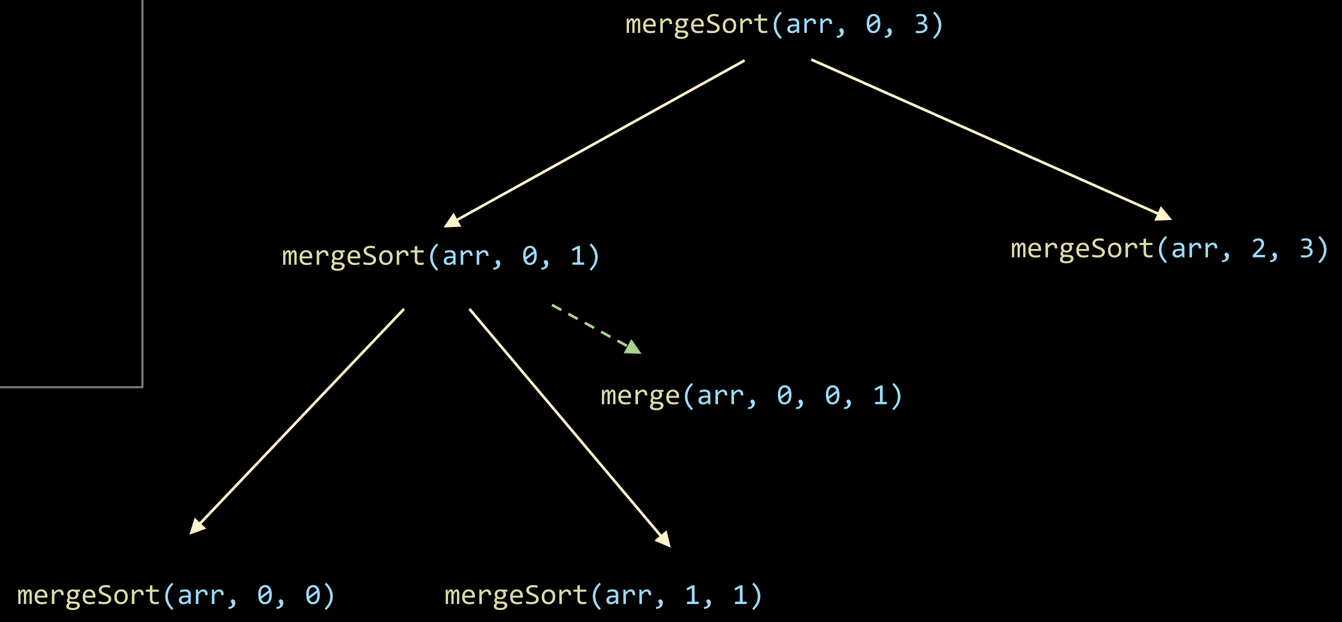
Merge Sort Code

arr

15	2	7	0
0	1	2	3

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```



arr

2	15	7	0
0	1	2	3

https://onlinegdb.com/HJT6FfMlv

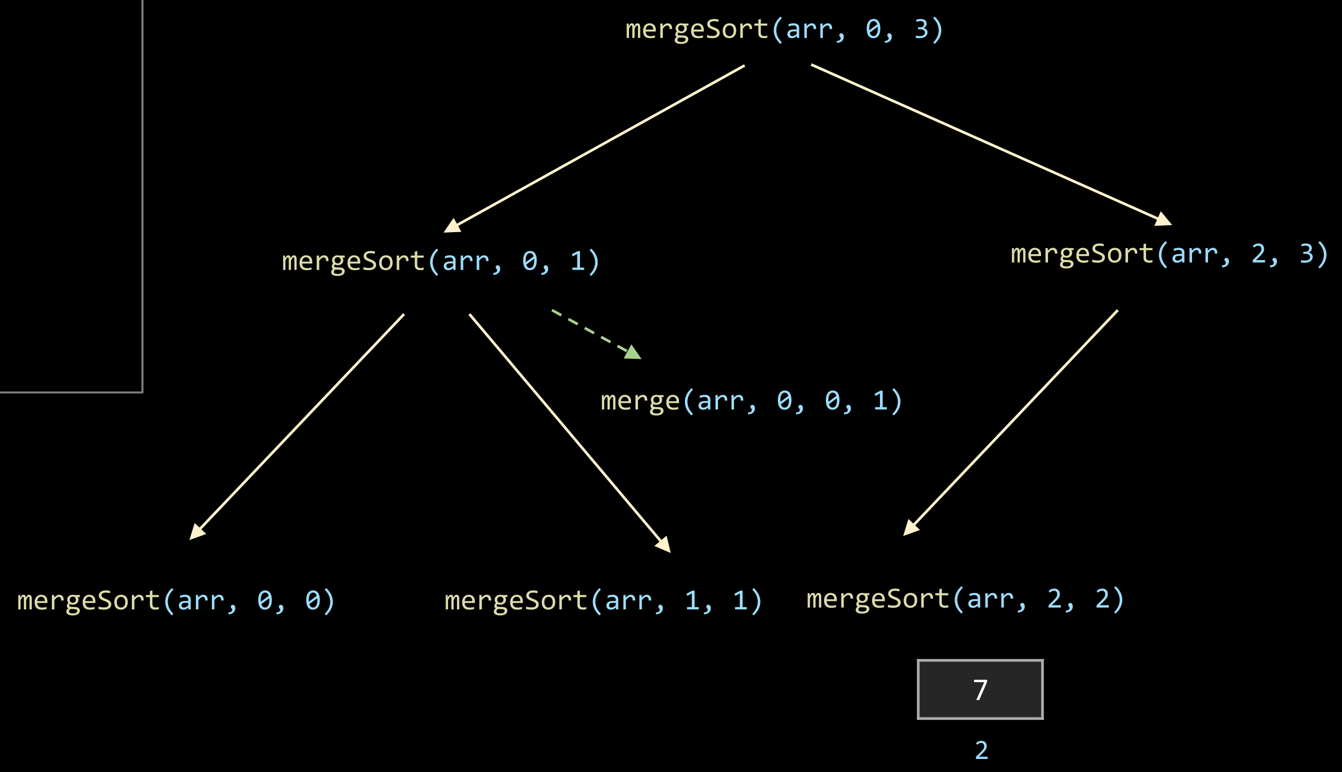
