

# *Topic 9*

## *Advanced Sorting*

CSE1303 Part A

Data Structures and Algorithms

# *Overview*

- Divide and Conquer
- Merge Sort
- Quick Sort

# *Divide and Conquer*

Recall: **Binary Search**

Search



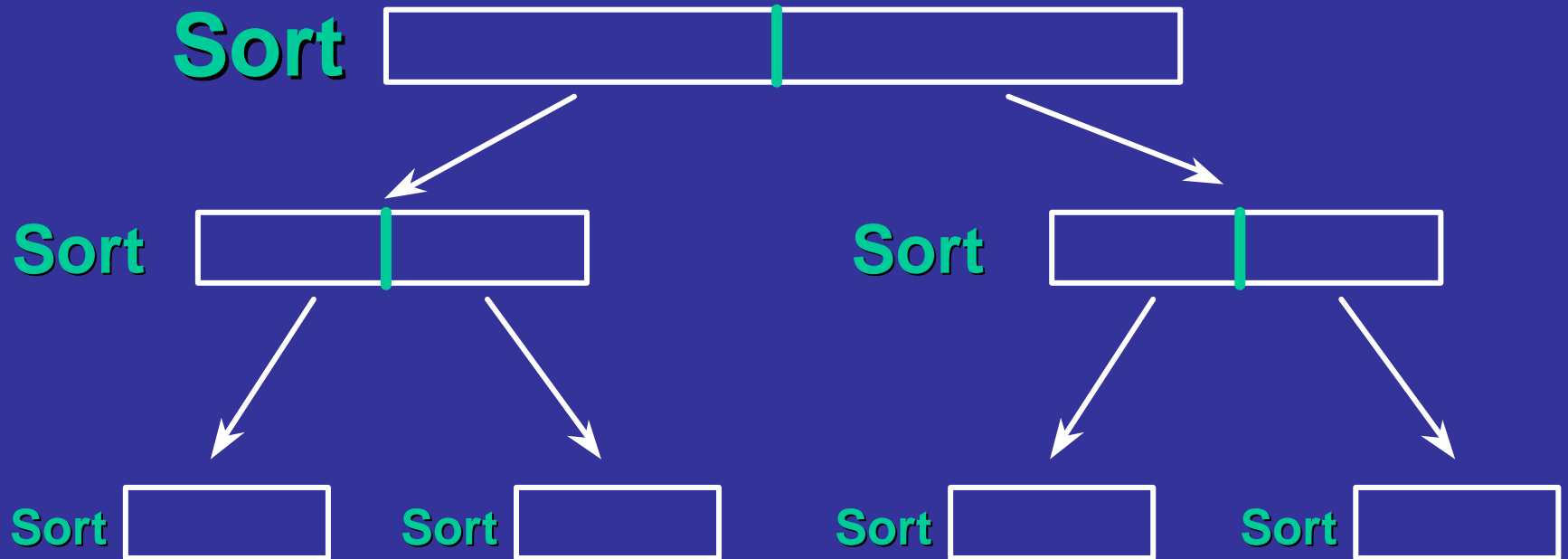
Search



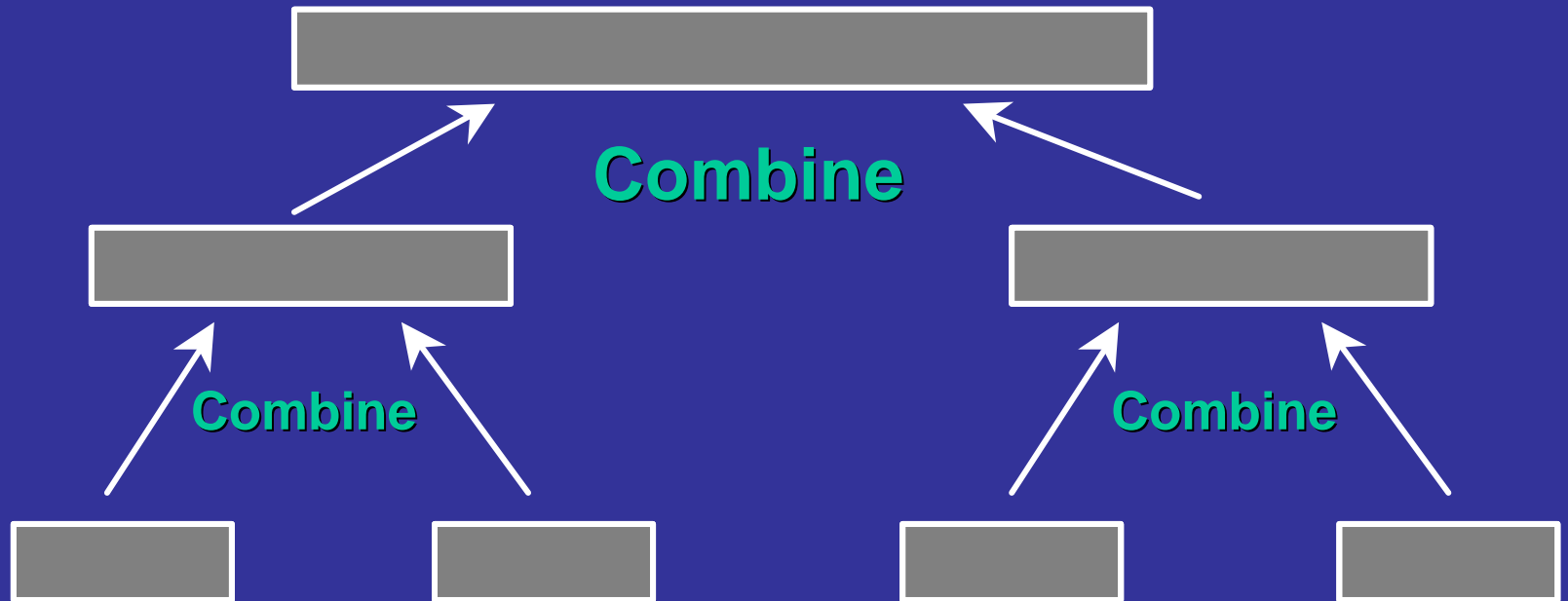
Search



# *Divide and Conquer*



# *Divide and Conquer*



# *Divide and Conquer*

```
module sort(array)
{
  if (size of array > 1)
  {
    split(array, firstPart, secondPart)
    sort(firstPart)
    sort(secondPart)
    combine(firstPart, secondPart)
  }
}
```

