자료구조

L09 Sorting (2)

2022년 1학기

국민대학교 소프트웨어학부

Overview

- **Merge Sort**
- Quicksort

Merge Sort

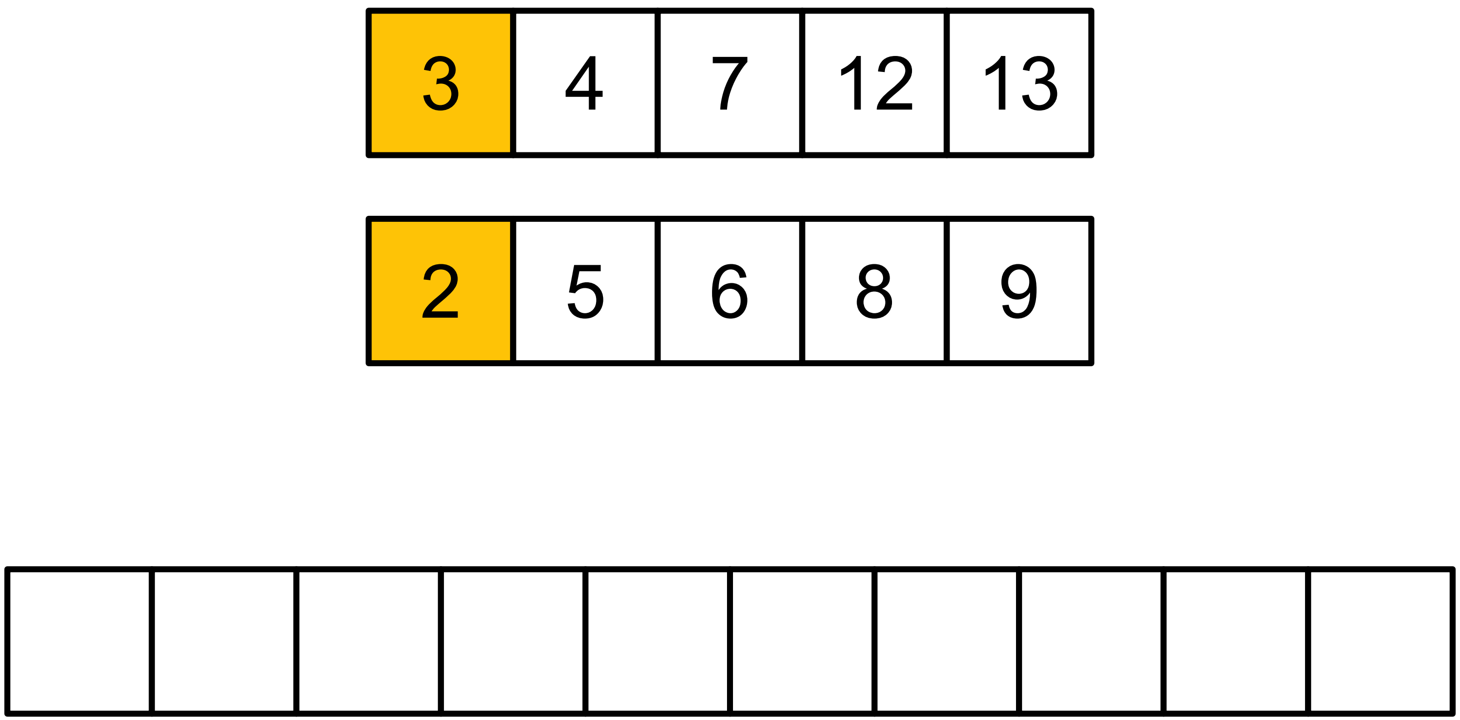
- 입력 배열을 반으로 쪼개서 각각 정렬하고 합친다
- 분할 정복 (Divide and Conquer)의 대표적 예시