Merge Sort Code

arr

15	2	7	0
0	1	2	3

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```



arr

2	15	7	0
0	1	2	3

https://onlinegdb.com/HJT6FfMlv

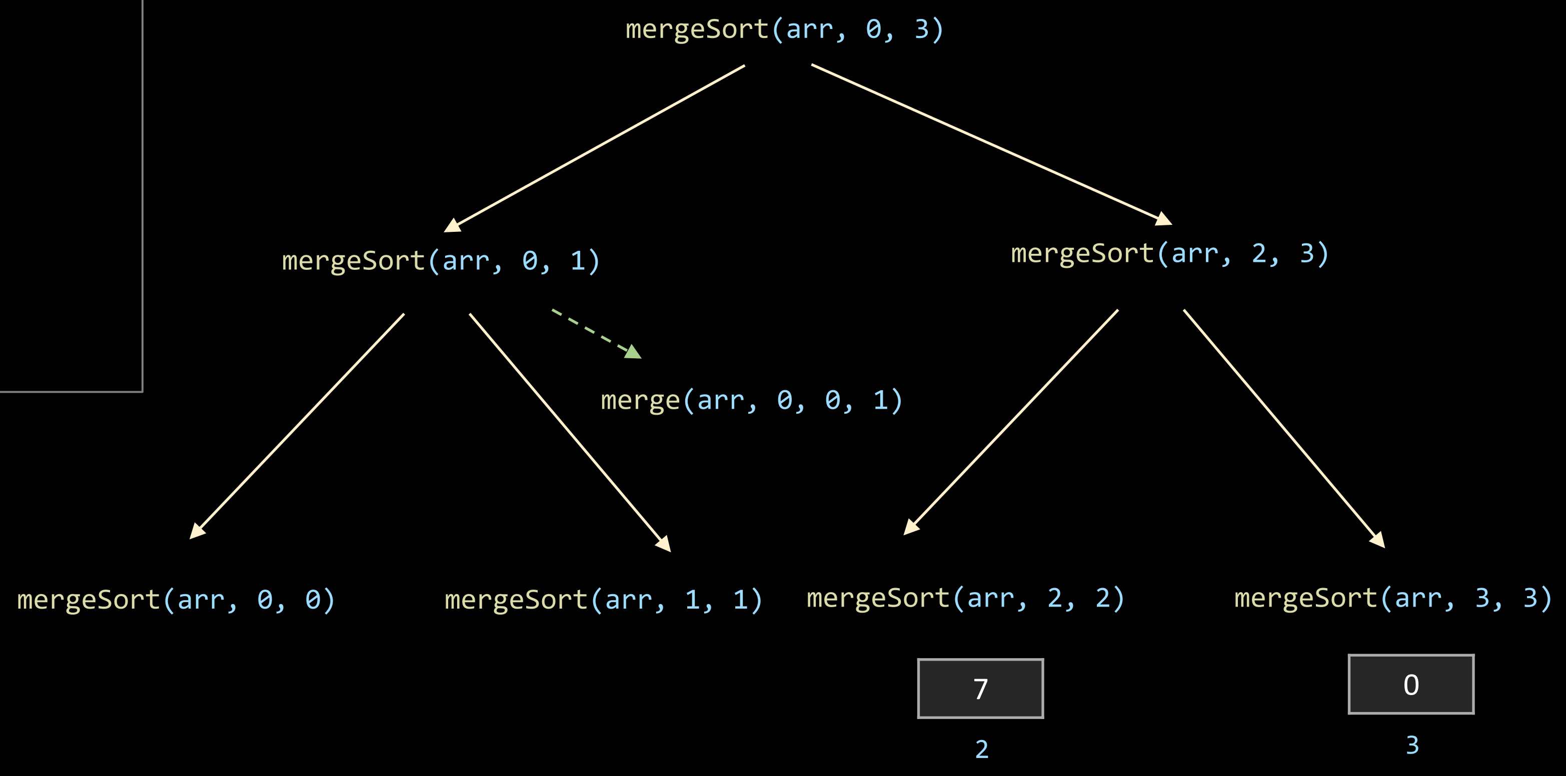