# *Divide and Conquer*

```
module sort(array)
{
  if (size of array > 1)
  {
    split(array, firstPart, secondPart)
    sort(firstPart)
    sort(secondPart)
    combine(firstPart, secondPart)
  }
}
```

# *Divide and Conquer*

```
module sort(array)
{
  if (size of array > 1)
  {
    split(array, firstPart, secondPart)
    sort(firstPart)
    sort(secondPart)
    combine(firstPart, secondPart)
  }
}
```

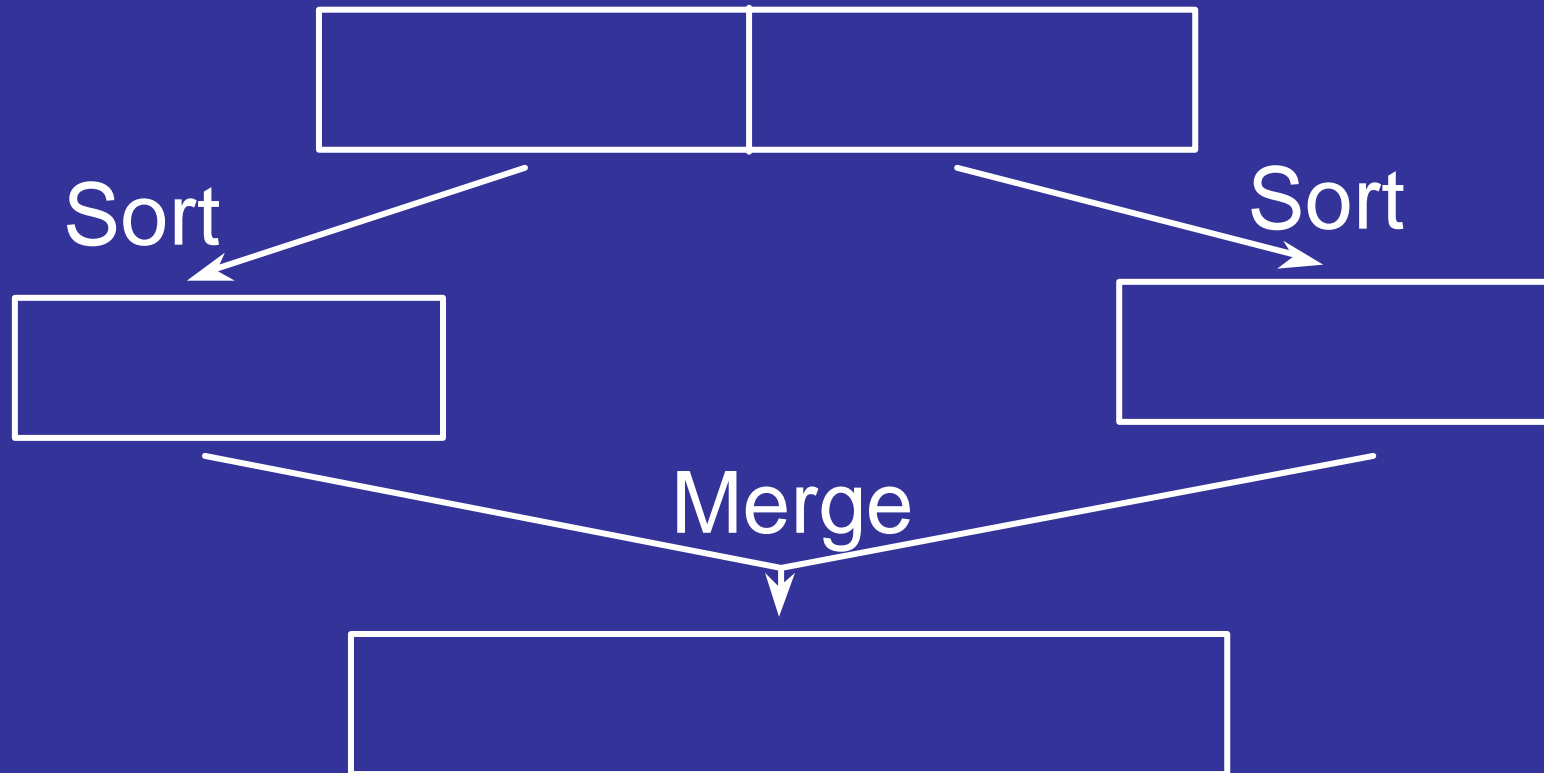
# *Divide and Conquer*

```
module sort(array)
{
  if (size of array > 1)
  {
    split(array, firstPart, secondPart)
    sort(firstPart)
    sort(secondPart)
    combine(firstPart, secondPart)
  }
}
```

# *Divide and Conquer*

```
module sort(array)
{
  if (size of array > 1)
  {
    split(array, firstPart, secondPart)
    sort(firstPart)
    sort(secondPart)
    combine(firstPart, secondPart)
  }
}
```