Merging two sorted values

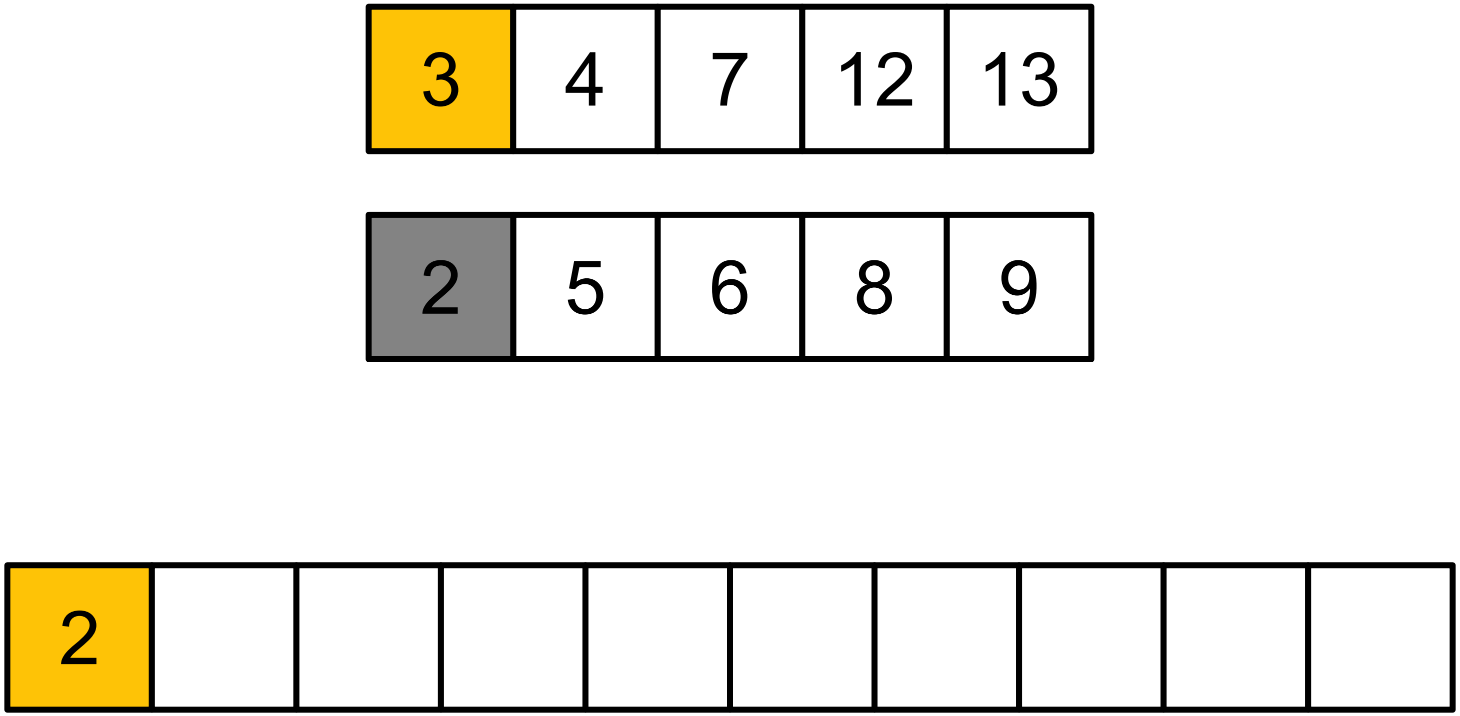
3	4	7	12	13

2	5	6	8	9

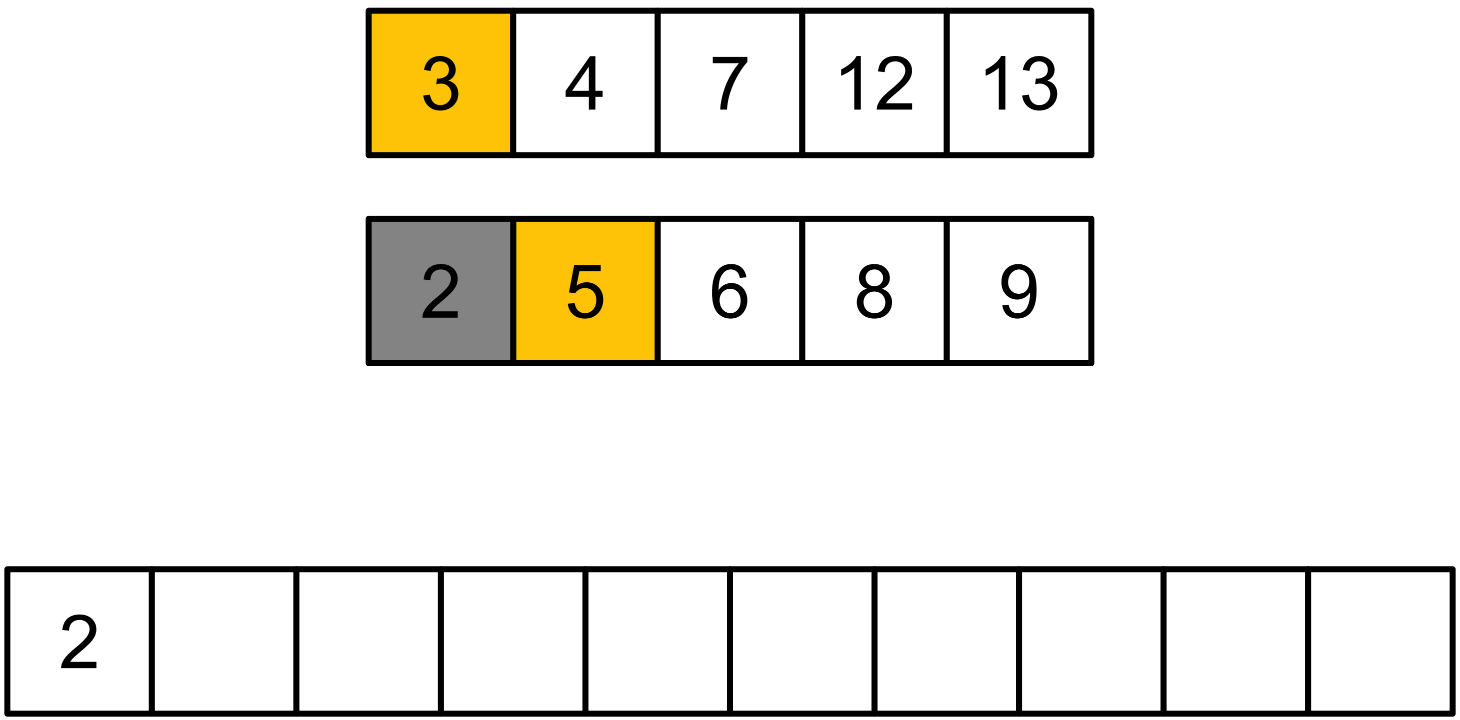
Merging two sorted values



Merging two sorted values



Merging two sorted values



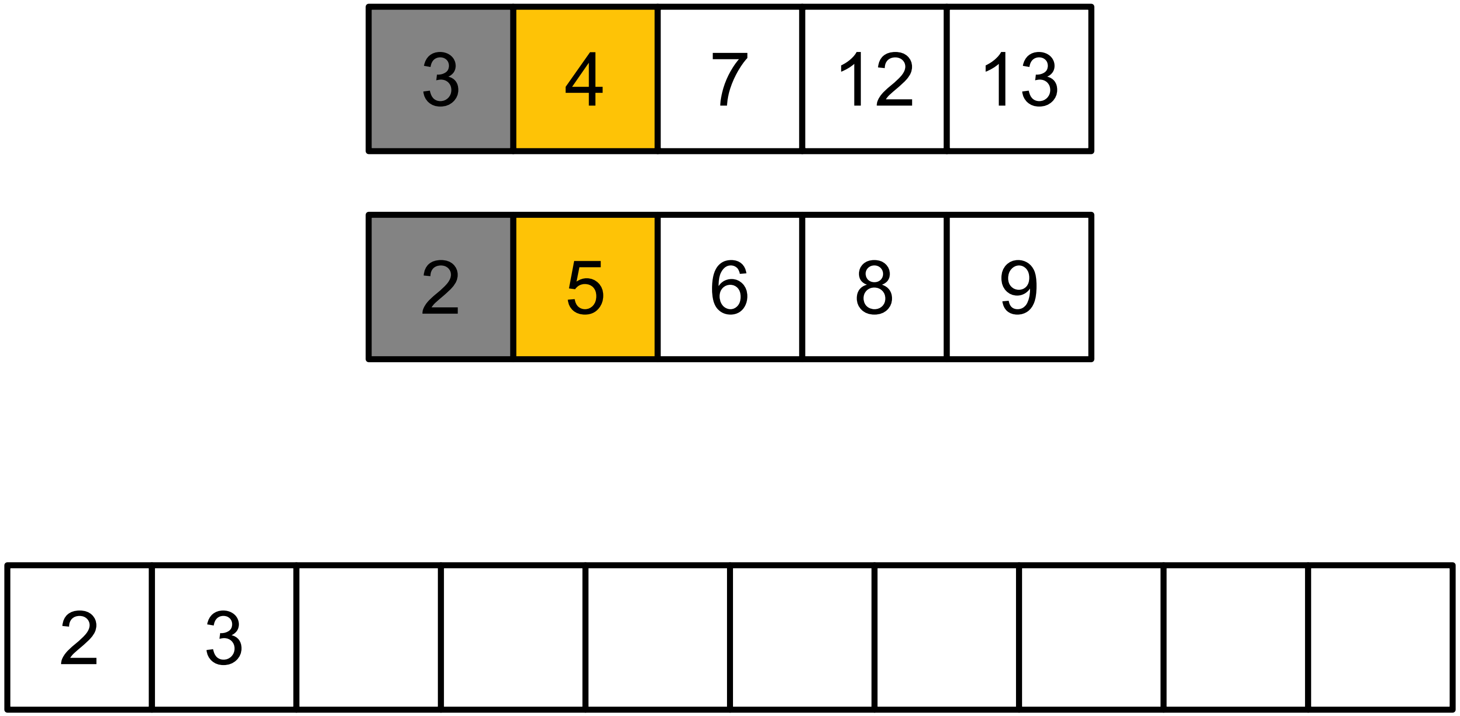
Merging two sorted values

3	4	7	12	13

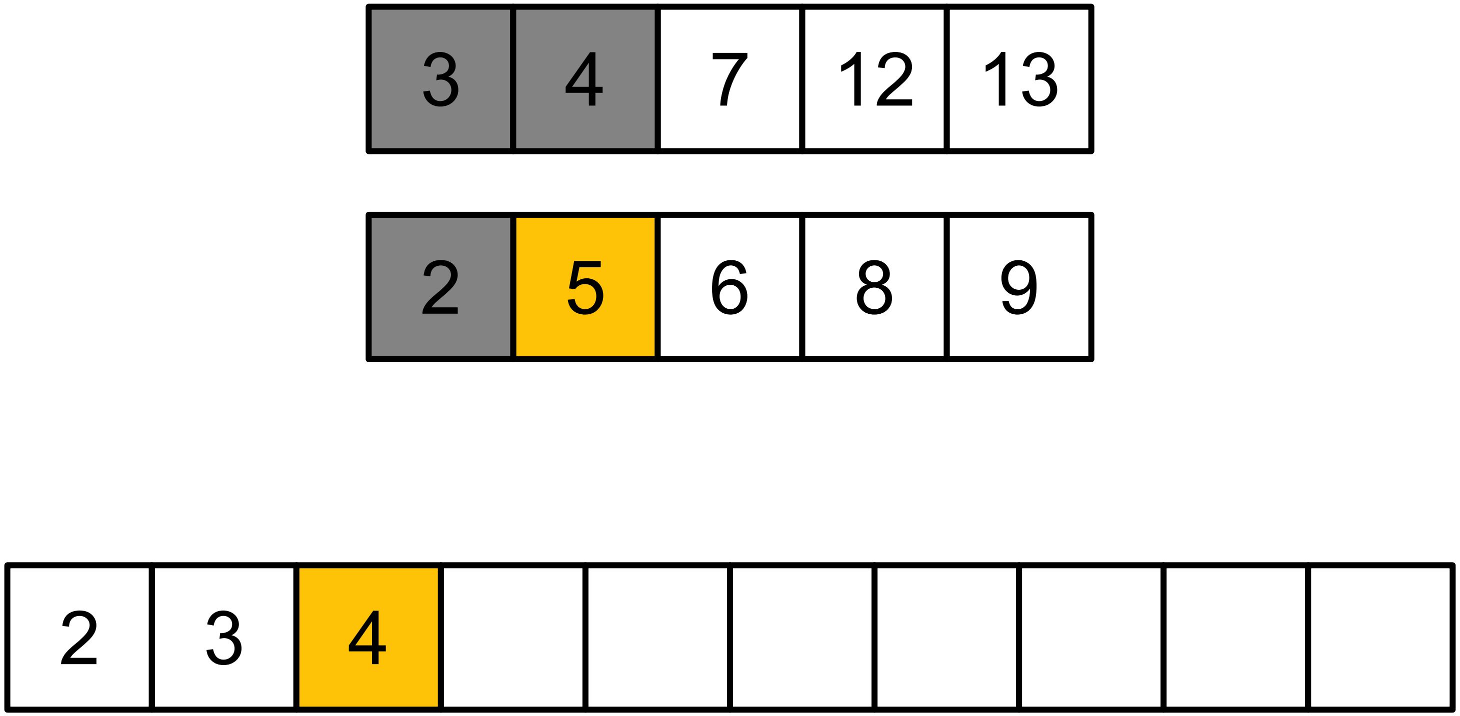
2	5	6	8	9

2	3								

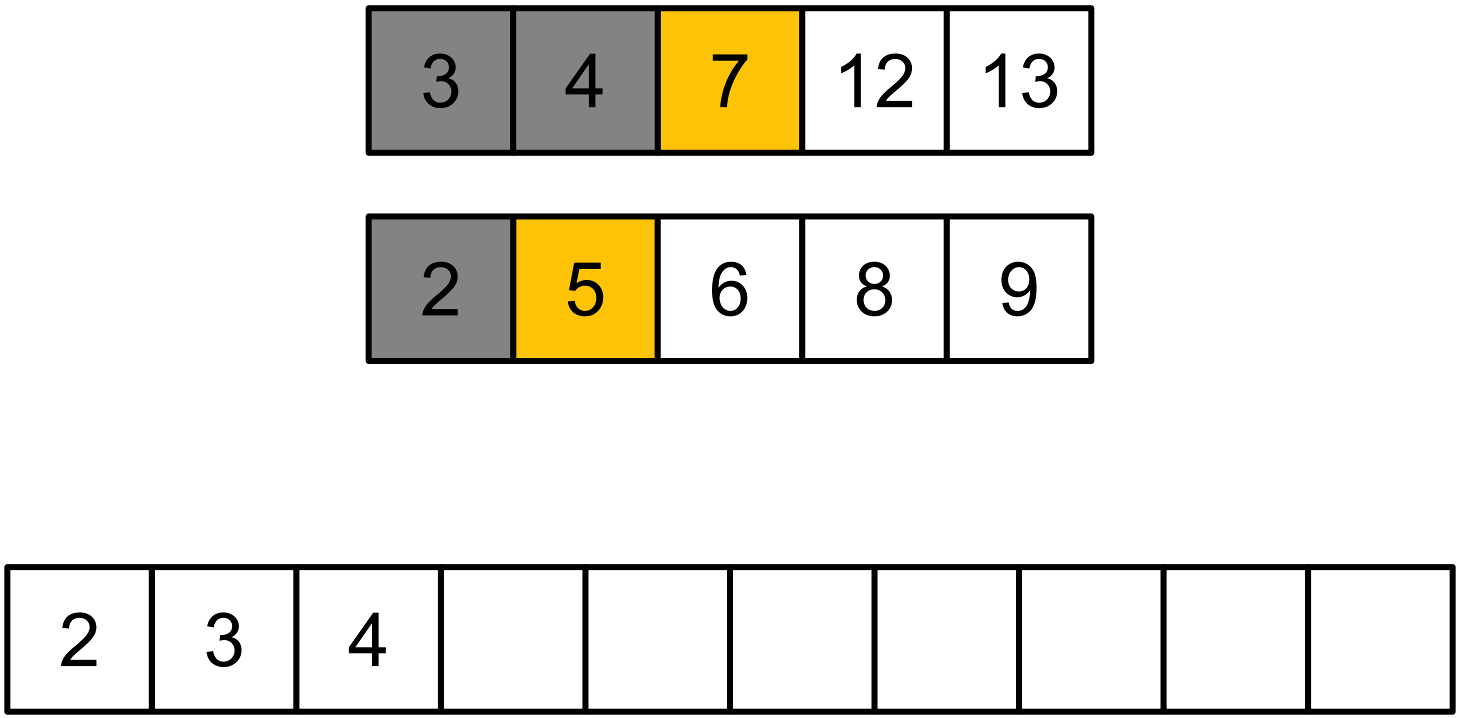
Merging two sorted values



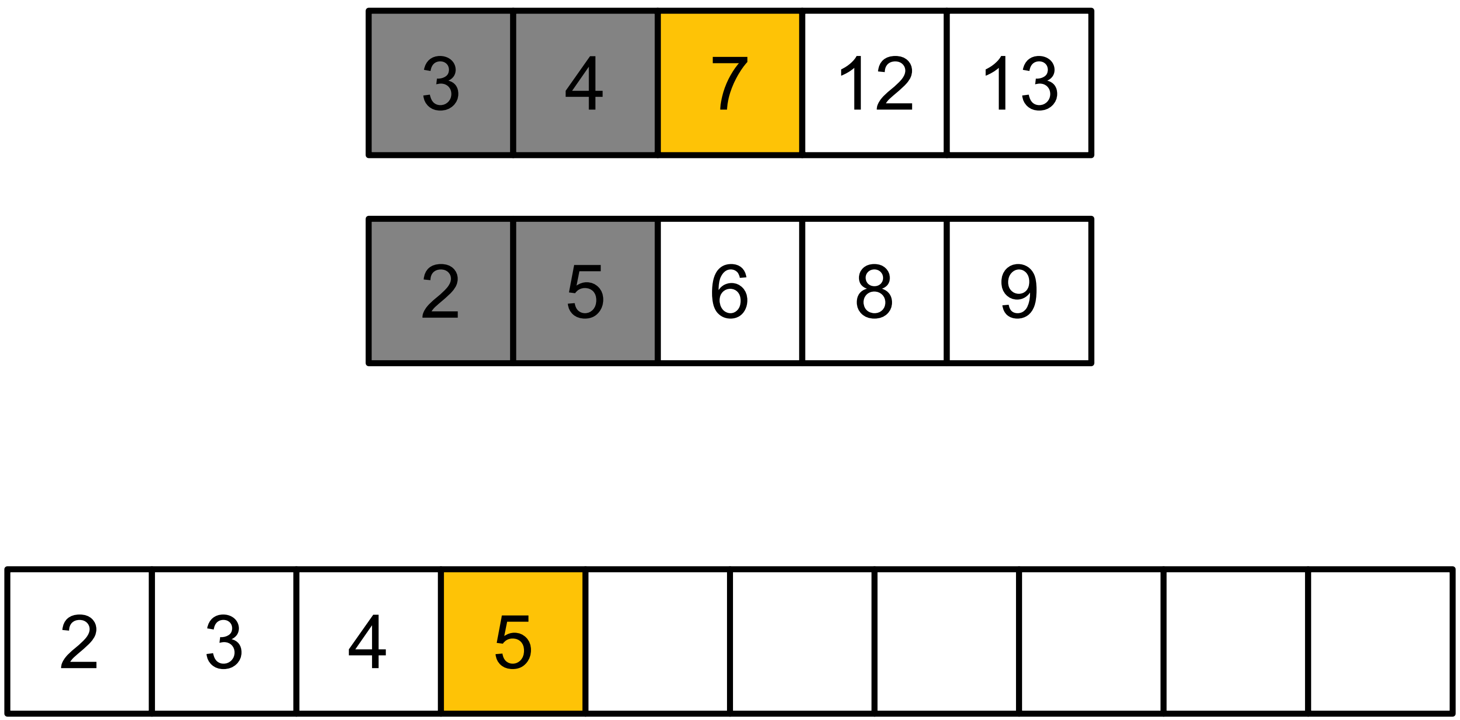
Merging two sorted values



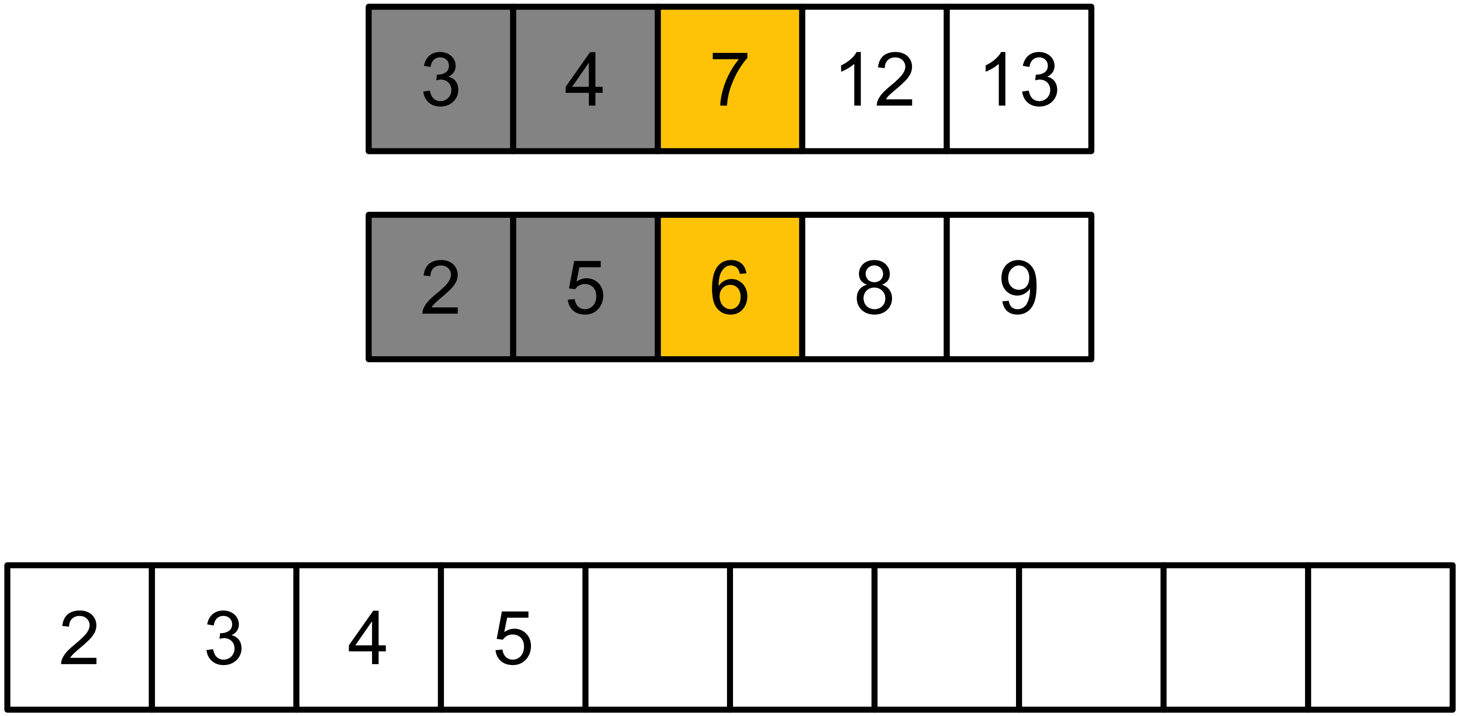
Merging two sorted values