Merge Sort Code

arr

15	2	7	0
0	1	2	3

```c
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```



arr

2	15	7	0
0	1	2	3

https://onlinegdb.com/HJT6FfMlv

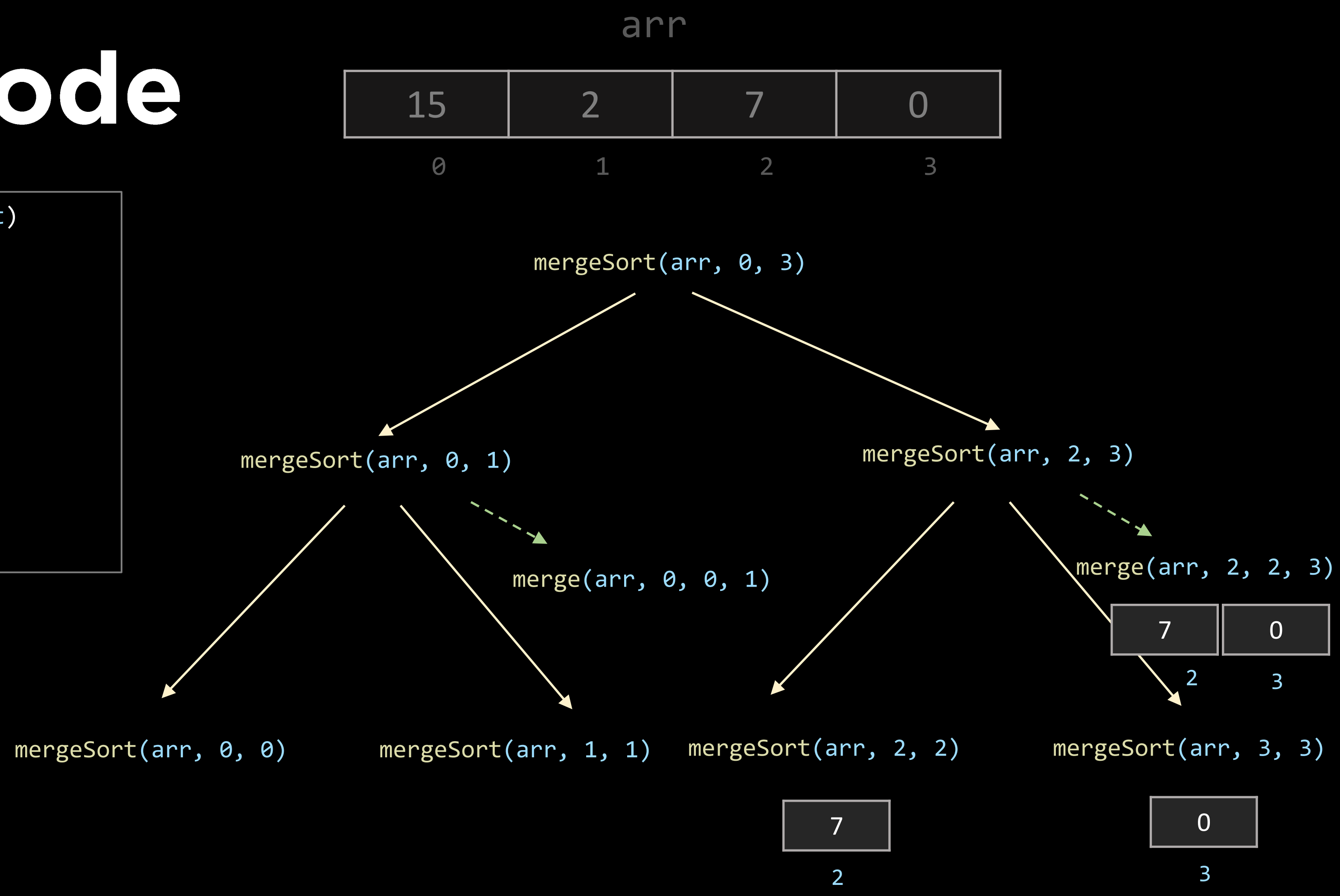
Merge Sort Code