# *Merge Sort*



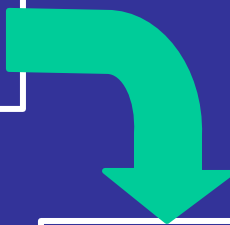
# Merge Sort

array:



size

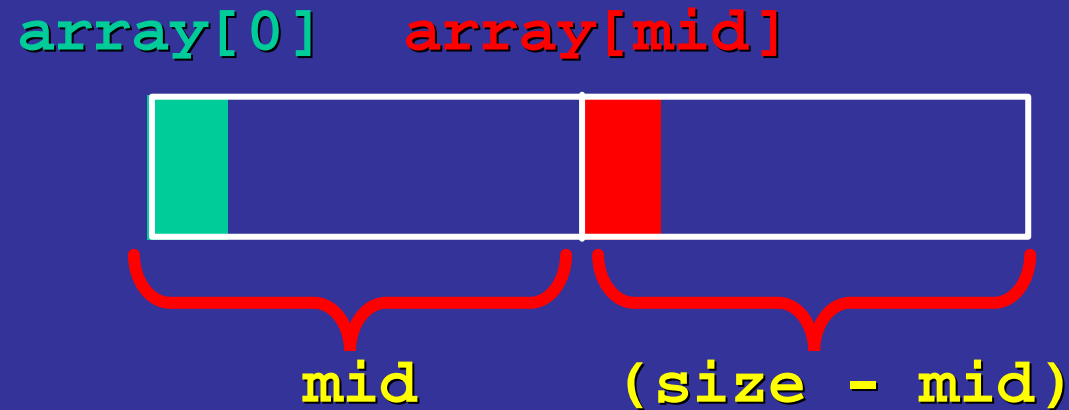
tmpArrayPtr:



tmp:



# Merge Sort



`firstPart : array`

`secondPart : array + mid`

# Merge Sort

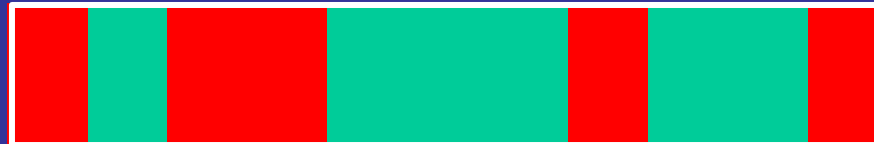
array:



mergeList



tmp:



# *Merge Sort*

array:



tmp:



# *Merge Sort*

array:



```
void mergeSort(float array[], int size)
{
    int* tmpArrayPtr = (int*)malloc(size*sizeof(int));

    if (tmpArrayPtr != NULL)
    {
        mergeSortRec(array, size, tmpArrayPtr);
    }
    else
    {
        fprintf(stderr, "Not enough memory to sort list.\n");
        exit(1);
    }

    free(tmpArrayPtr);
}
```

```
void mergeSortRec(float array[], int size, float tmp[])
{
    int    i;
    int    mid = size/2;

    if (size > 1)
    {
        mergeSortRec(array, mid, tmp);
        mergeSortRec(array+mid, size-mid, tmp);

        mergeArrays(array, mid, array+mid, size-mid, tmp);

        for (i = 0; i < size; i++)
        {
            array[i] = tmp[i];
        }
    }
}
```

# Example: mergeArrays

**a:**

3	5	15	28	30
---	---	----	----	----



**aSize: 5**

**b:**

6	10	14	22	43	50
---	----	----	----	----	----



**bSize: 6**

**tmp:**

--	--	--	--	--	--	--	--	--	--	--

# Example: mergeArrays

a: 

3	5	15	28	30
---	---	----	----	----

i=0

b: 

6	10	14	22	43	50
---	----	----	----	----	----

j=0

tmp: 

--	--	--	--	--	--	--	--	--	--	--

k=0

# Example: mergeArrays

a: 

3	5	15	28	30
---	---	----	----	----

i=0

b: 

6	10	14	22	43	50
---	----	----	----	----	----

j=0

tmp: 

3										
---	--	--	--	--	--	--	--	--	--	--

k=0

# Example: mergeArrays

a: 

3	5	15	28	30
---	---	----	----	----

i=1

b: 

6	10	14	22	43	50
---	----	----	----	----	----

j=0

tmp: 

3	5								
---	---	--	--	--	--	--	--	--	--

k=1

# Example: mergeArrays

a: 

3	5	15	28	30
---	---	----	----	----

i=2

b: 

6	10	14	22	43	50
---	----	----	----	----	----

j=0

tmp: 

3	5	6							
---	---	---	--	--	--	--	--	--	--

k=2

# Example: mergeArrays

a: 

3	5	15	28	30
---	---	----	----	----

i=2

b: 

6	10	14	22	43	50
---	----	----	----	----	----

j=1

tmp: 

3	5	6	10						
---	---	---	----	--	--	--	--	--	--

k=3

# Example: mergeArrays

**a:**

3	5	15	28	30
---	---	----	----	----

**i=2**

**b:**

6	10	14	22	43	50
---	----	----	----	----	----

**j=2**

**tmp:**

3	5	6	10	14						
---	---	---	----	----	--	--	--	--	--	--

**k=4**

# Example: mergeArrays

a: 

3	5	15	28	30
---	---	----	----	----

i=2

b: 

6	10	14	22	43	50
---	----	----	----	----	----

j=3

tmp: 

3	5	6	10	14	15					
---	---	---	----	----	----	--	--	--	--	--

k=5

# Example: mergeArrays

a: 

3	5	15	28	30
---	---	----	----	----

i=3

b: 

6	10	14	22	43	50
---	----	----	----	----	----

j=3

tmp: 

3	5	6	10	14	15	22				
---	---	---	----	----	----	----	--	--	--	--

k=6

# Example: mergeArrays

**a:**

3	5	15	28	30
---	---	----	----	----

**i=3**

**b:**

6	10	14	22	43	50
---	----	----	----	----	----

**j=4**

**tmp:**

3	5	6	10	14	15	22	28			
---	---	---	----	----	----	----	----	--	--	--

**k=7**

# Example: mergeArrays

a: 

3	5	15	28	30
---	---	----	----	----

i=4

b: 