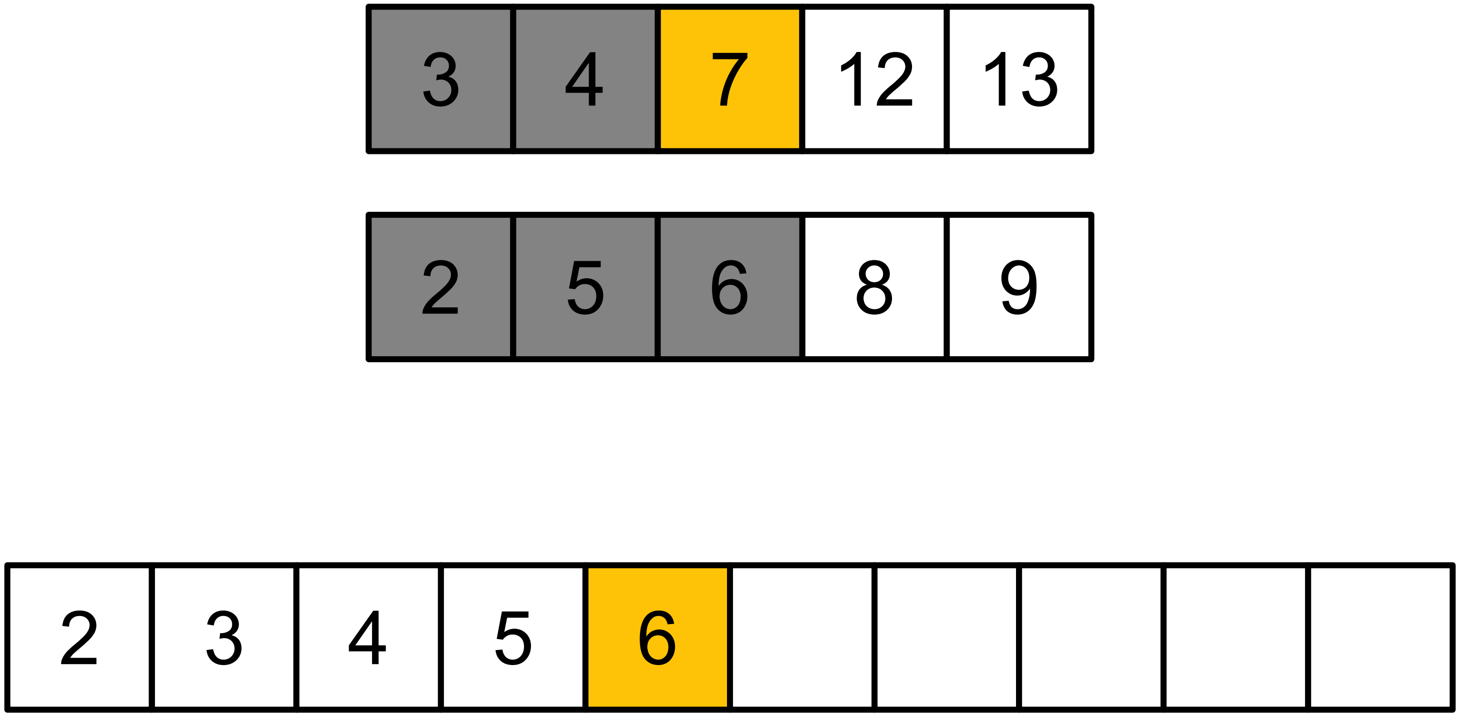
Merging two sorted values



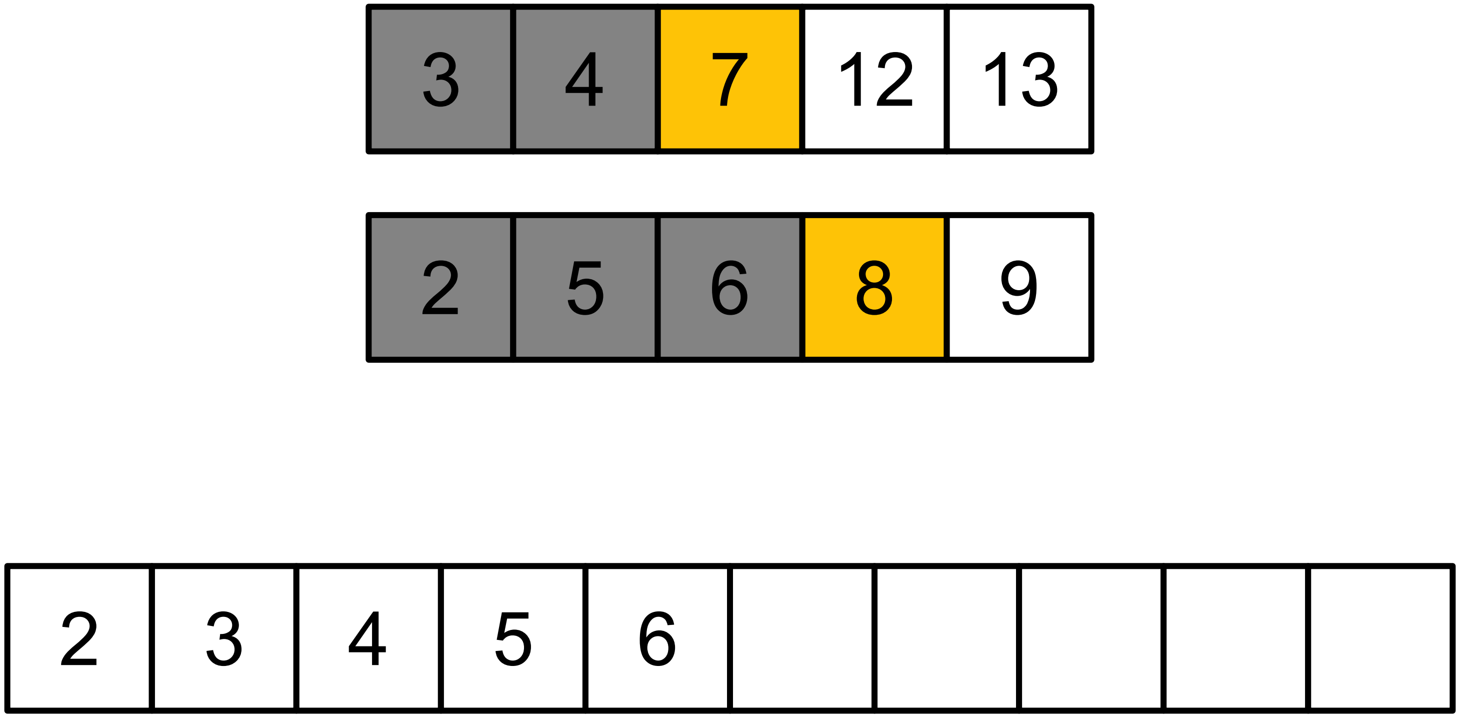
Merging two sorted values



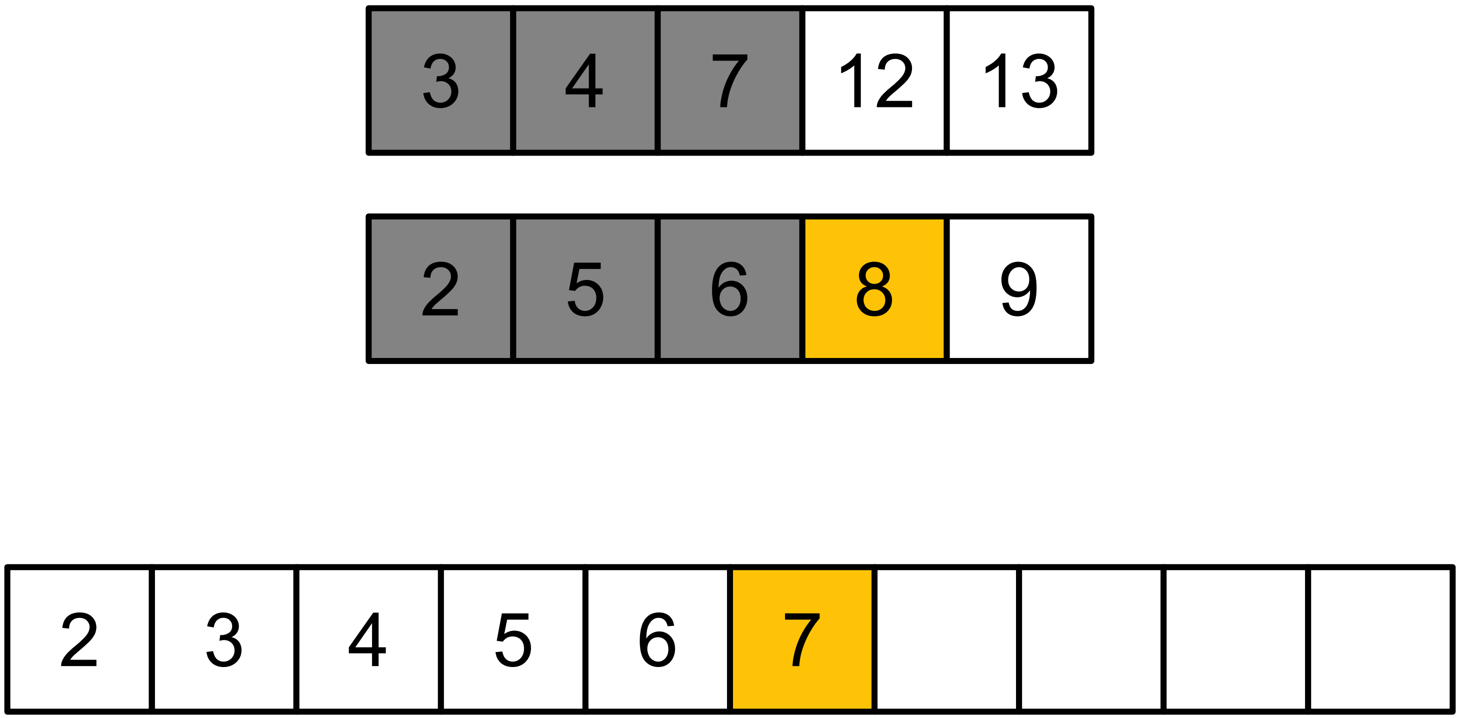
Merging two sorted values



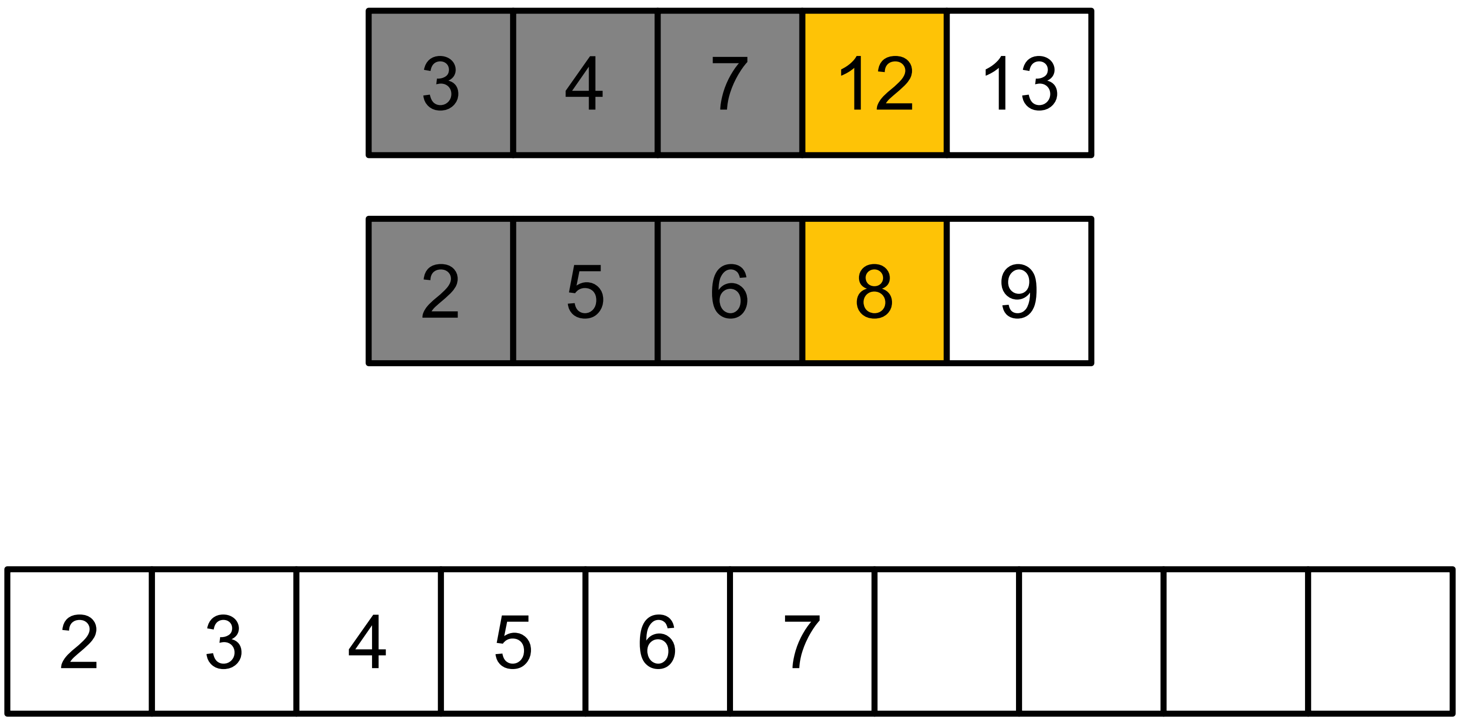
Merging two sorted values



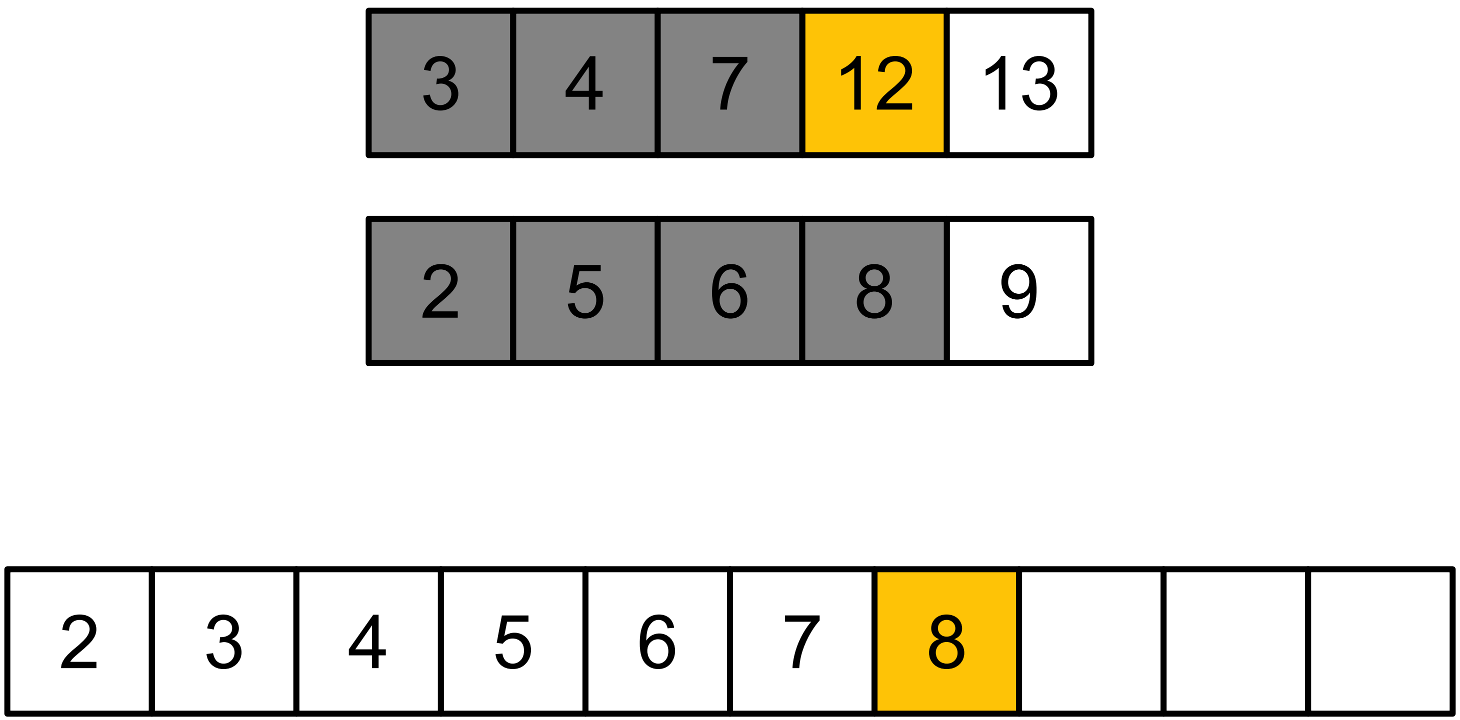
Merging two sorted values



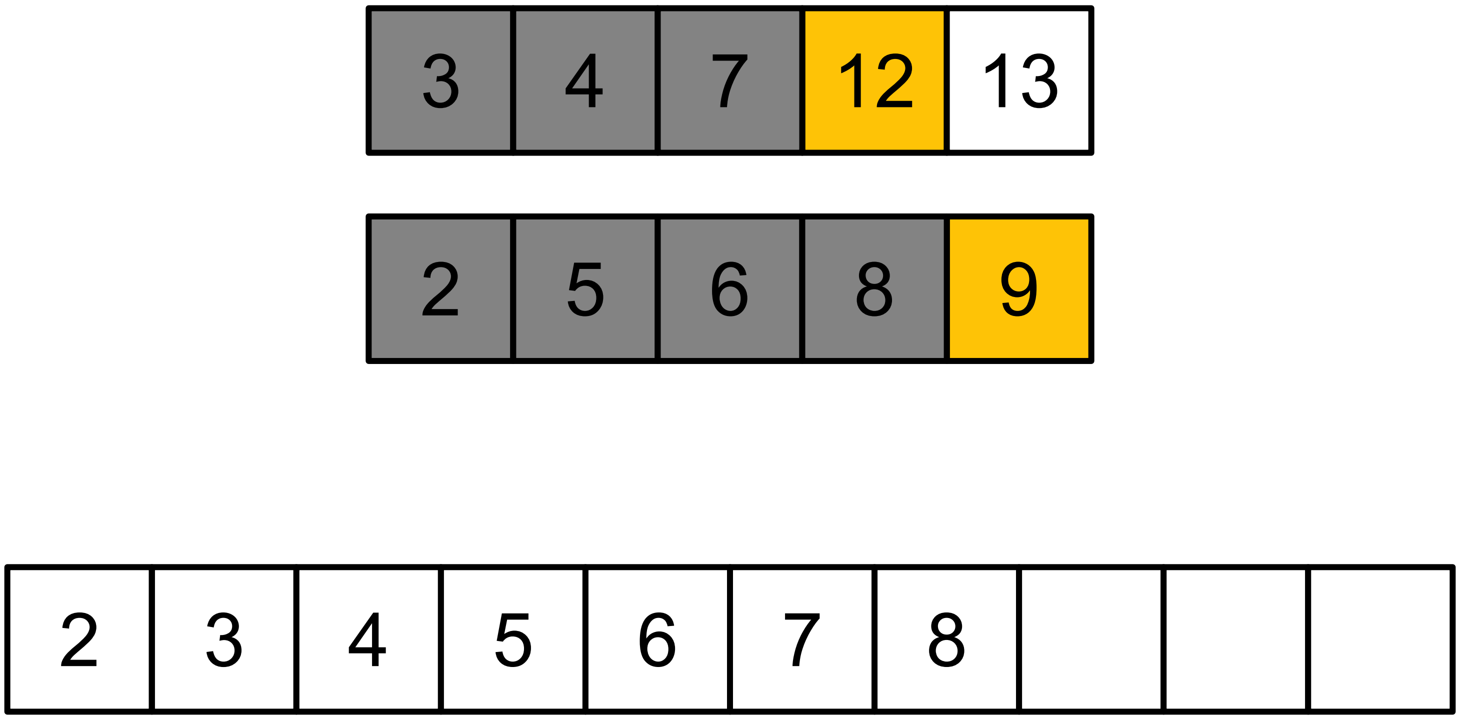
Merging two sorted values



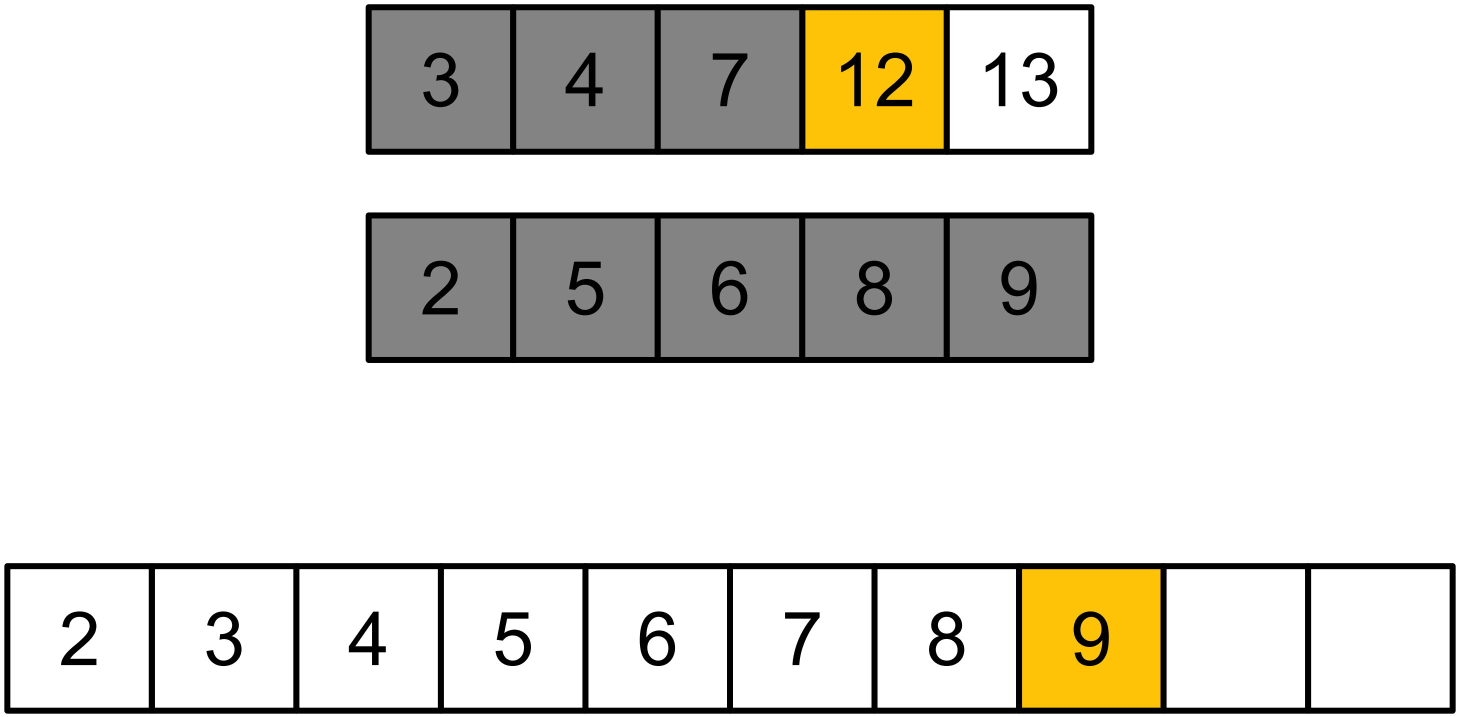
Merging two sorted values



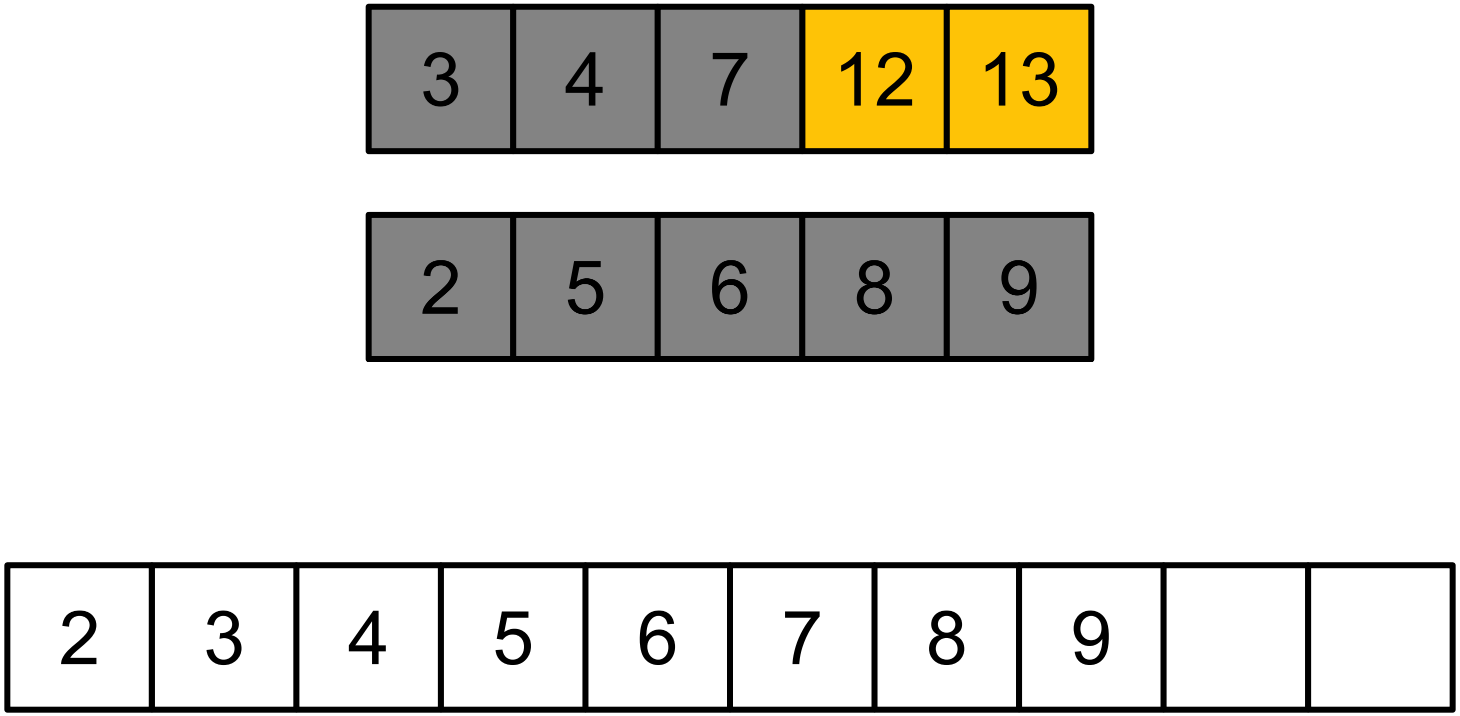
Merging two sorted values