```c
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```

arr

2	15	7	0
0	1	2	3

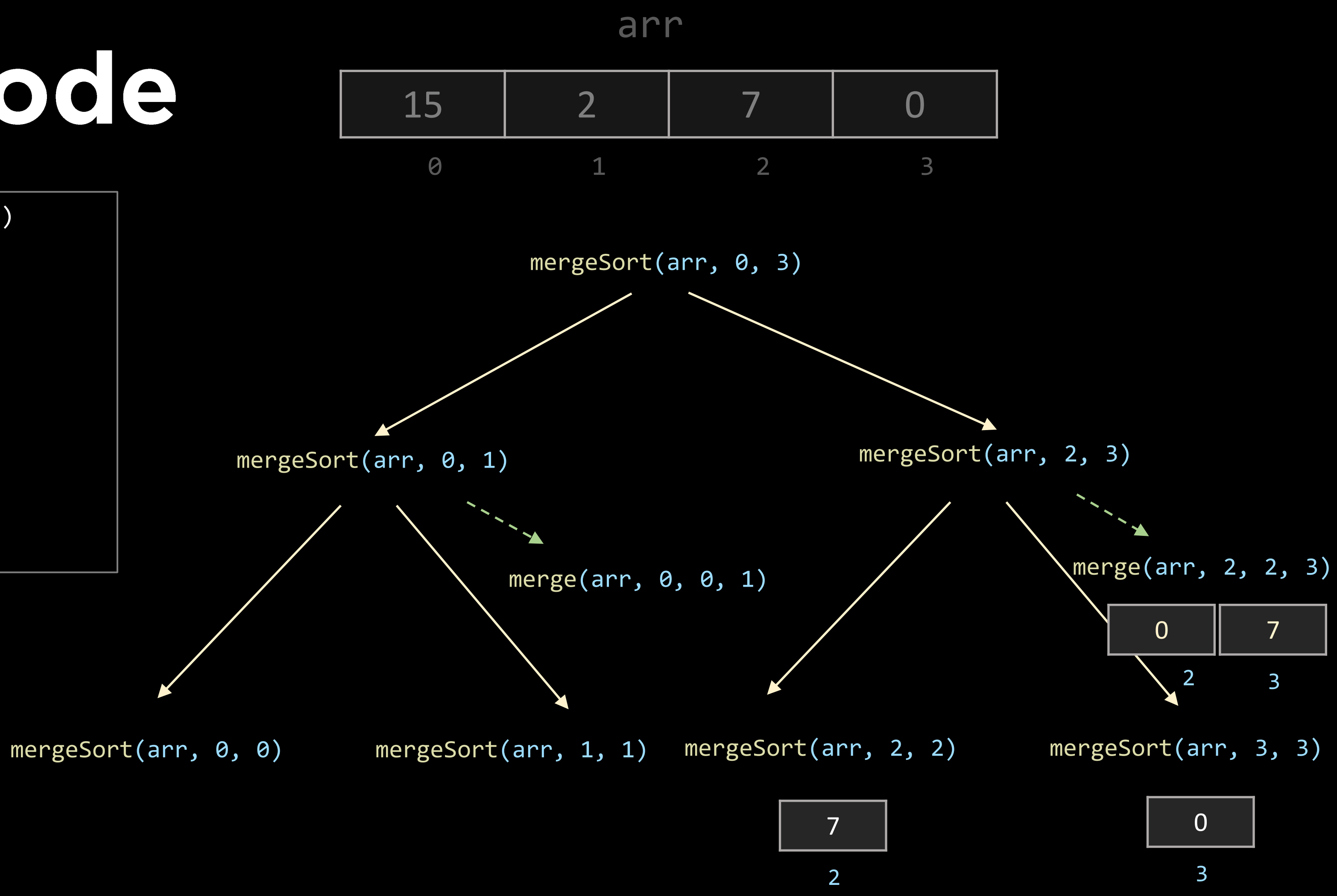


https://onlinegdb.com/HJT6FfMlv

Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```



arr

2	15	0	7
0	1	2	3

https://onlinegdb.com/HJT6FfMlv