6	10	14	22	43	50
---	----	----	----	----	----

j=4

tmp: 

3	5	6	10	14	15	22	28	30		
---	---	---	----	----	----	----	----	----	--	--

k=8

# Example: mergeArrays

a: 

3	5	15	28	30
---	---	----	----	----

b: 

6	10	14	22	43	50
---	----	----	----	----	----

i=5

j=4



*Done.*

tmp: 

3	5	6	10	14	15	22	28	30	43	50
---	---	---	----	----	----	----	----	----	----	----

k=9

```

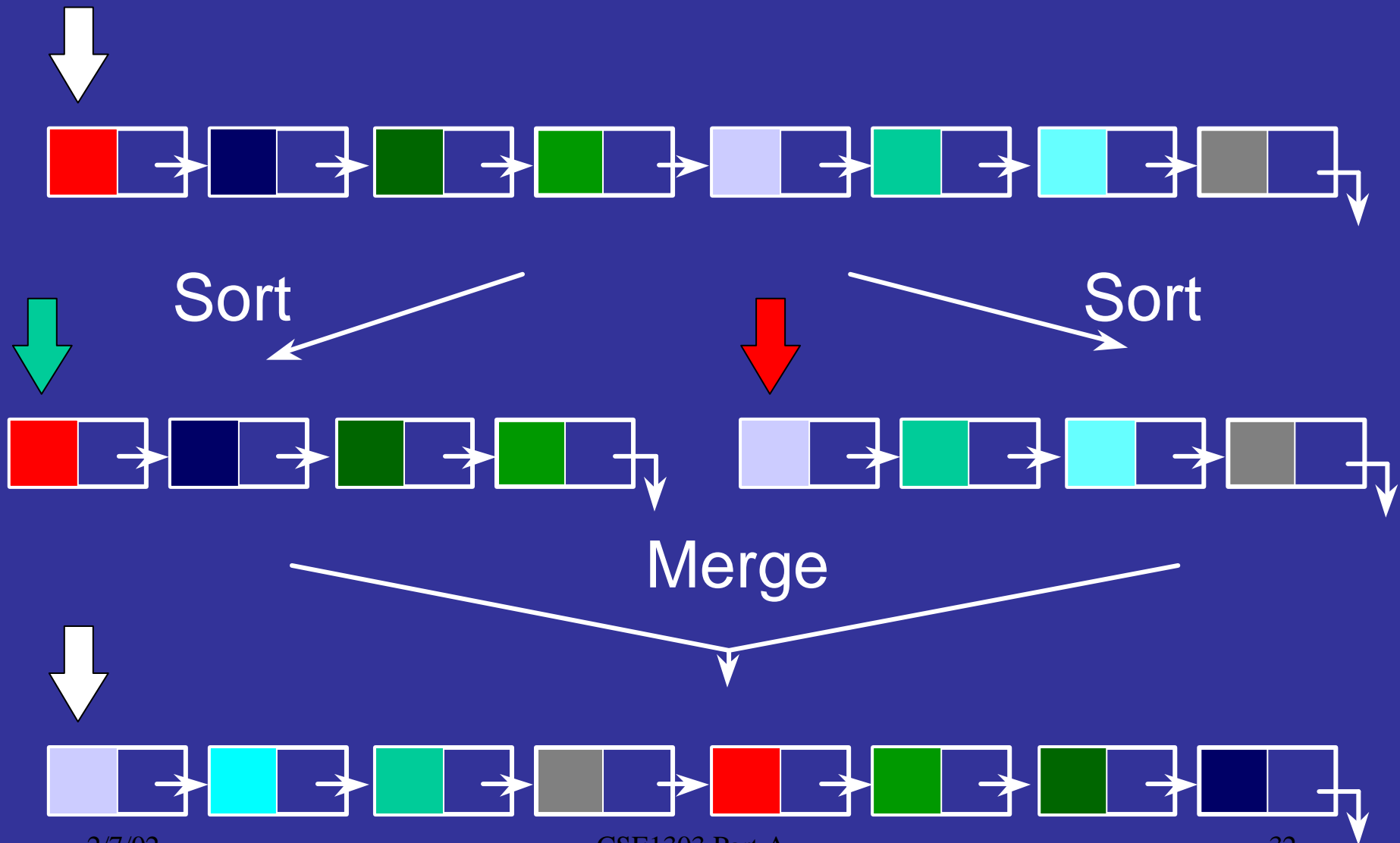
void
mergeArrays(float a[],int aSize,float b[],int bSize,float tmp[])
{
    int    k, i = 0, j = 0;

    for (k = 0; k < aSize + bSize; k++)
    {
        if (i == aSize) {
            tmp[k] = b[j];
            j++;
        }
        else if (j == bSize) {
            tmp[k] = a[i];
            i++;
        }
        else if (a[i] <= b[j]) {
            tmp[k] = a[i];
            i++;
        }
        else {
            tmp[k] = b[j];
            j++;
        }
    }
}