Merging two sorted values



Merging two sorted values



Merging two sorted values

3	4	7	12	13

2	5	6	8	9

2	3	4	5	6	7	8	9	12	13

Merging two sorted values

3	4	7	12	13

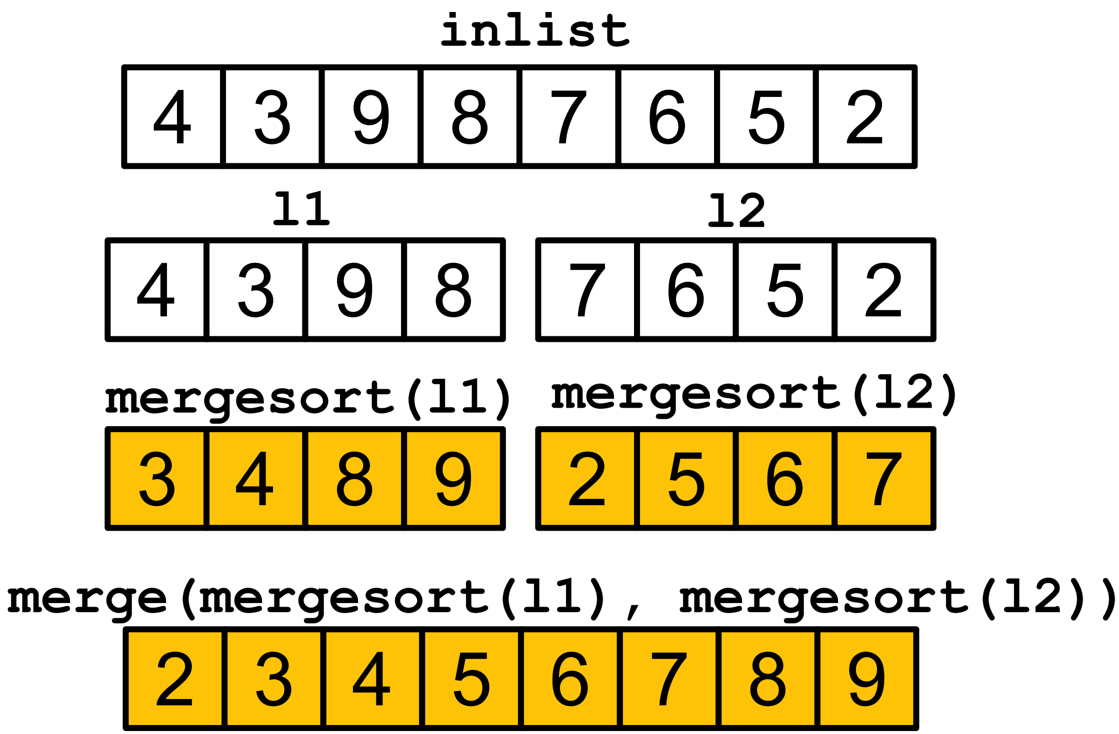
2	5	6	8	9

2	3	4	5	6	7	8	9	12	13

The cost for merging two sorted values: $\Theta(n)$ (n is # values)

Merge Sort - Pseudo Code

```
List mergesort(List inlist) {
  if (inlist.length() <= 1)return inlist;
  List l1 = half of the items from inlist;
  List l2 = other half of items from inlist;
  return merge(mergesort(l1),
               mergesort(l2));
}
```