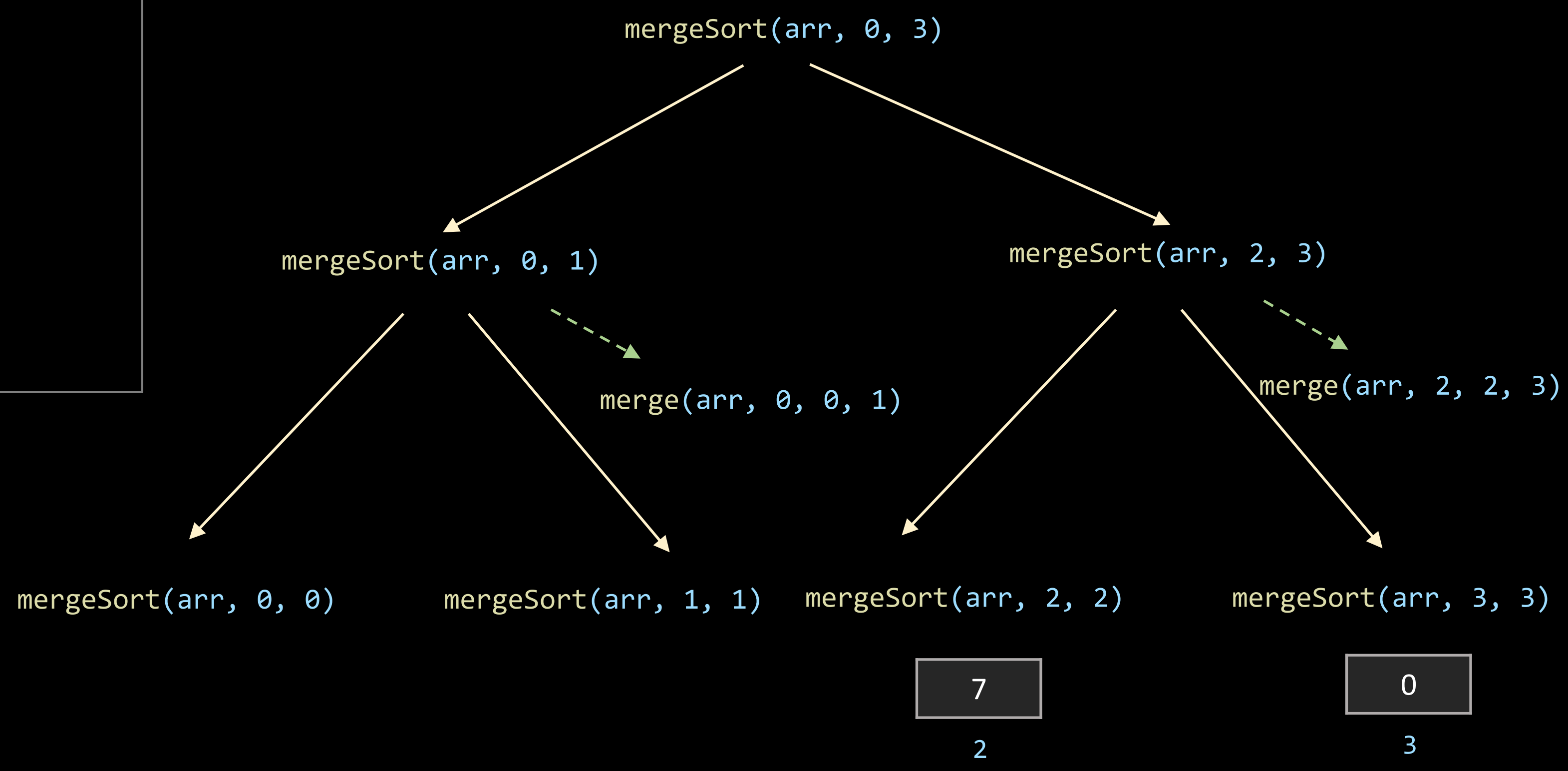
Merge Sort Code

arr

15	2	7	0
0	1	2	3

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```



arr

2	15	0	7
0	1	2	3

https://onlinegdb.com/HJT6FfMlv

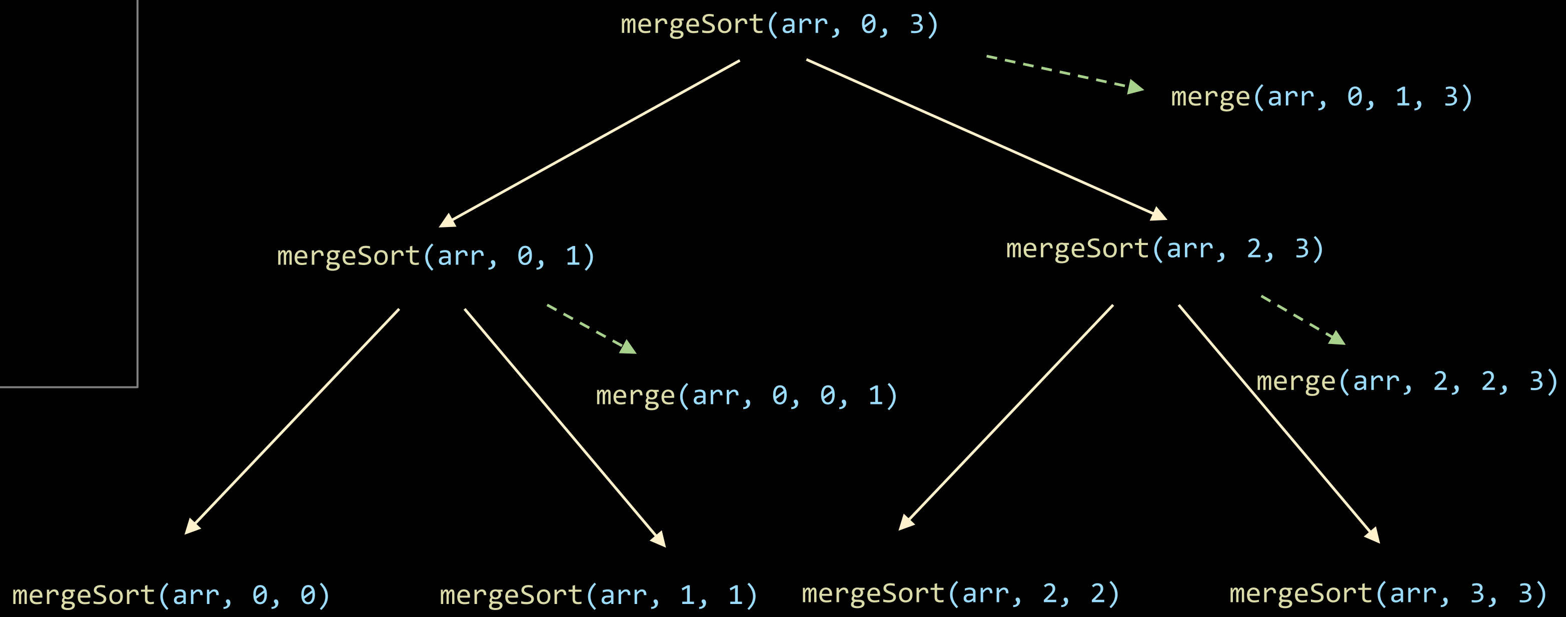
Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```