```

2/7/02

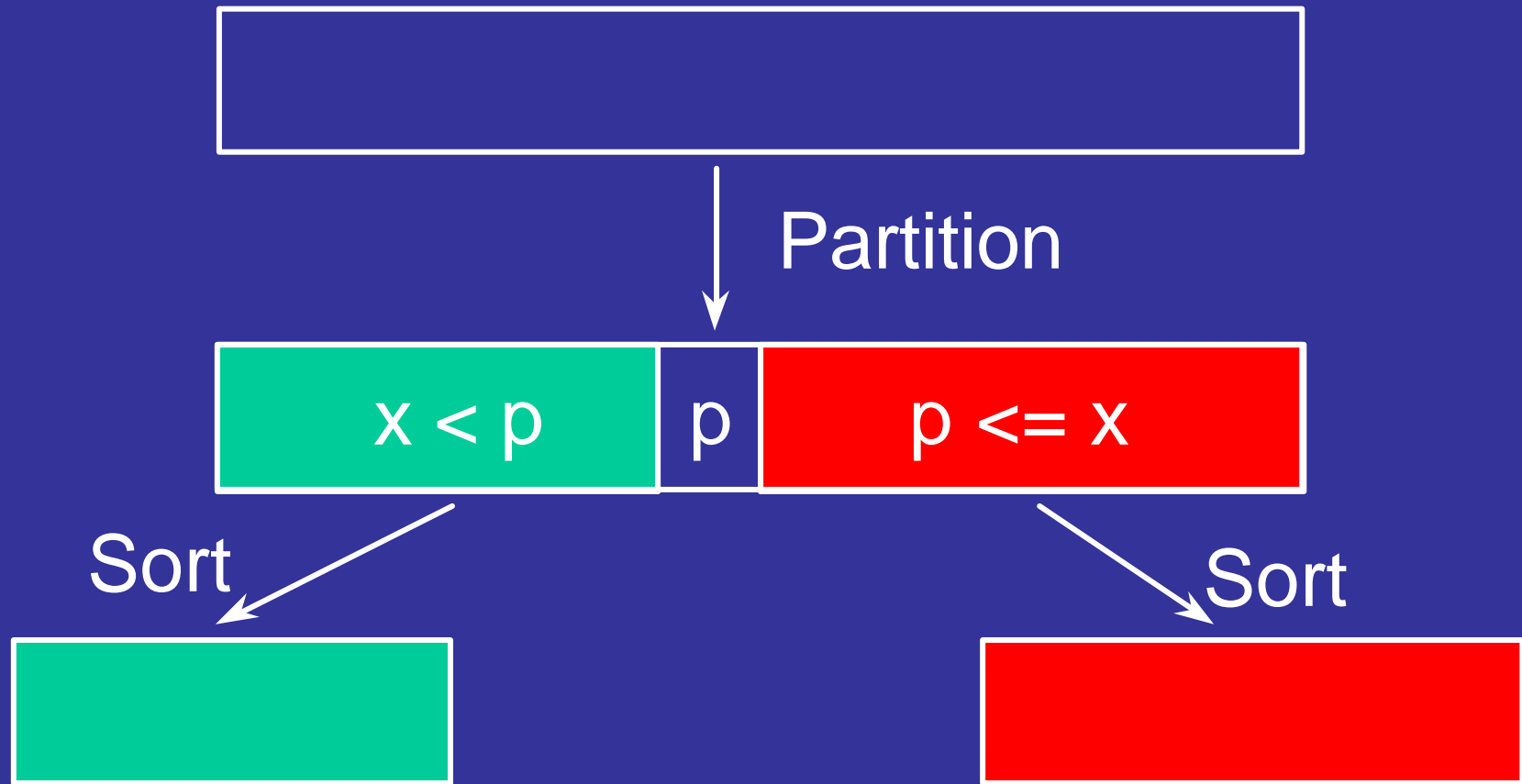
# Merge Sort and Linked Lists



# *Merge Sort Analysis*

- Most of the work is in the merging.
- Takes  **$O(n \log(n))$**
- Uses more space than other sorts.
- Useful for linked lists.

# *Quick Sort*



Example:

# Quick Sort

array:

5	89	35	10	24	15	37	13	20	17	70
---	----	----	----	----	----	----	----	----	----	----



size: 11

Example:

# Quick Sort

array:

5	89	35	14	24	15	37	13	20	7	70
---	----	----	----	----	----	----	----	----	---	----

*“pivot  
element”*

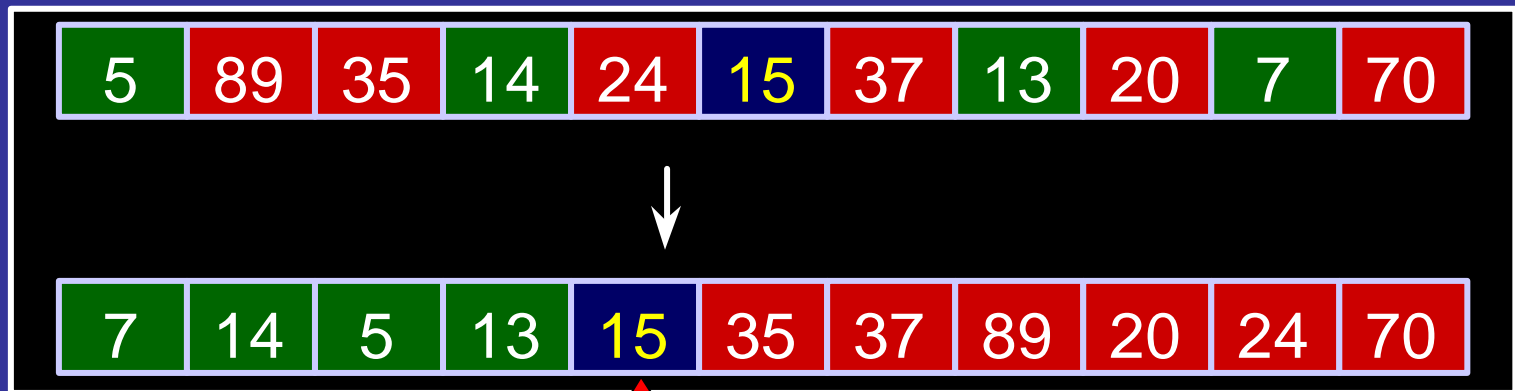
Example:

# Quick Sort

array: 

5	89	35	14	24	15	37	13	20	7	70
---	----	----	----	----	----	----	----	----	---	----

partition:



index: 4

Example:

# Quick Sort

index: 4



array:



index



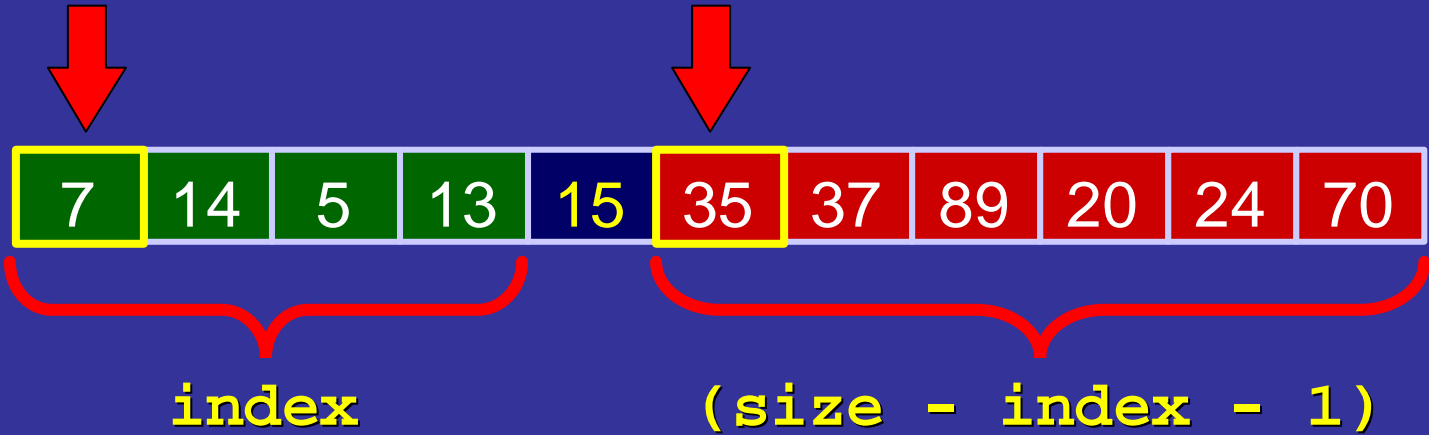
(size - index - 1)

Example:

# Quick Sort

array[0]

array[index + 1]



firstPart : array

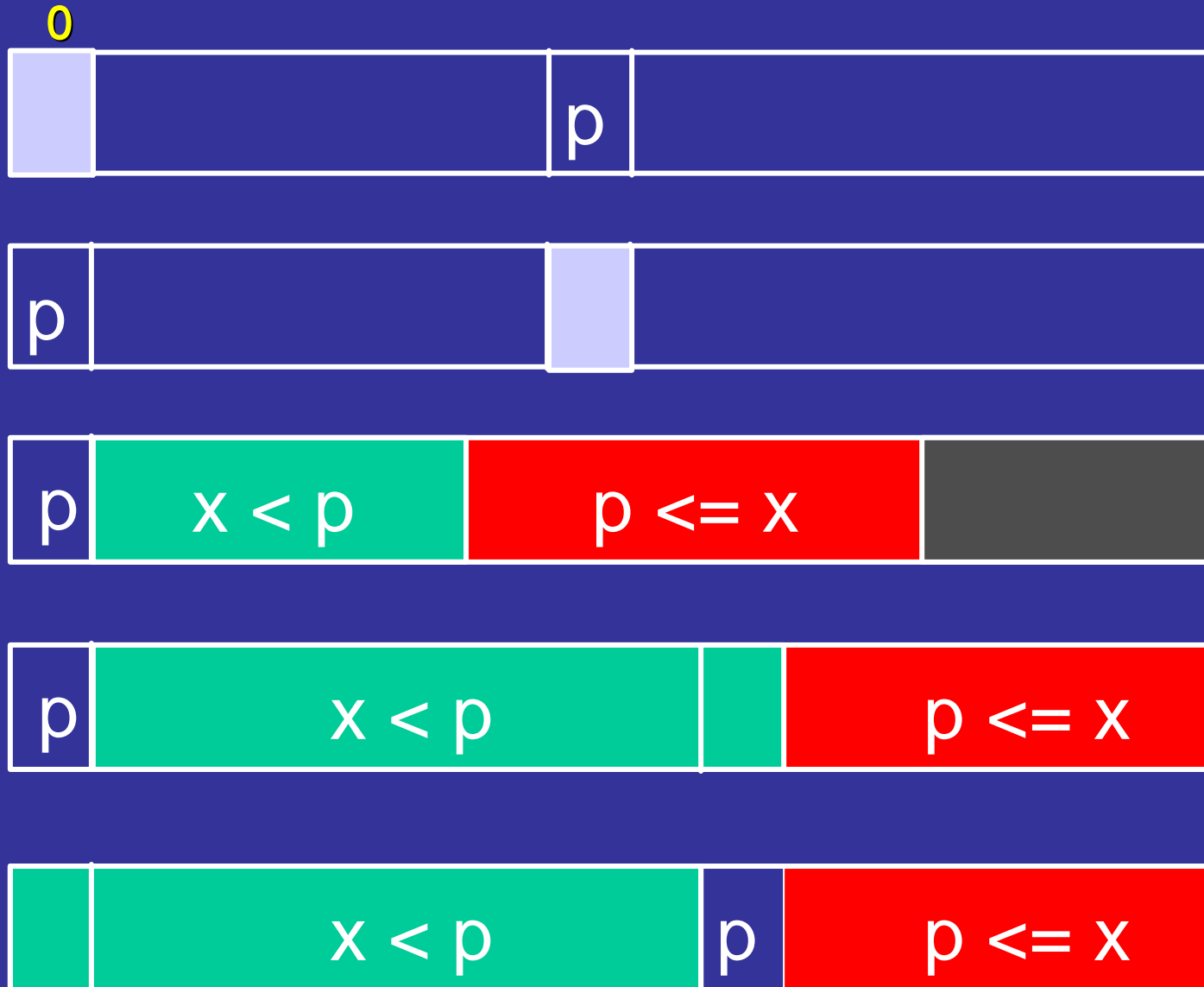
secondPart : array + index + 1

# *Quick Sort*

```
void quickSort(float array[], int size)
{
    int index;

    if (size > 1)
    {
        index = partition(array, size);
        quickSort(array, index);
        quickSort(array+index+1, size - index - 1);
    }
}
```