Merge Sort

4	3	9	8	7	6	5	2

Merge Sort

4	3	9	8

7	6	5	2

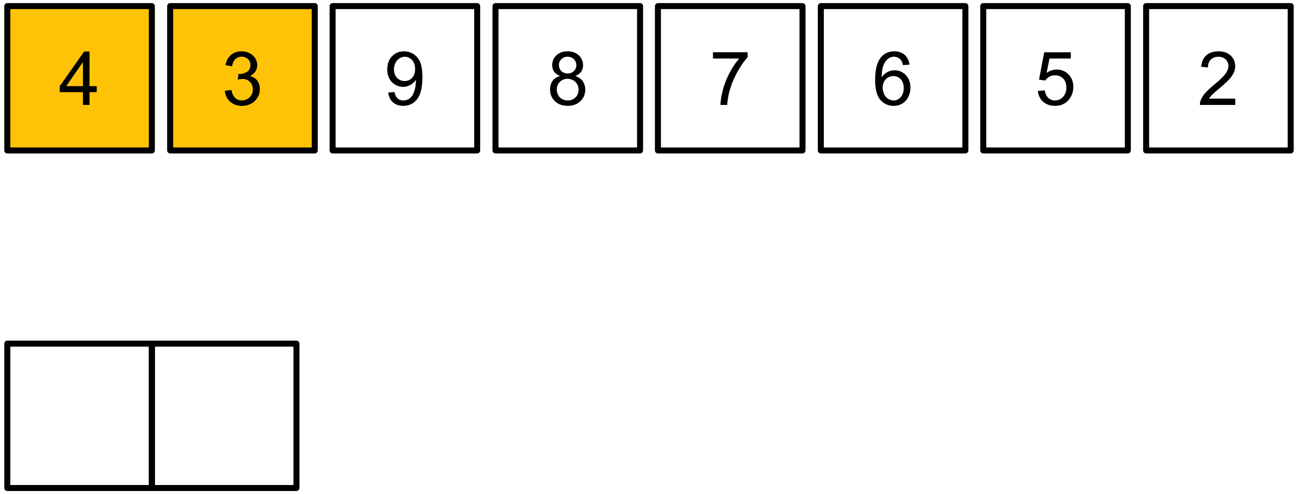
Merge Sort

4	3	9	8	7	6	5	2

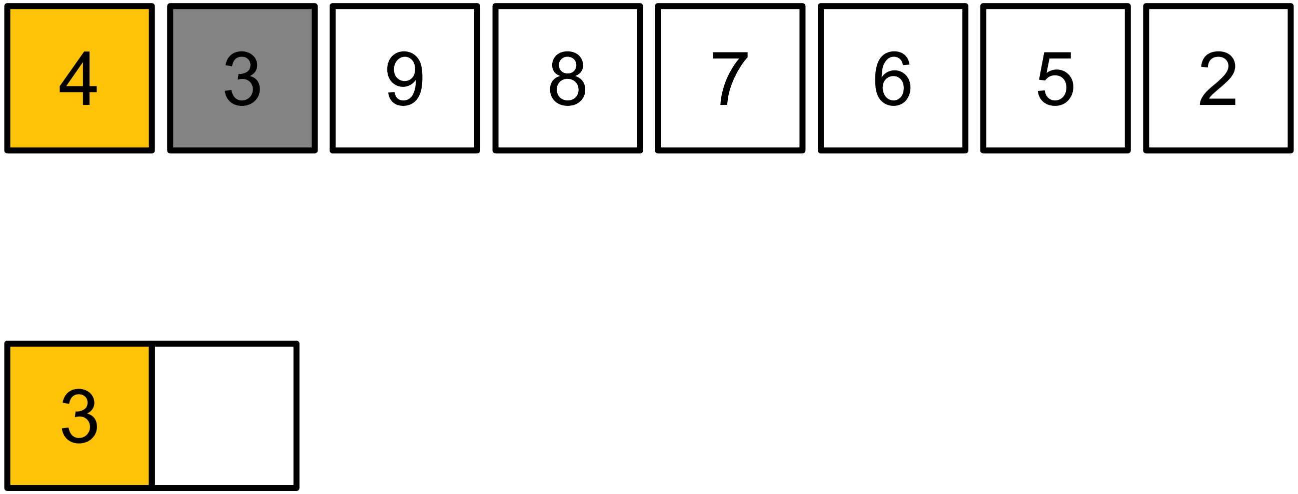
Merge Sort

4	3	9	8	7	6	5	2

Merge Sort



Merge Sort



Merge Sort

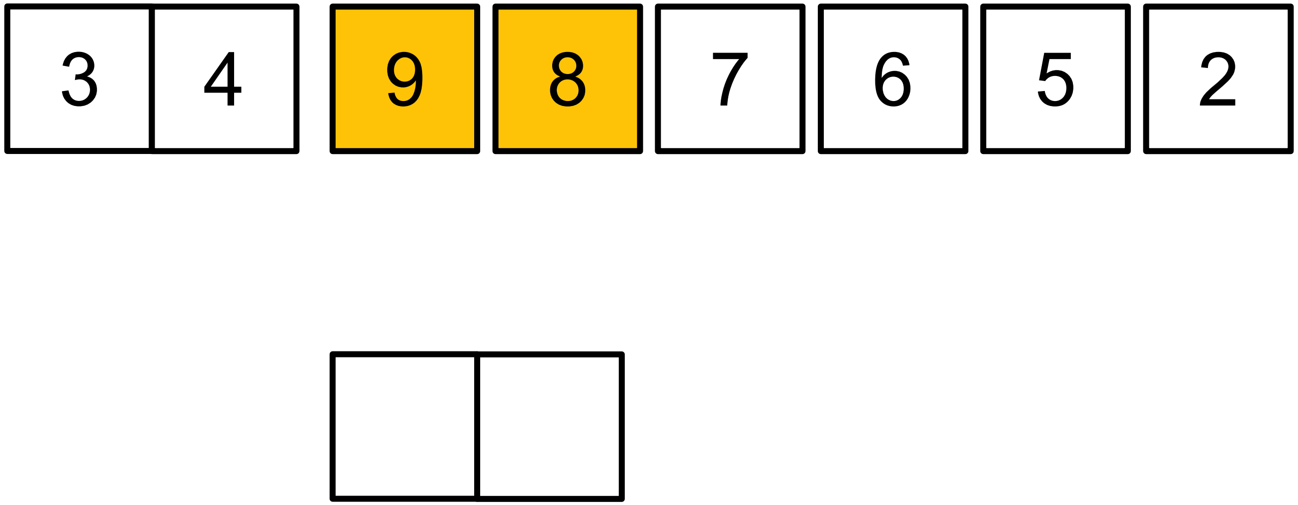
4	3	9	8	7	6	5	2

3	4

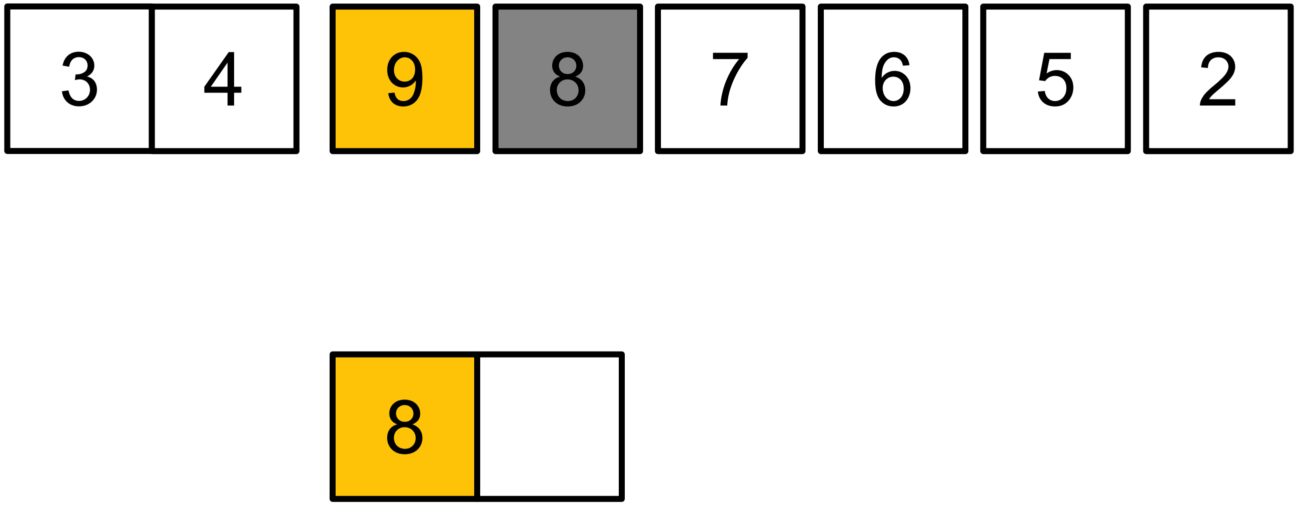
Merge Sort

3	4	9	8	7	6	5	2

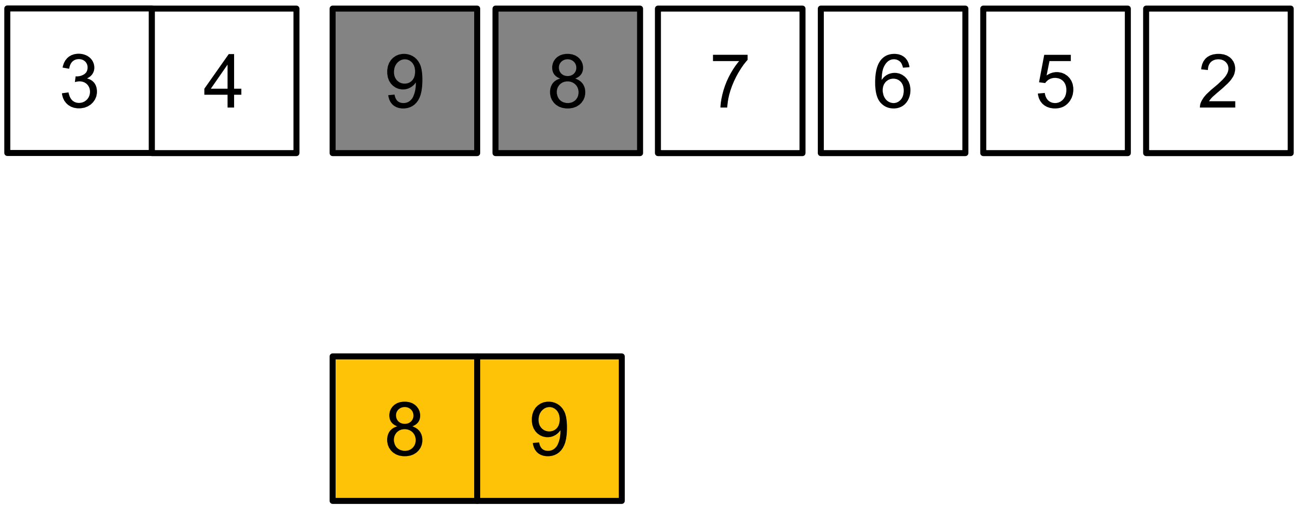
Merge Sort



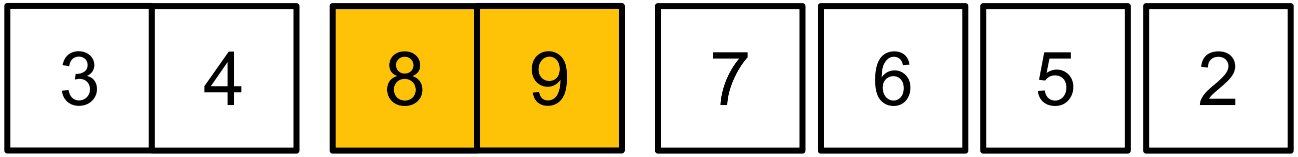
Merge Sort



Merge Sort



Merge Sort



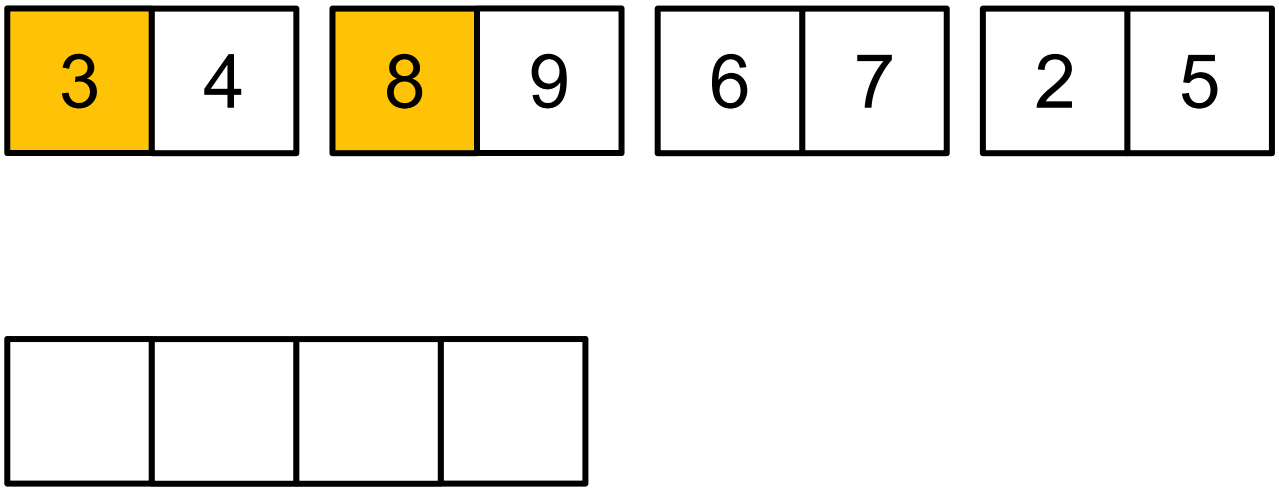
Merge Sort

3	4	8	9	6	7	5	2

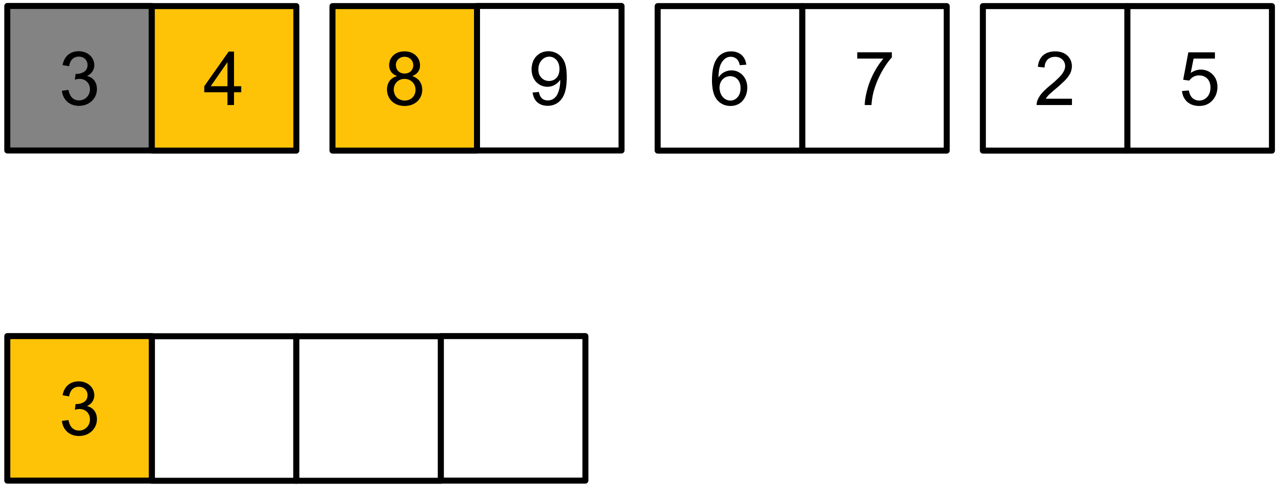
Merge Sort

3	4	8	9	6	7	2	5

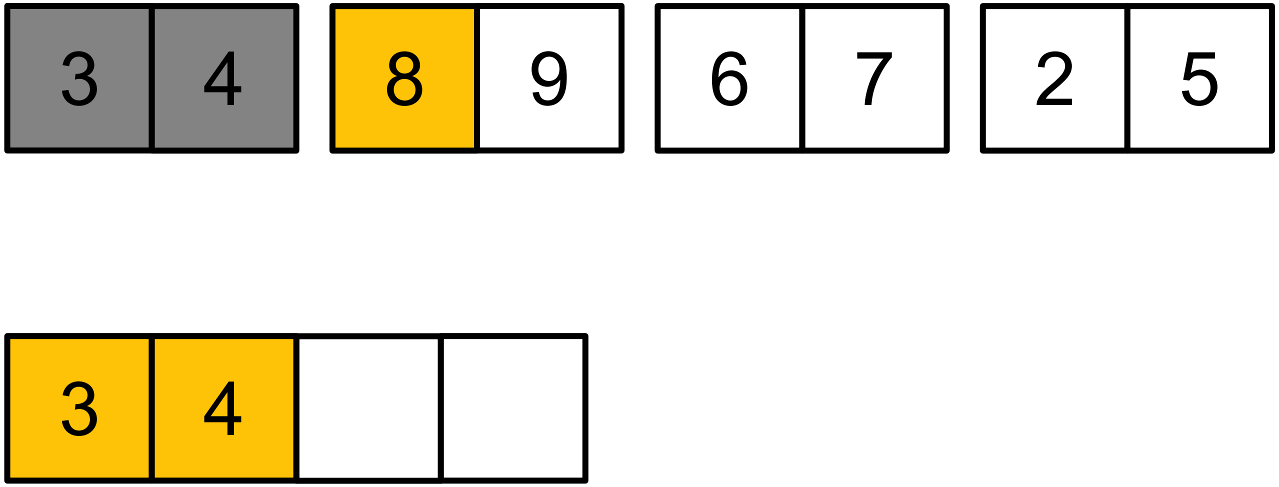
Merge Sort



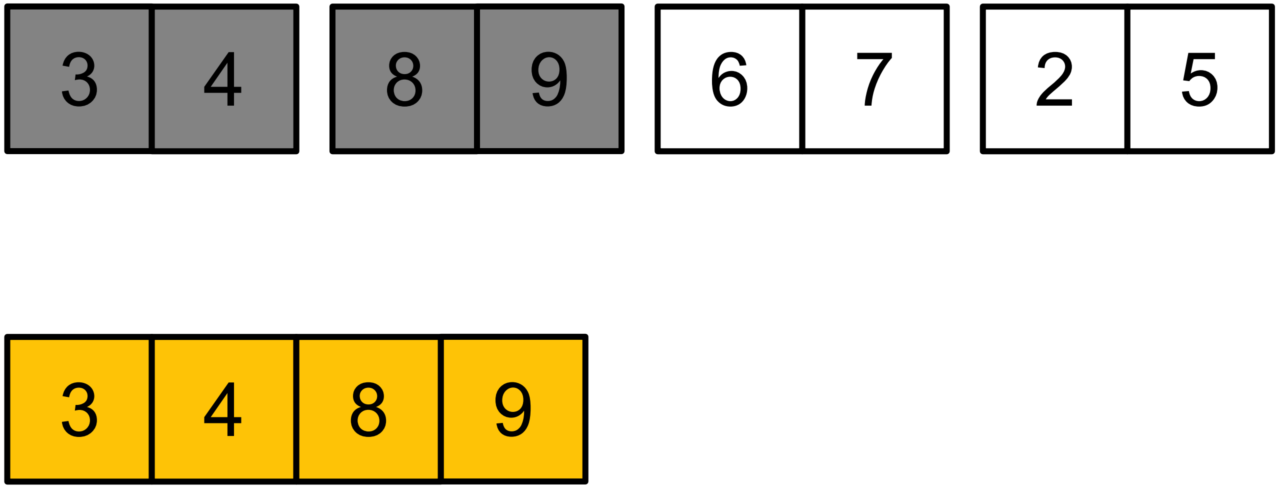
Merge Sort



Merge Sort



Merge Sort



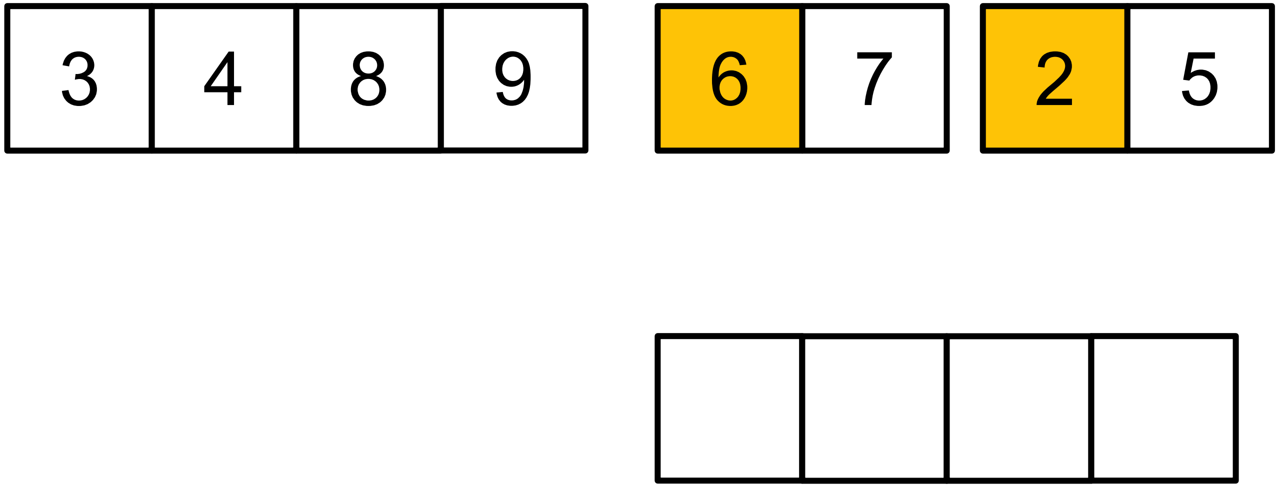
Merge Sort

3	4	8	9

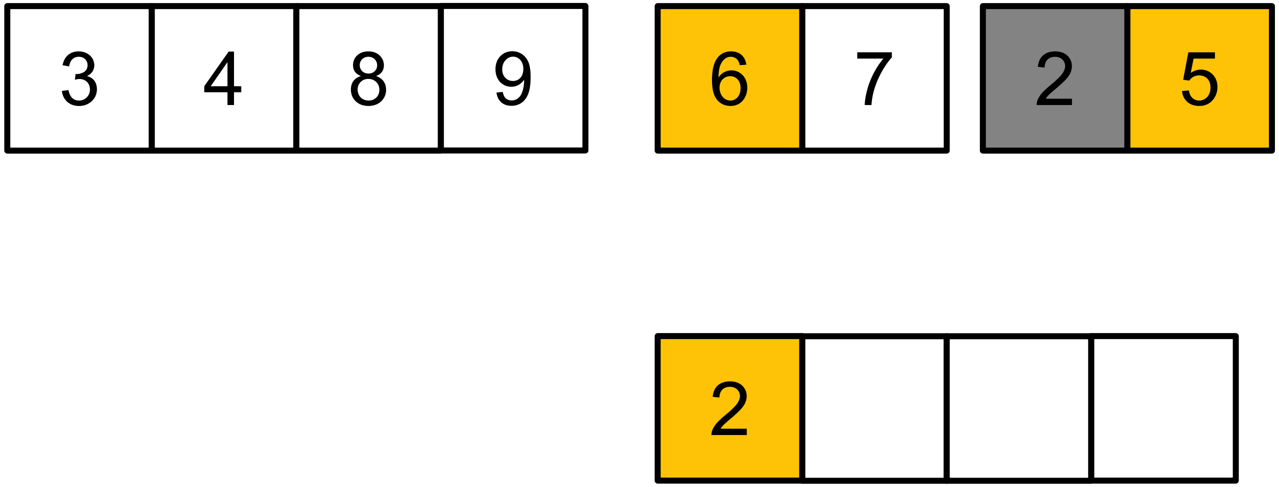
6	7

2	5

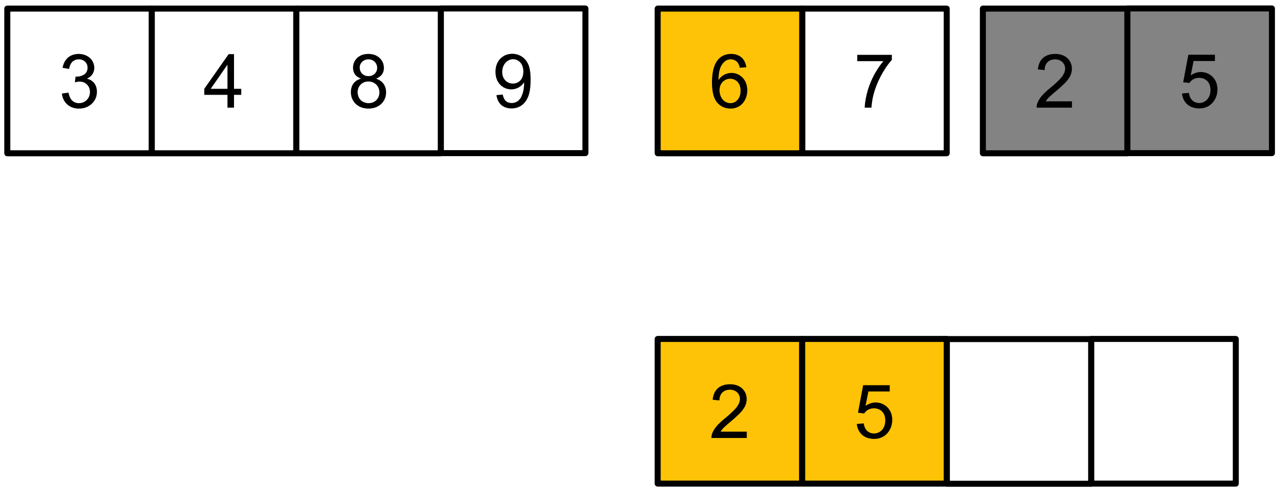
Merge Sort



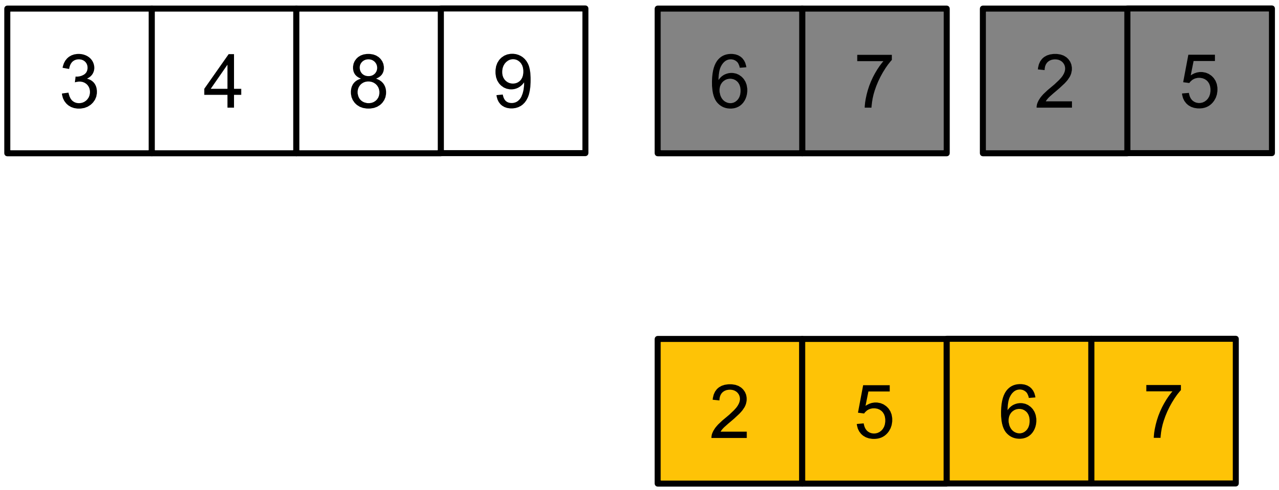
Merge Sort



Merge Sort



Merge Sort

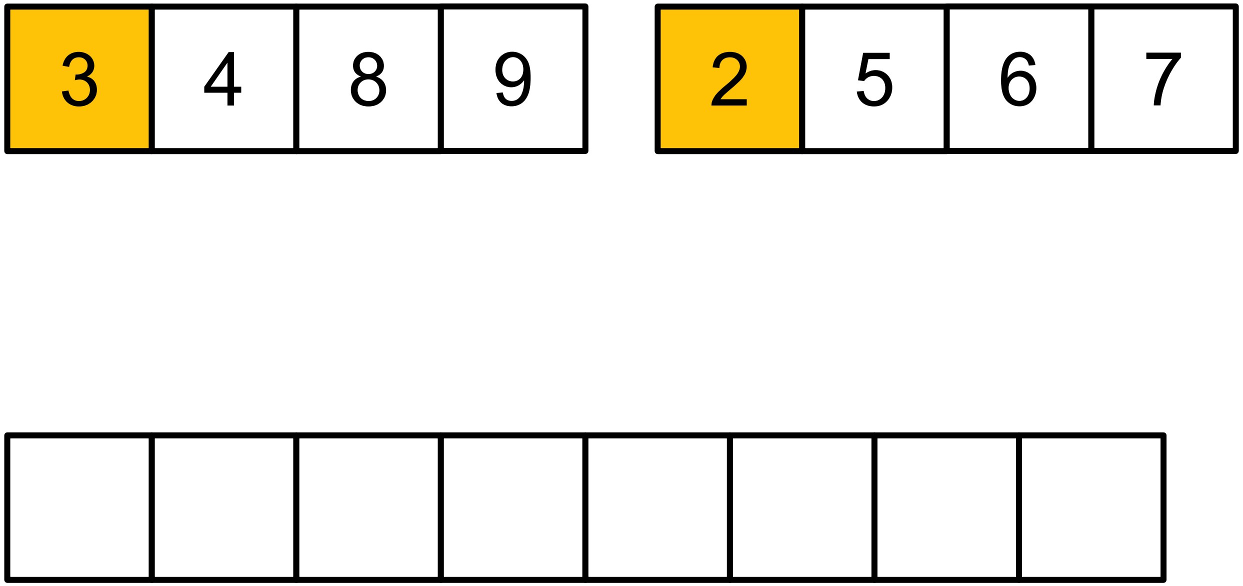


Merge Sort

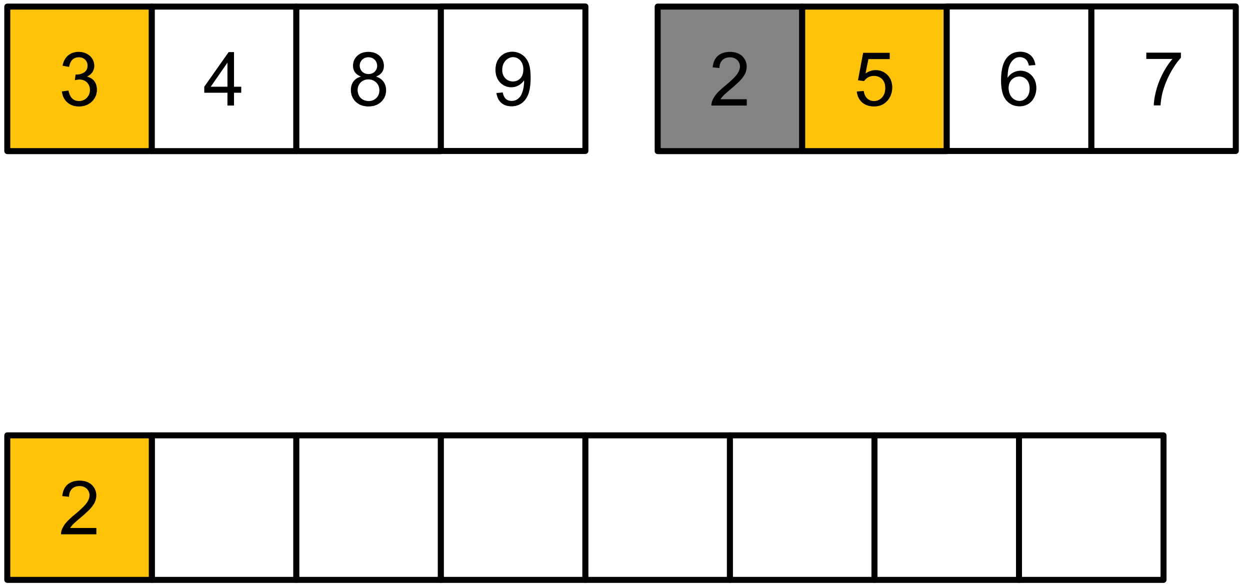
3	4	8	9

2	5	6	7

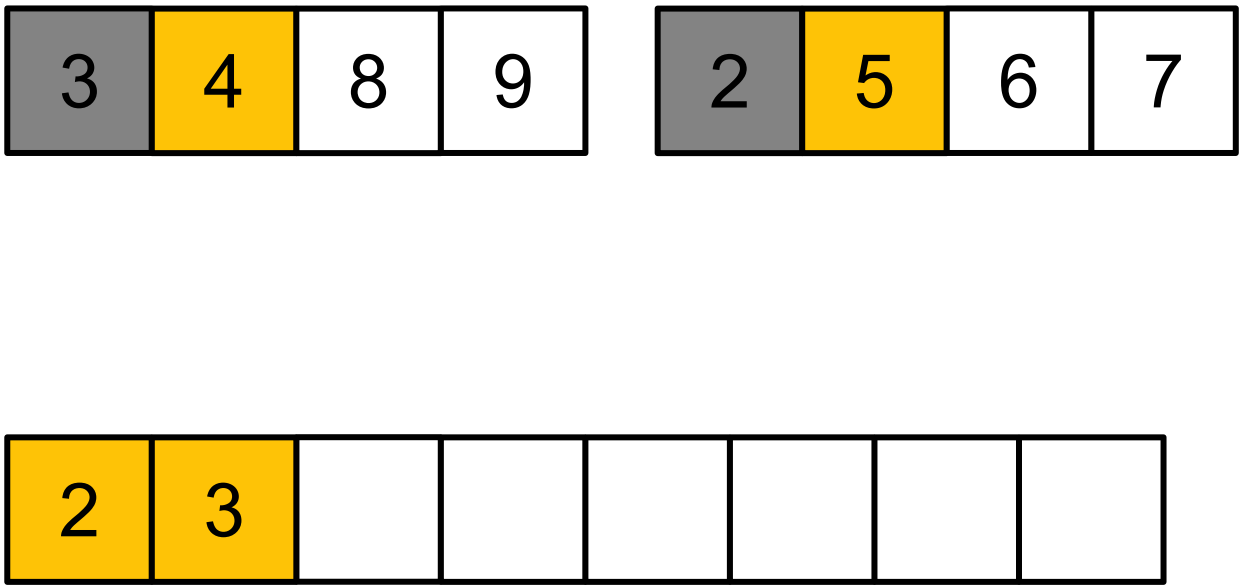
Merge Sort



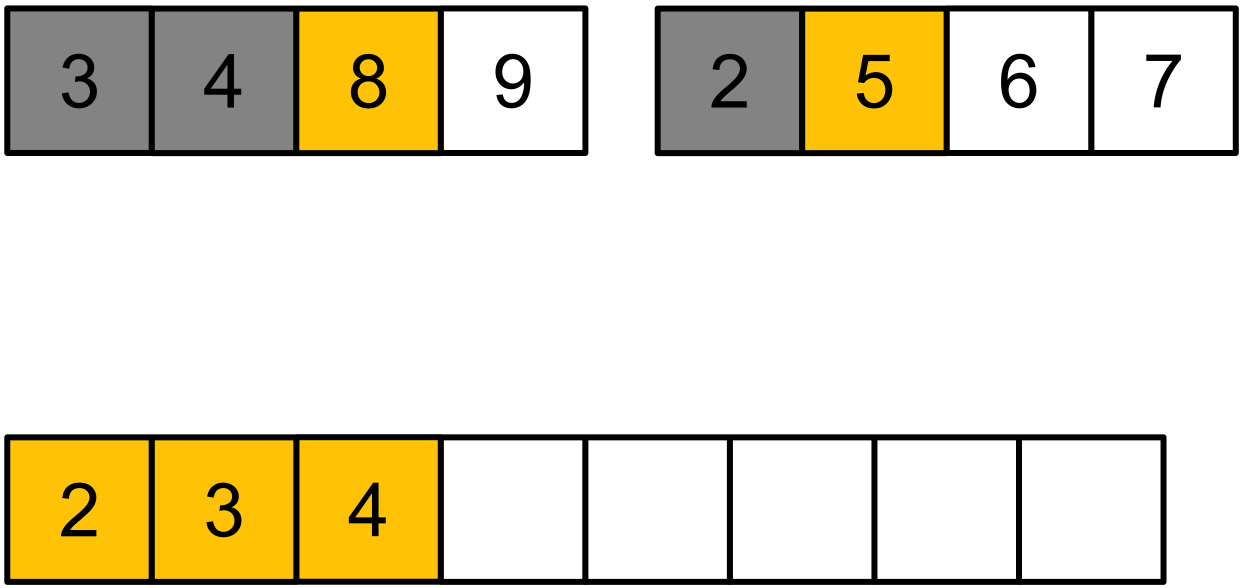
Merge Sort



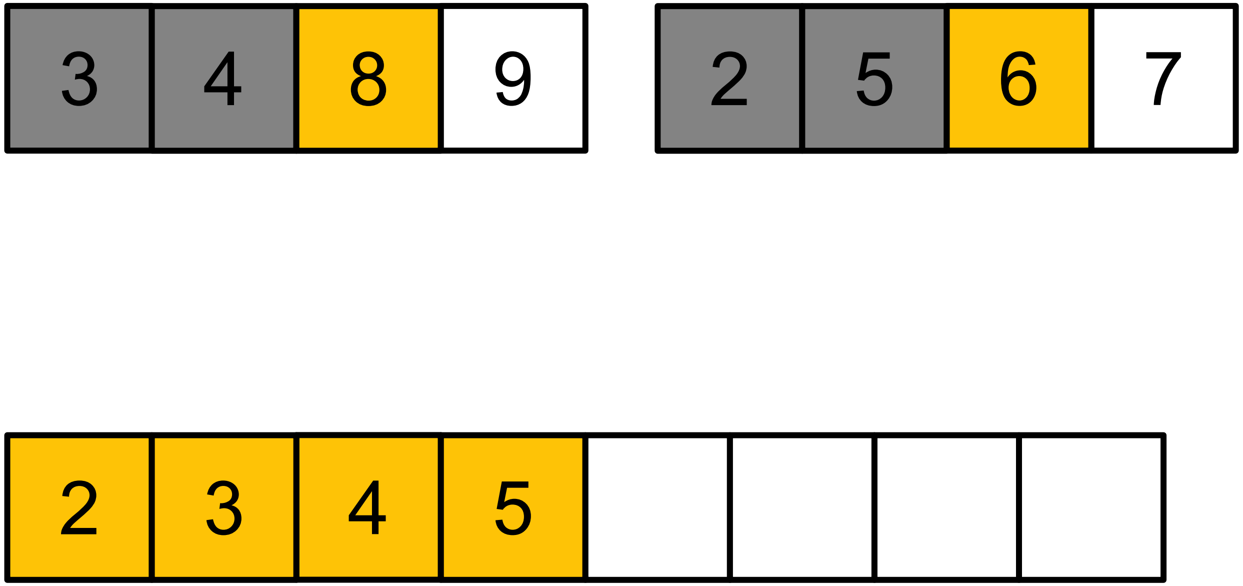
Merge Sort



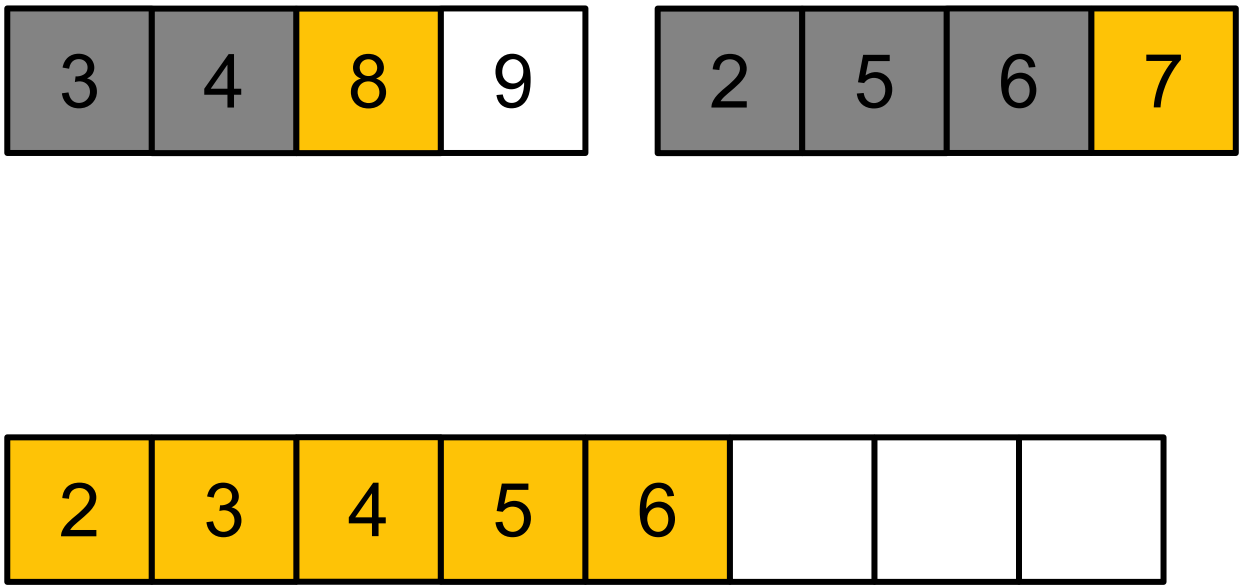
Merge Sort



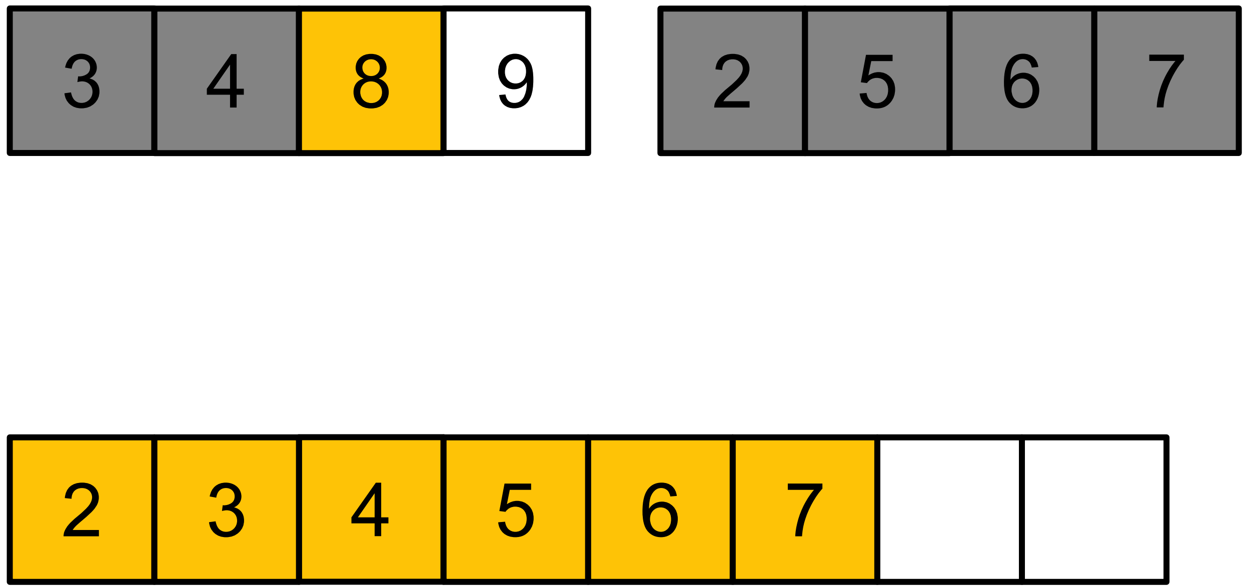
Merge Sort



Merge Sort



Merge Sort



Merge Sort

3	4	8	9		2	5	6	7

2	3	4	5	6	7	8	9

Merge Sort

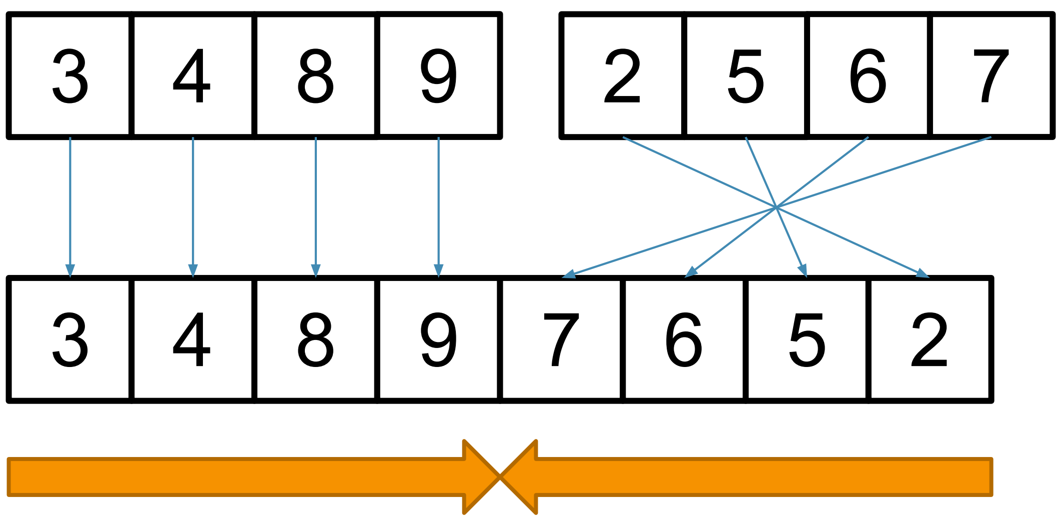
2	3	4	5	6	7	8	9

Merge Sort Implementation

```
static <E extends Comparable<? super E>>
void mergesort(E[] A, E[] temp, int l, int r) {
    int mid = (l + r) / 2; // Select midpoint