arr

15	2	7	0
0	1	2	3



arr

2	15	0	7
0	1	2	3

https://onlinegdb.com/HJT6FfMlv

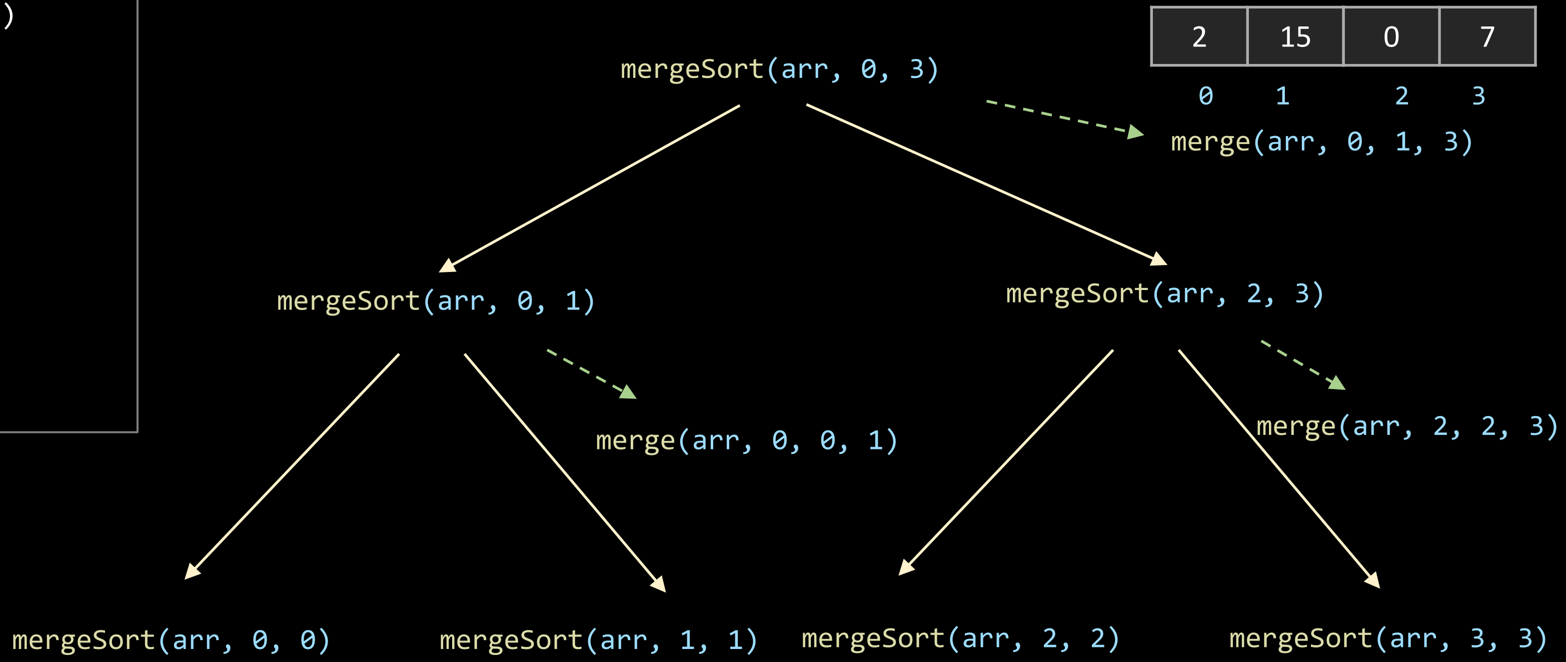
Merge Sort Code

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```

arr

15	2	7	0
0	1	2	3



arr

2	15	0	7
0	1	2	3

https://onlinegdb.com/HJT6FfMlv

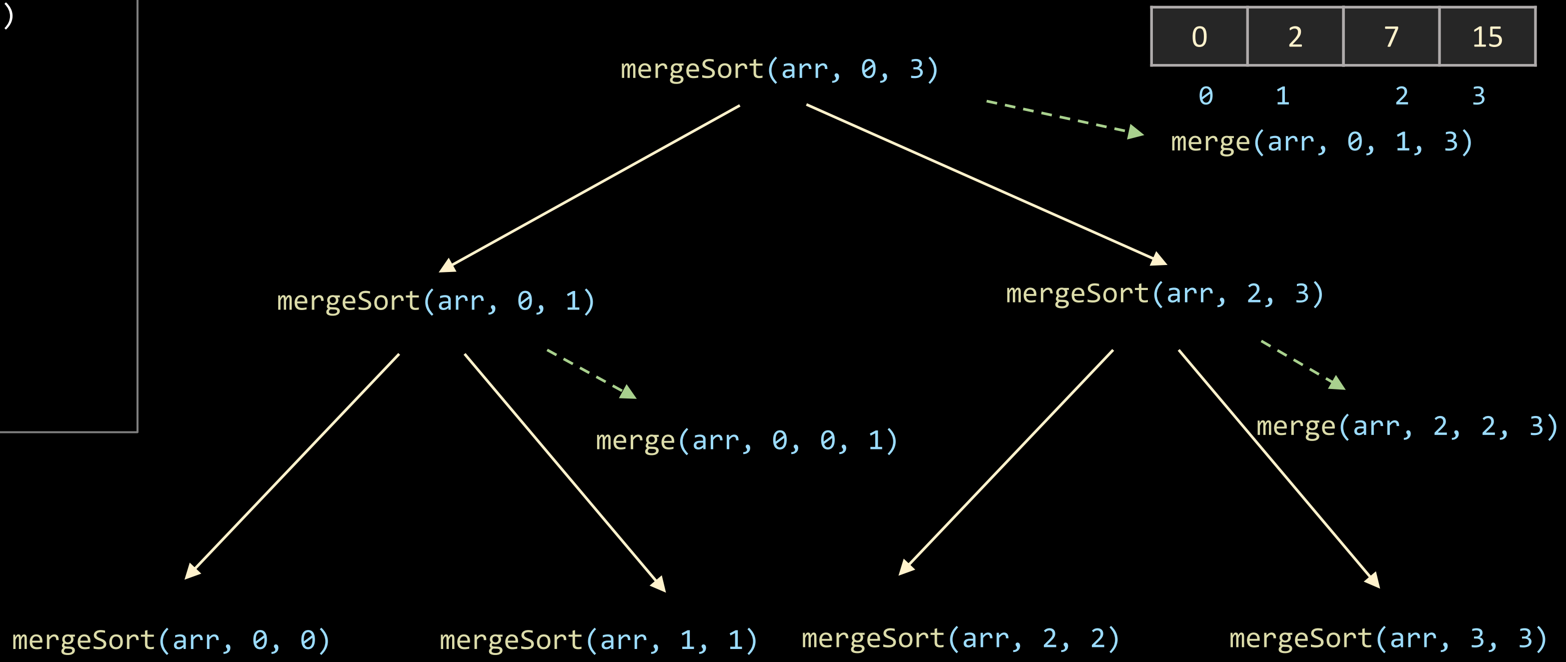
Merge Sort Code

arr

15	2	7	0
0	1	2	3

```
void mergeSort(int arr[], int left, int right)
{
  if (left < right)
  {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    // Merge the sorted subarrays
    merge(arr, left, mid, right);
  }
}
```



arr

0	2	7	15
0	1	2	3

https://onlinegdb.com/HJT6FfMlv

Merge Sort Time Complexity

	Merge Sort
Worst Case	O(n*log n)
Average Case	O(n*log n)
Best Case	O(n*log n)
Space	O(n)

Quick Sort

Quick Sort

Premise:

- Quicksort rearranges the array into two parts – called partitioning
- A pivot is selected, and the following is executed:
    - All the elements in the left subarray are less than or equal to the pivot
    - All the elements in the right subarray are larger than the pivot

:

- The process is repeated until the array is sorted

Quick Sort

Algorithm for Quicksort

if first < last then

- Partition the elements in the subarray first . . . last so that the pivot value is in its correct place (subscript pivIndex)
- Recursively apply quicksort to the subarray first . . . pivIndex - 1
- Recursively apply quicksort to the subarray pivIndex + 1 . . . last

7	4	9	3	2	8	6	5

Quick Sort

Example:

Initial array

7	4	9	3	2	8	6	5

Algorithm for partition Method

1. Define the pivot value as the contents of `table[first]`.
2. Initialize `up` to `first` and `down` to `last`.
3. do
4. Increment `up` until `up` selects the first element greater than the pivot value or `up` has reached `last`.
5. Decrement `down` until `down` selects the first element less than or equal to the pivot value or `down` has reached `first`.
6. `if up < down` then
7. Exchange `table[up]` and `table[down]`.
8. `while up` is to the left of `down`
9. Exchange `table[first]` and `table[down]`.