# Partition: Checklist



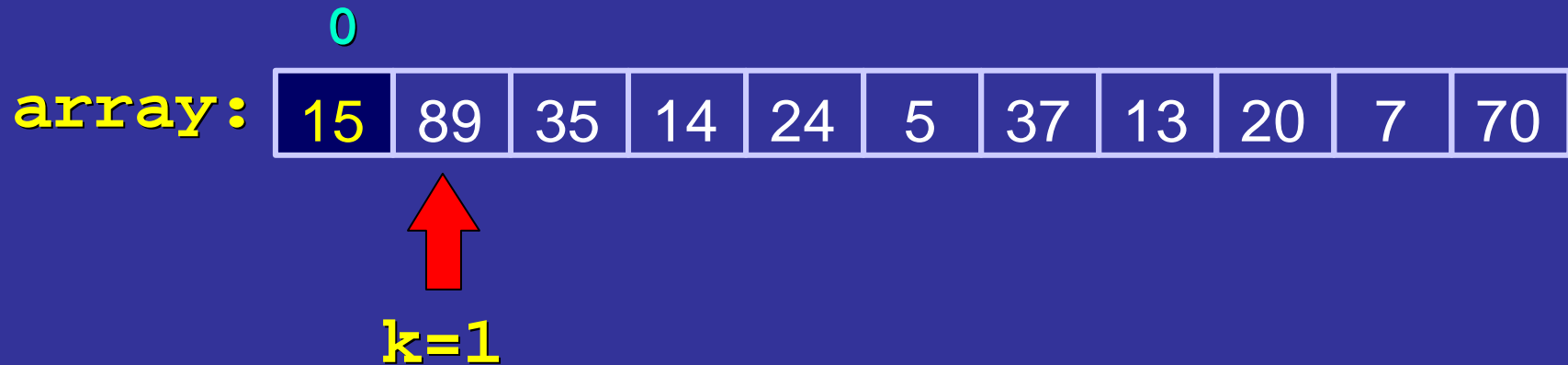
Example:

# Partition



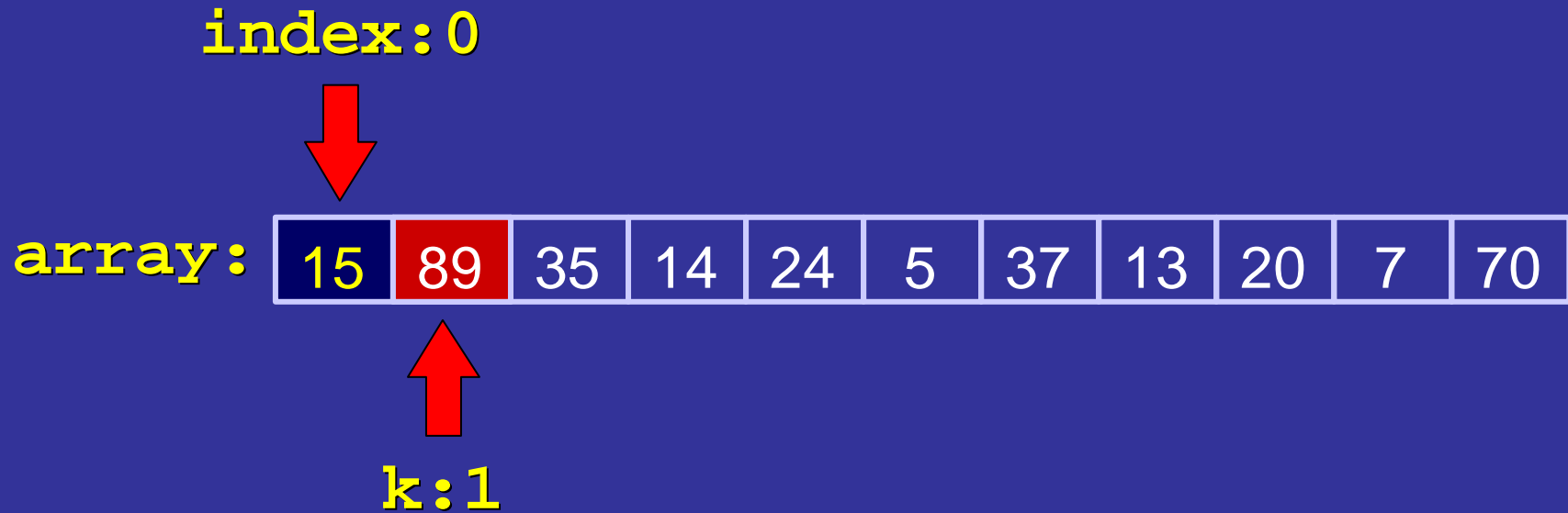
Example:

# Partition



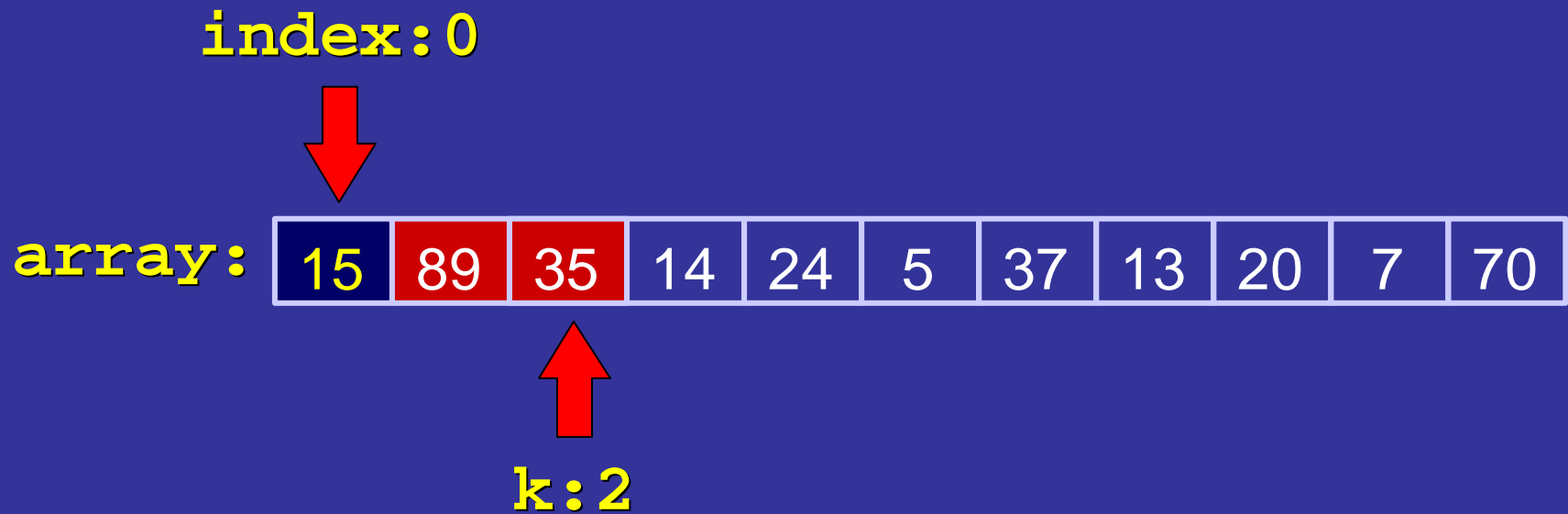
Example:

# Partition



Example:

# Partition



Example:

# Partition

index: 0



array:

15	89	35	14	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----



k: 3

Example:

# Partition

index: 1



array:

15	14	35	89	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----



k: 3

Example:

# Partition

index: 1



array:

15	14	35	89	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----



k: 4

Example:

# Partition

index: 1



array:

15	14	35	89	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----



k: 5

Example:

# Partition

index: 2



array:

15	14	5	89	24	35	37	13	20	7	70
----	----	---	----	----	----	----	----	----	---	----



k: 5

Example:

# Partition

index: 2



array:

15	14	5	89	24	35	37	13	20	7	70
----	----	---	----	----	----	----	----	----	---	----



k: 6

Example:

# Partition

index: 2



array:

15	14	5	89	24	35	37	13	20	7	70
----	----	---	----	----	----	----	----	----	---	----



k: 7

*etc...*

Example:

# Partition

index: 4



array:



k: 11

Example:

# Partition

index: 4



array:

15	14	5	13	7	35	37	89	20	24	70
----	----	---	----	---	----	----	----	----	----	----

Example:

# Partition

index: 4



array:



$x < 15$



$15 \leq x$

## Example:

*pivot now in  
correct position*

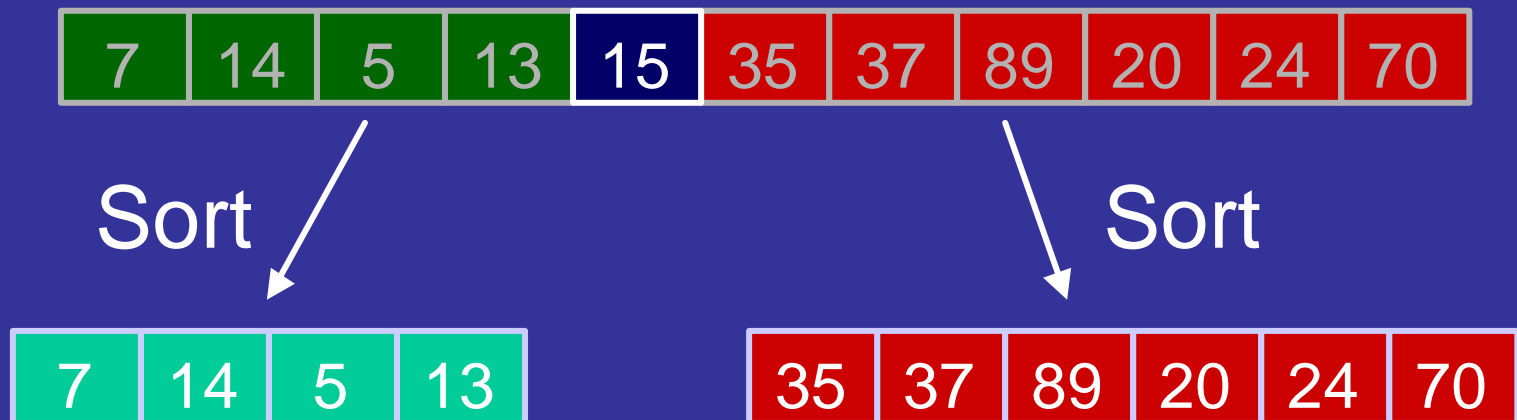
array:



$x < 15$

$15 \leq x$

# Example:



```

int partition(float array[], int size)
{
    int k;
    int mid = size/2;
    int index = 0;

    swap(array, array+mid);

    for (k = 1; k < size; k++)
    {
        if (list[k] < list[0])
        {
            index++;
            swap(array+k, array+index);
        }
    }

    swap(array, array+index);

    return index;
}

```

# *Quick Sort Analysis*

- Most of the work done in partitioning.
- Need to be careful of choice of pivot.
  - *Example: 2, 4, 6, 7, 3, 1, 5*
- Average case takes  **$O(n \log(n))$**  time.
- Worst case takes  **$O(n^2)$**  time.

# *Revision*

- Merge Sort
- Quick Sort