    if (l == r) return; // List has one element
    mergesort(A, temp, l, mid); // Mergesort first half
    mergesort(A, temp, mid + 1, r); // Mergesort second half
    for (int i = l; i <= r; i++) // Copy subarray to temp
        temp[i] = A[i];
    // Do the merge operation back to A
    int i1 = l; int i2 = mid + 1;
    for (int curr = l; curr <= r; curr++) {
        if (i1 == mid + 1) // Left sublist exhausted
            A[curr] = temp[i2++];
        else if (i2 > r) // Right sublist exhausted
            A[curr] = temp[i1++];
        else if (temp[i1].compareTo(temp[i2]) < 0) // Get smaller
            A[curr] = temp[i1++];
        else A[curr] = temp[i2++];
    }
}
```

Optimized Merge Sort

- 두 가지 최적화 방법
  - 입력 배열 크기가 작을 때 insertion sort 사용하기.
    - Insertion sort: 재귀 없음. 배열 값 복사 안해도 됨.
  - 리스트의 끝을 확인 안하기
    - 두 번째 리스트를 반대 순서로 temp 배열에 넣기



Optimized Merge Sort

```
static <E extends Comparable<? super E>>
void mergesort(E[] A, E[] temp, int l, int r) {
   int i, j, k, mid = (l + r) / 2; // Select the midpoint
   if (l == r) return; // List has one element
   if ((mid - l) >= THRESHOLD) mergesort(A, temp, l, mid);
   else inssort(A, l, mid - l + 1);
   if ((r - mid) > THRESHOLD) mergesort(A, temp, mid + 1, r);
   else inssort(A, mid + 1, r - mid);
   // Do the merge operation. First, copy 2 halves to temp.
   for (i = l; i <= mid; i++) temp[i] = A[i];
   for (j = 1; j <= r - mid; j++) temp[r - j + 1] = A[j + mid];
   // Merge sublists back to array
   for (i = l, j = r, k = l; k <= r; k++)
      if (temp[i].compareTo(temp[j]) < 0) A[k] = temp[i++];
      else A[k] = temp[j--];
}
```

Merge Sort Cost

- Merge sort cost:
  - $\Theta(n \log n)$ in the best, average and worst cases