10. Return the value of `down` to `pivIndex`.

Quick Sort

Example:

Up — Down

Initial array

7	4	9	3	2	8	6	5

Pivot

Algorithm for `partition` Method

1. Define the pivot value as the contents of `table[first]`.
2. Initialize `up` to `first` and `down` to `last`.
3. `do`
4. Increment `up` until `up` selects the first element greater than the pivot value or `up` has reached `last`.
5. Decrement `down` until `down` selects the first element less than or equal to the pivot value or `down` has reached `first`.
6. `if up < down then`
7. Exchange `table[up]` and `table[down]`.
8. `while up` is to the left of `down`
9. Exchange `table[first]` and `table[down]`.
10. Return the value of `down` to `pivIndex`.

Quick Sort

Example:

Up Down

Initial array

7	4	9	3	2	8	6	5

Pivot

Algorithm for partition Method

1. Define the pivot value as the contents of `table[first]`.
2. Initialize `up` to `first` and `down` to `last`.
3. do
4. Increment `up` until `up` selects the first element greater than the pivot value or `up` has reached `last`.
5. Decrement `down` until `down` selects the first element less than or equal to the pivot value or `down` has reached `first`.
6. `if up < down then`
7. Exchange `table[up]` and `table[down]`.
8. `while up` is to the left of `down`
9. Exchange `table[first]` and `table[down]`.
10. Return the value of `down` to `pivIndex`.

Quick Sort

Example:

Up — Down

Initial array

7	4	5	3	2	8	6	9

Pivot

Algorithm for partition Method

1. Define the pivot value as the contents of `table[first]`.
2. Initialize `up` to `first` and `down` to `last`.
3. `do`
4. Increment `up` until `up` selects the first element greater than the pivot value or `up` has reached `last`.
5. Decrement `down` until `down` selects the first element less than or equal to the pivot value or `down` has reached `first`.
6. `if up < down then`
7. Exchange `table[up]` and `table[down]`.
8. `while up` is to the left of `down`
9. Exchange `table[first]` and `table[down]`.
10. Return the value of `down` to `pivIndex`.

Quick Sort

Example:

Up Down

Initial array

7	4	5	3	2	8	6	9

Pivot

Algorithm for partition Method

1. Define the pivot value as the contents of `table[first]`.
2. Initialize `up` to `first` and `down` to `last`.
3. `do`
4. Increment `up` until `up` selects the first element greater than the pivot value or `up` has reached `last`.
5. Decrement `down` until `down` selects the first element less than or equal to the pivot value or `down` has reached `first`.
6. `if up < down then`
7. Exchange `table[up]` and `table[down]`.
8. `while up` is to the left of `down`
9. Exchange `table[first]` and `table[down]`.
10. Return the value of `down` to `pivIndex`.

Quick Sort

Example:

Up Down

Initial array

7	4	5	3	2	6	8	9

Pivot

Algorithm for partition Method

1. Define the pivot value as the contents of `table[first]`.
2. Initialize `up` to `first` and `down` to `last`.
3. `do`
4. Increment `up` until `up` selects the first element greater than the pivot value or `up` has reached `last`.
5. Decrement `down` until `down` selects the first element less than or equal to the pivot value or `down` has reached `first`.
6. `if up < down then`
7. Exchange `table[up]` and `table[down]`.
8. `while up` is to the left of `down`
9. Exchange `table[first]` and `table[down]`.
10. Return the value of `down` to `pivIndex`.

Quick Sort

Example:

					Down	Up	
7	4	5	3	2	6	8	9
Pivot							

Initial array

Algorithm for partition Method

1. Define the pivot value as the contents of `table[first]`.
2. Initialize `up` to `first` and `down` to `last`.
3. `do`
4. Increment `up` until `up` selects the first element greater than the pivot value or `up` has reached `last`.
5. Decrement `down` until `down` selects the first element less than or equal to the pivot value or `down` has reached `first`.
6. `if up < down then`
7. Exchange `table[up]` and `table[down]`.
8. `while up` is to the left of `down`
9. Exchange `table[first]` and `table[down]`.
10. Return the value of `down` to `pivIndex`.

Quick Sort

Example:

Down Up

Initial array

6	4	5	3	2	7	8	9

Pivot

Algorithm for partition Method

1. Define the pivot value as the contents of `table[first]`.
2. Initialize `up` to `first` and `down` to `last`.
3. do
4. Increment `up` until `up` selects the first element greater than the pivot value or `up` has reached `last`.
5. Decrement `down` until `down` selects the first element less than or equal to the pivot value or `down` has reached `first`.
6. if `up` < `down` then
7. Exchange `table[up]` and `table[down]`.
8. `while` `up` is to the left of `down`
9. Exchange `table[first]` and `table[down]`.
10. Return the value of `down` to `pivIndex`.

Quick Sort

Example:

Down Up

Initial array	6	4	5	3	2	7	8	9

Pivot

Algorithm for partition Method

1. Define the pivot value as the contents of `table[first]`.
2. Initialize `up` to `first` and `down` to `last`.
3. `do`
4. Increment `up` until `up` selects the first element greater than the pivot value or `up` has reached `last`.
5. Decrement `down` until `down` selects the first element less than or equal to the pivot value or `down` has reached `first`.
6. `if up < down` then
7. Exchange `table[up]` and `table[down]`.
8. `while up` is to the left of `down`
9. Exchange `table[first]` and `table[down]`.
10. Return the value of `down` to `pivIndex`.

Quick Sort Code

```
void quickSort(int array[], int low, int high)
{
  if (low < high)
  {
    int pivot = partition(array, low, high);
    quickSort(array, low, pivot - 1);
    quickSort(array, pivot + 1, high);
  }
}
```

```
int partition(int array[], int low, int high)
{
  // Select the pivot element
  int pivot = array[low];
  int up = low, down = high;

  while(up < down)
  {
      for (int j = up; j < high; j++)
      {
        if(array[up] > pivot)
            break;
        up++;
      }
      for (int j = high; j > low; j--)
      {
        if(array[down] < pivot)
            break;
        down--;
      }
      if(up < down)
        swap(&array[up], &array[down]);
  }
  swap(&array[low], &array[down]);
  return down;
}
```

```
int main()
{
  int data[] = {15, 0, 5, 6};
  int n = sizeof(data) / sizeof(data[0]);
  quickSort(data, 0, n - 1);
  return 0;
}
```

https://onlinegdb.com/ryRlgsGgD

Quick Sort

Time Complexity:

- If the pivot value is a random value selected from the current subarray,
  - then statistically half of the items in the subarray will be less than the pivot and half will be greater
  - thus there will be log n levels of recursion
- Partitioning requires n moves
- Total time: O(n log n) on average
- A quicksort will give very poor behavior if, each time the array is partitioned, a subarray is empty. In that case, the sort will be $O(n^2)$. Under these circumstances, the overhead of recursive calls and the extra run-time stack storage required by these calls makes this version of quicksort a poor performer relative to the quadratic sorts

Quick Sort Time Complexity

	Merge Sort	Quick Sort
Worst Case	O(n*log n)	$O(n^2)$
Average Case	O(n*log n)	O(n*log n)
Best Case	O(n*log n)	O(n*log n)
Space	O(n)	O(log n)*

* Recursion Stack

Other Sorts

- Sleep sort
- Counting sort
- Tim Sort
- Radix Sort
- Bucket Sort

Resources

- https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
- https://www.programiz.com/dsa
- https://www.youtube.com/user/AlgoRythmics/videos
- https://www.toptal.com/developers/sorting-algorithms

Questions

Mentimeter

Menti.com

3914 9376