- Merge sort is also good for sorting linked lists.
  - Because merging requires only sequential access
- Merge sort requires twice the space.

Overview

- Merge Sort
- **Quicksort**

Quicksort

- 또 다른 분할 정복 알고리즘

- 입력 배열이 주어졌을 때
  - 배열의 값 하나를 선택해서 'pivot'으로 지정
  - pivot보다 작은 값이 pivot보다 큰 값 보다 항상 왼쪽에 오도록 배열을 재 정렬 = 'Partition 연산'
    - pivot과 같은 값은 어느 쪽에 있어도 상관 없음

```
static <E extends Comparable<? super E>>
int findpivot(E[] A, int i, int j){
    return (i+j)/2;
}
```

Quicksort

10	1	9	3	4	7	2	5	8	6

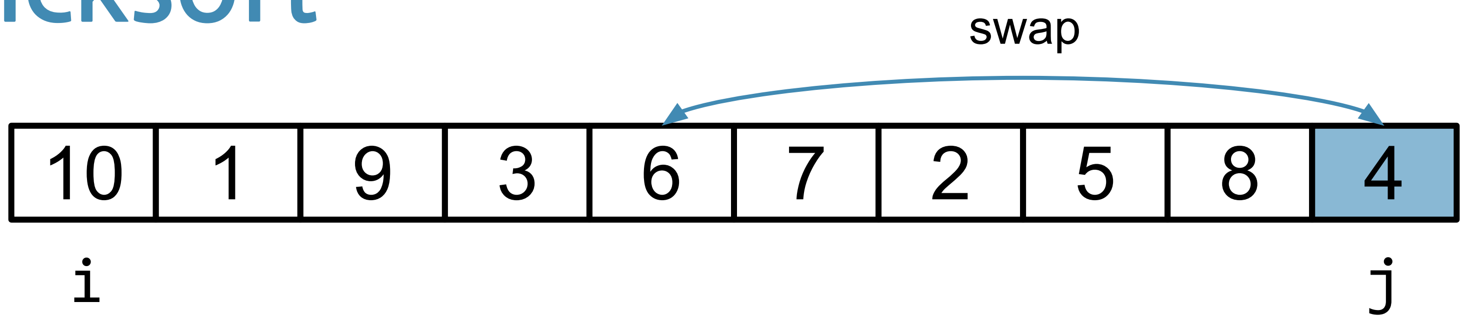
```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort




```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

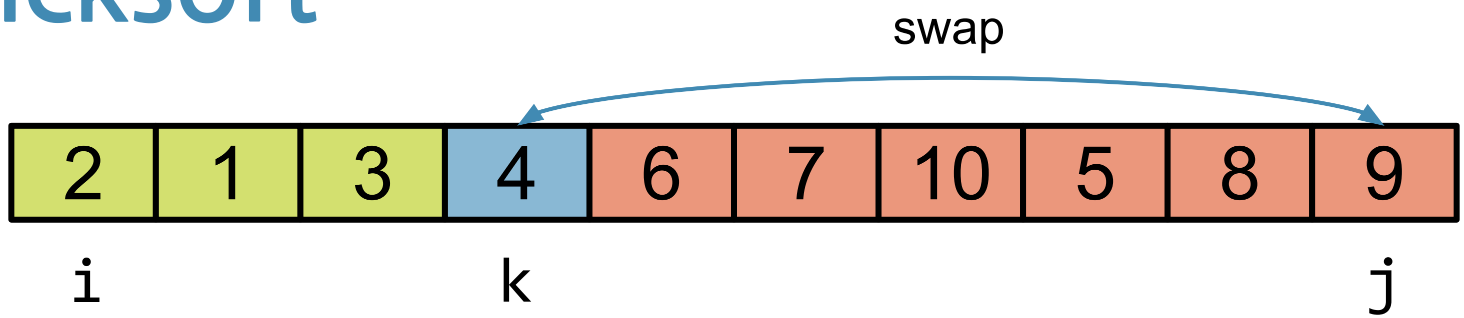
```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort




```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

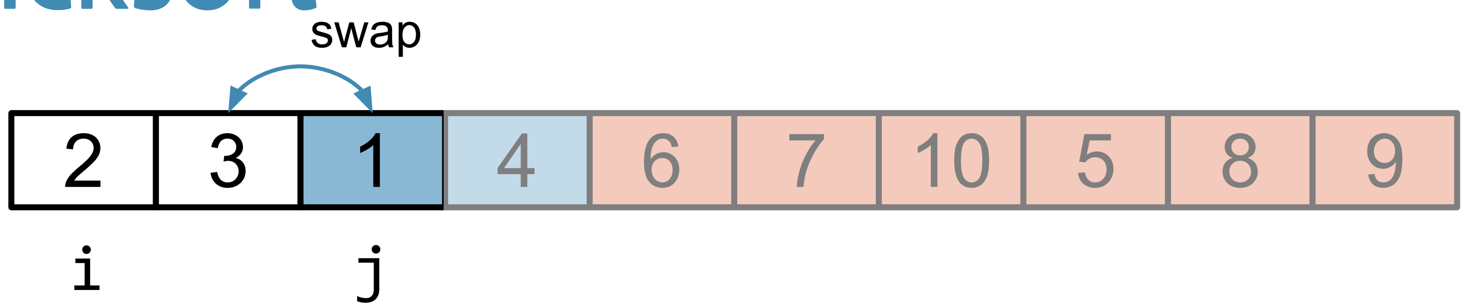
```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort



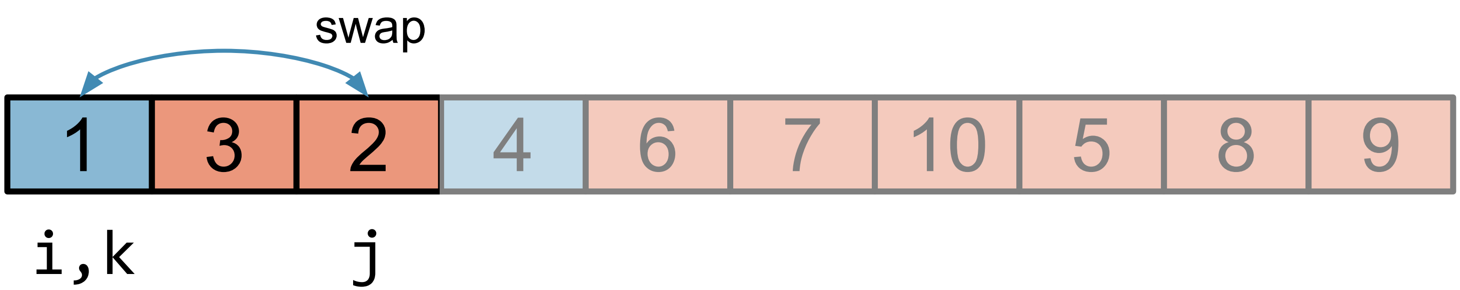

```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

2	3	1	4	6	7	10	5	8	9
i,k		j							

```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort




```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

1	3	2	4	6	7	10	5	8	9
i,k		j							

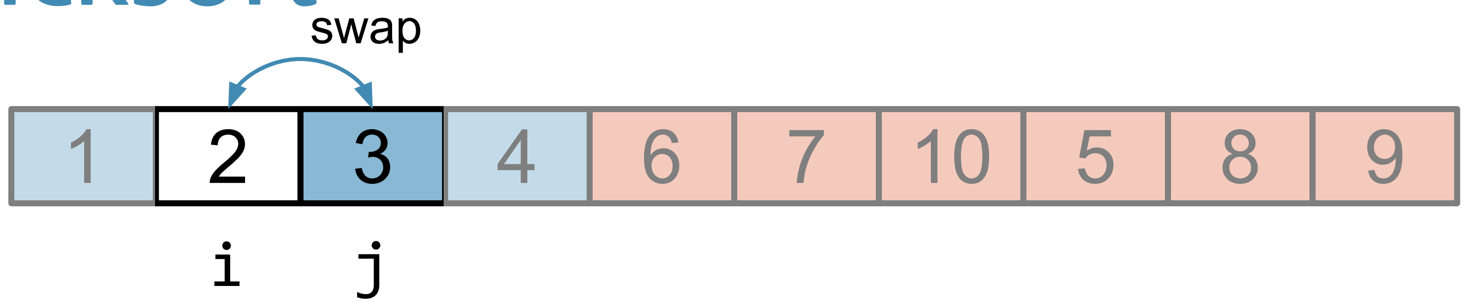
```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

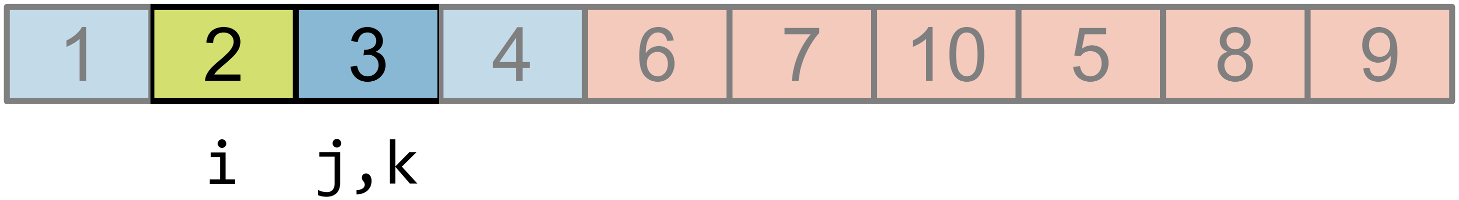
```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort




```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort



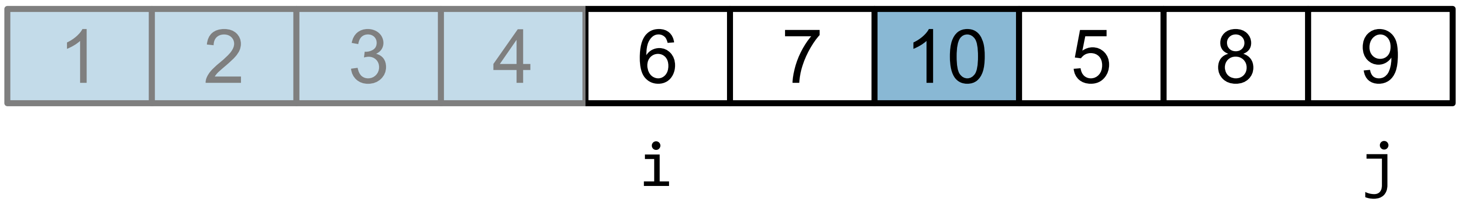

```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

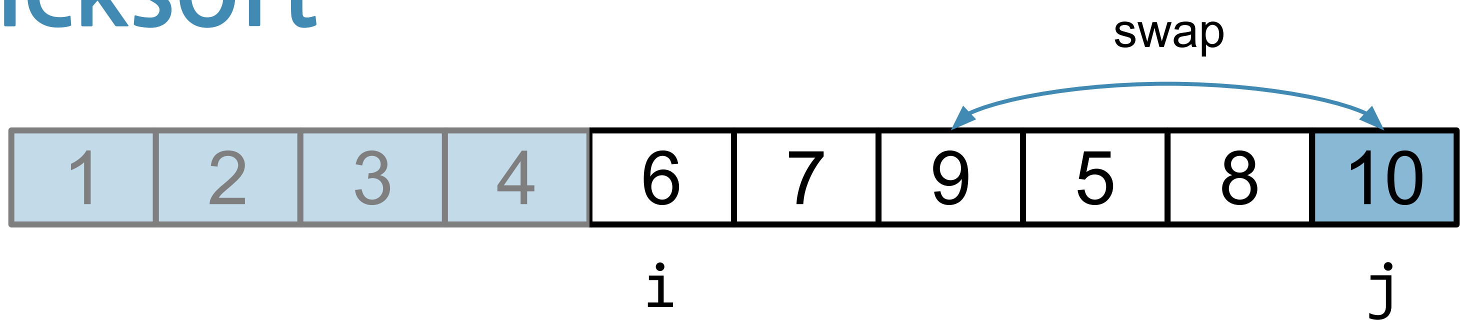
```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort




```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort



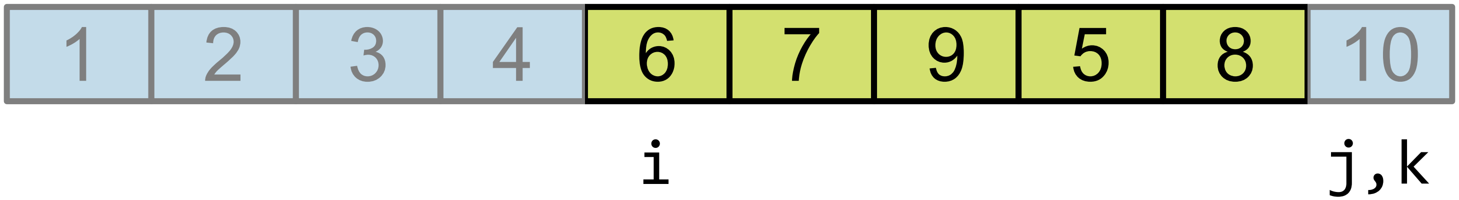

```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

1	2	3	4	6	7	9	5	8	10
				i					j,k

```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort



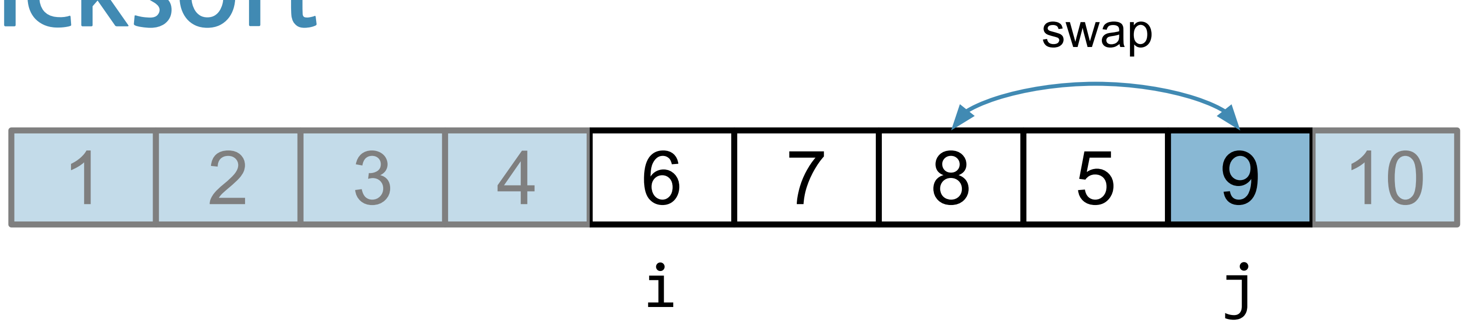

```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

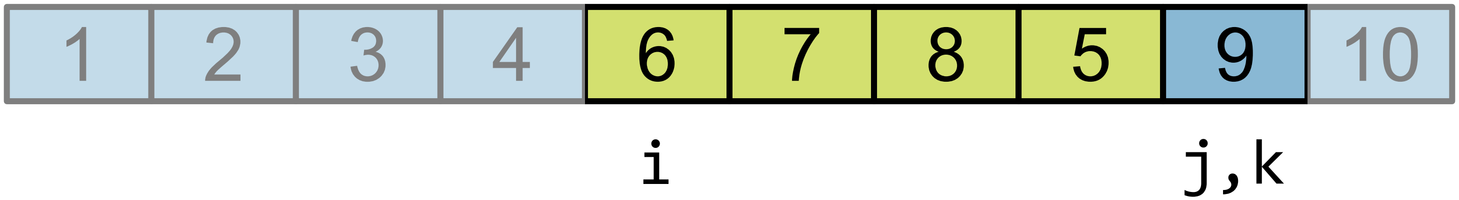
```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort




```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort




```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

1	2	3	4	6	7	8	5	9	10
				i				j,k	

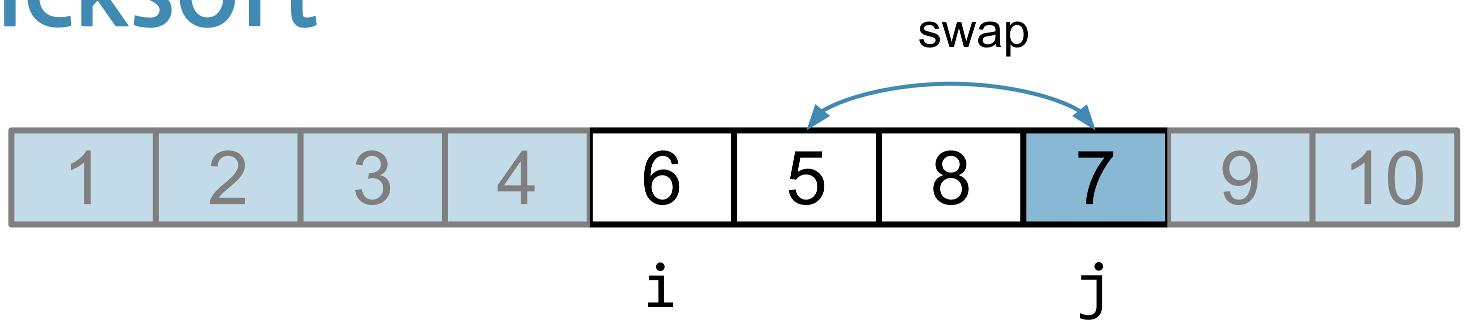
```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

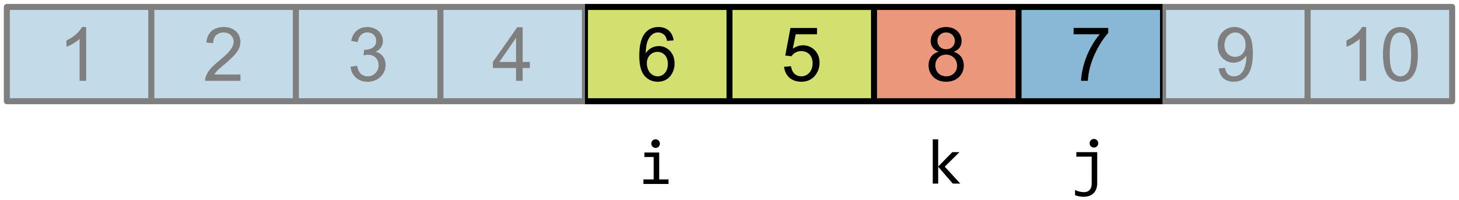
```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort



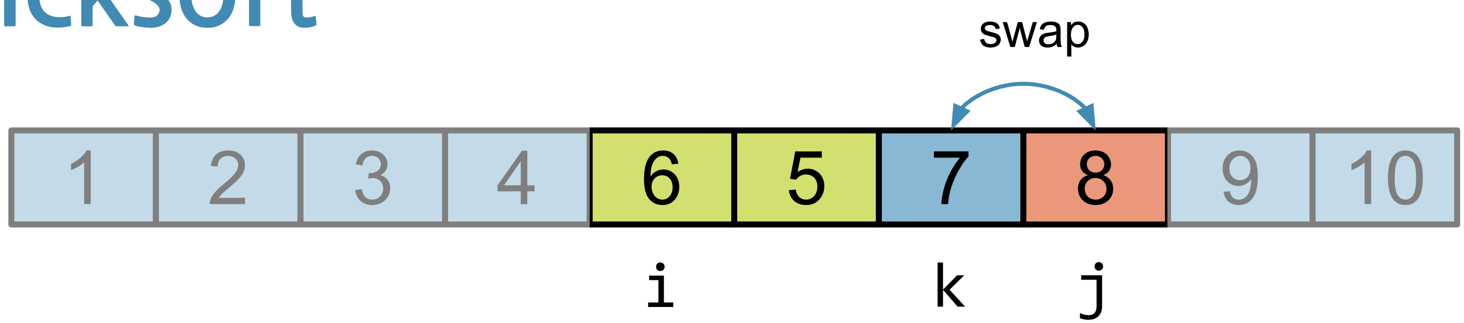

```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort




```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort




```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

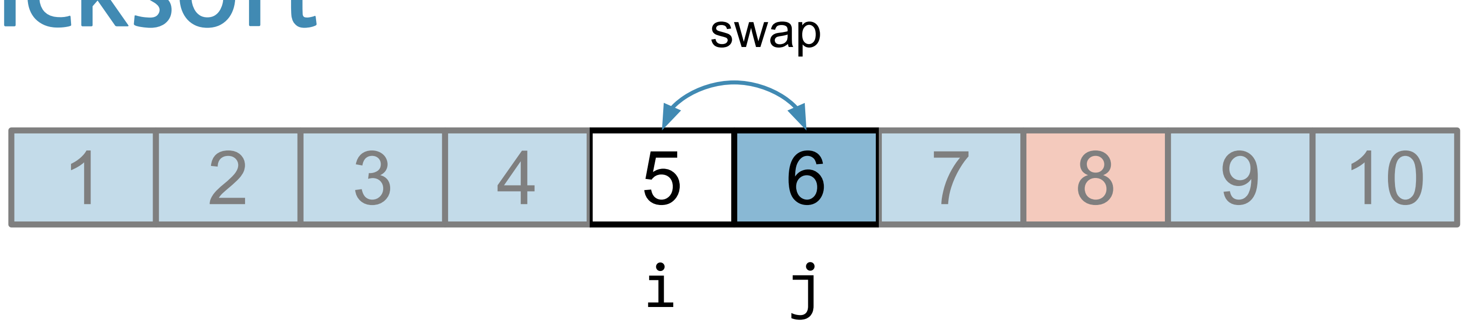
```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

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

```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort




```
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

1	2	3	4	5	6	7	8	9	10
				i	j,k				

```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort

1	2	3	4	5	6	7	8	9	10

```java
static <E extends Comparable<? super E>>
void qsort(E[] A, int i, int j) {
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition(A, i - 1, j, A[j]);
    swap(A, k, j);
    if (i < k - 1) qsort(A, i, k - 1);
    if (k + 1 < j) qsort(A, k + 1, j);
}
```

Quicksort Partition

10	1	9	3	6	7	2	5	8	4

l r

```
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition

10	1	9	3	6	7	2	5	8	4
l									r

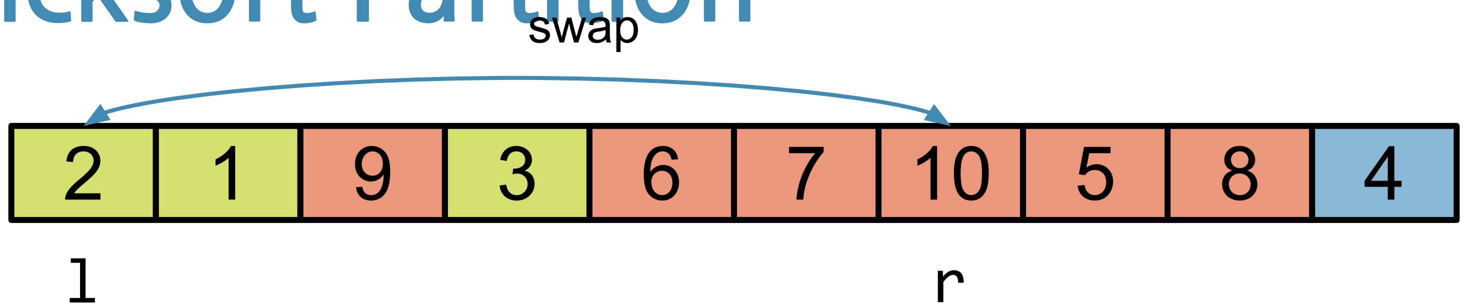
```java
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition

10	1	9	3	6	7	2	5	8	4
l						r			

```
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition




```
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition

2	1	9	3	6	7	10	5	8	4
		l				r			

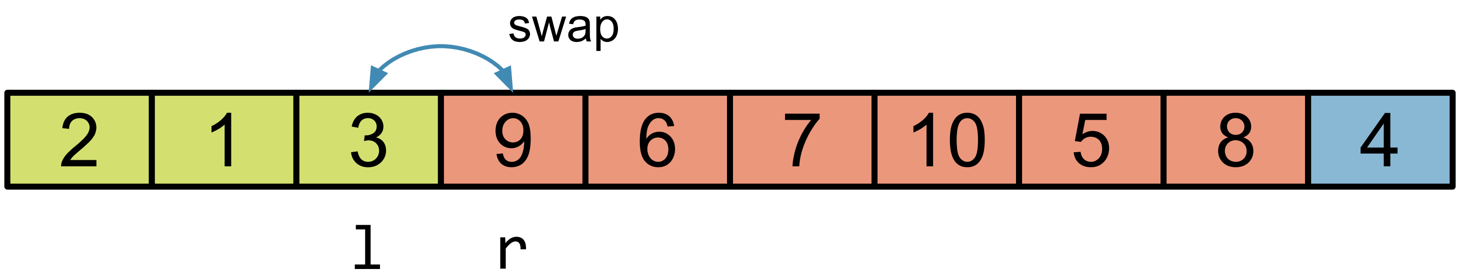
```java
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition

2	1	9	3	6	7	10	5	8	4
		l	r						

```
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition




```java
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition

2	1	3	9	6	7	10	5	8	4
			l,r						

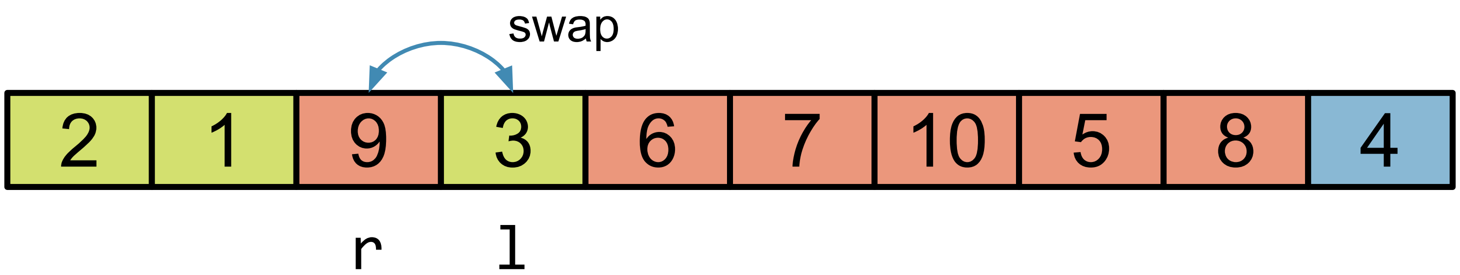
```
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition

2	1	3	9	6	7	10	5	8	4
		r	l						

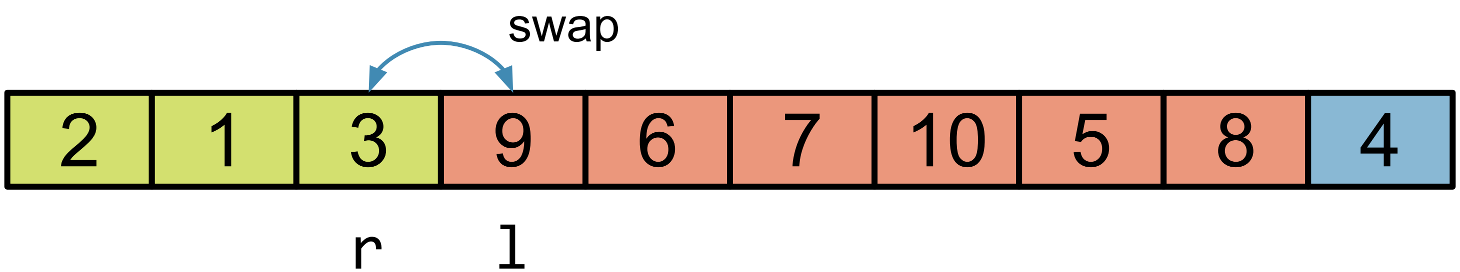
```java
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition




```
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition




```
static <E extends Comparable<? super E>>
int partition(E[] A, int l, int r, E pivot) {
    do {
        while (A[++l].compareTo(pivot) < 0);
        while ((r != 0) && (A[--r].compareTo(pivot) > 0));
        swap(A, l, r);
    } while (l < r);
    swap(A, l, r);
    return l;
}
```

Quicksort Partition

- Return value: pivot보다 큰 값들의 첫 번째 index
- 비용: $\Theta(n)$

Cost of Quicksort

- Best case: 항상 절반씩 분할되는 경우
- Worst case: pivot보다 모든 값이 큰 경우 (혹은 작은 경우)
- Average case:

$$T(n) = cn + \frac{1}{n}\sum_{k=0}^{n-1}(T(k) + T(n-1-k))$$

$$T(0) = T(1) = c$$

  - 위의 재귀 관계를 풀어보면? (증명?)

$$T(n) = \Theta(n \log n)$$

Optimizations for Quicksort

- Optimizations for Quicksort:
  - Pivot을 더 잘 선택하기 (how?)
  - sublist의 크기가 작을 때 다른 알고리즘 사용하기
    - Quicksort는 n이 작을 때 느린 편 (why?)
    - n이 작은 sublist에서는 insertion sort나 selection sort 사용하기
  - 재귀 호출 없애기: e.g., use stack

What you need to know

- Merge sort
  - Main idea: ‘divide and conquer’
  - Cost analysis
  - Advantage of optimized merge sort
- Quicksort
  - Main idea: ‘divide and conquer’
  - Cost analysis

Questions?