Advanced Sorting Algorithms (Merge Sort)

Data Structures CS218

Overview

- A quick review to the time complexity of elementary sorting techniques
- Merge sort
  - A recursive divide and conquer algorithm
  - Merging two lists
  - Implementation details

Properties of Sorting Algorithms

Algorithm	Adaptive	Stable	In-place
Bubble Sort	No	Yes	Yes
Selection Sort	No	No	Yes
Insertion Sort	Yes	Yes	Yes
Shell Sort	Yes	No	Yes

Time complexity

	Best	Average	Worst
Bubble Sort	O(n)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Insertion Sort	O(n)	O(n^2)	O(n^2)
Merge Sort	O(nlog(n))	O(nlog(n))	O(nlog(n))

Merge Sort

The merge sort algorithm is defined recursively:

- If the list is of size 1, it is sorted—we are done;
- Otherwise:
    - Divide an unsorted list into two sub-lists,
    - Sort each sub-list recursively using merge sort, and
    - Merge the two sorted sub-lists into a single sorted list

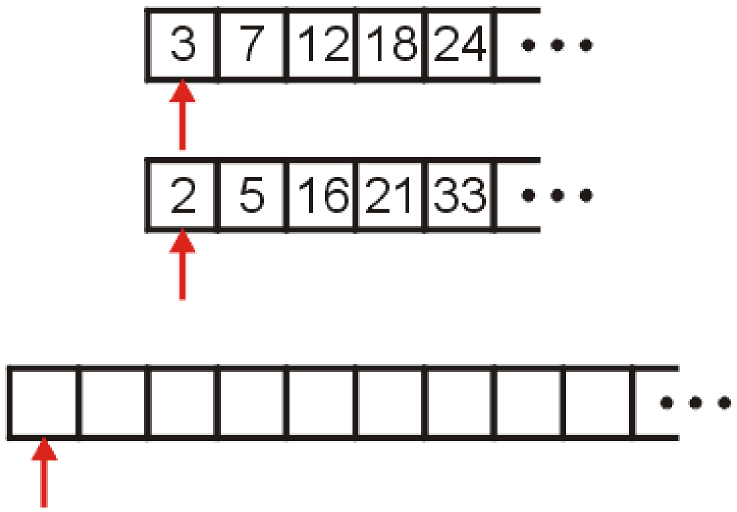
This is the first significant *divide-and-conquer* algorithm we will see

Question: How quickly can we recombine the two sub-lists into a single sorted list?

Merging Example

Consider the two sorted arrays and an empty array

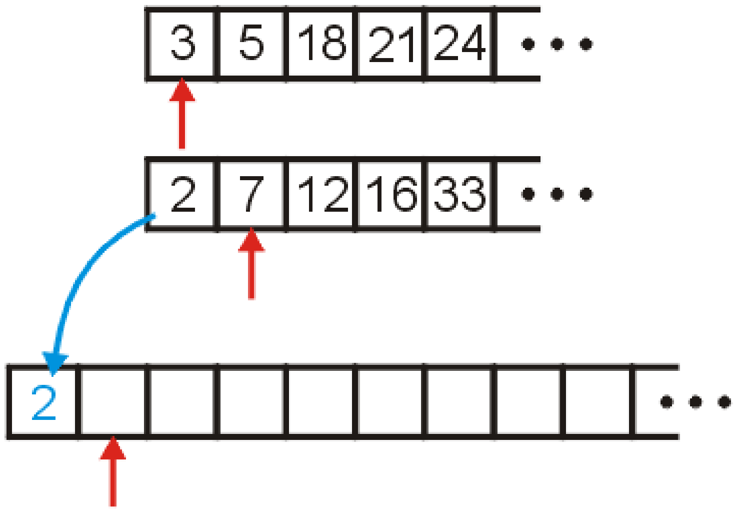
Define three indices at the start of each array



Merging Example

We compare 2 and 3: 2 < 3

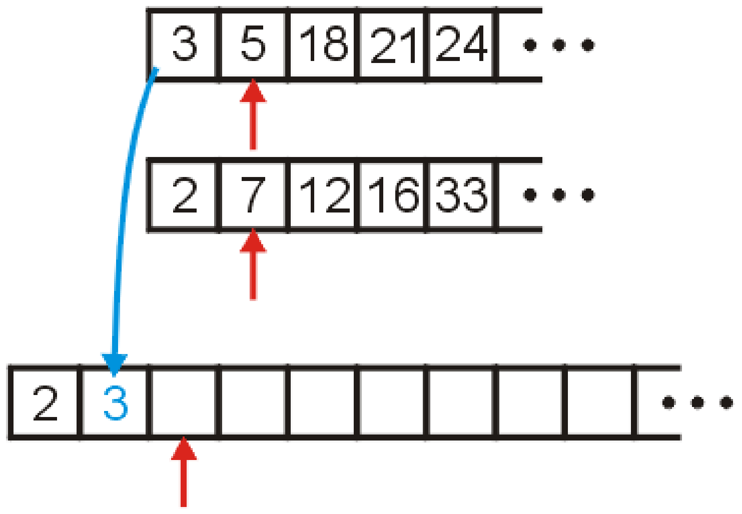
- Copy 2 down
- Increment the corresponding indices



Merging Example

We compare 3 and 7

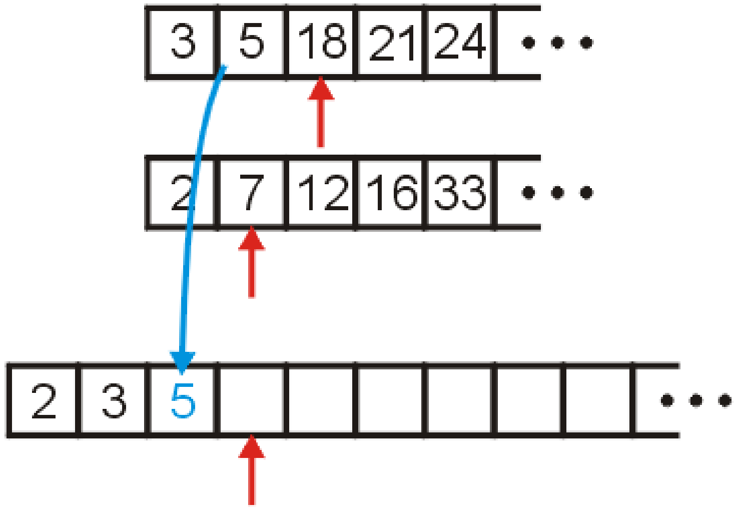
- Copy 3 down
- Increment the corresponding indices



Merging Example

We compare 5 and 7

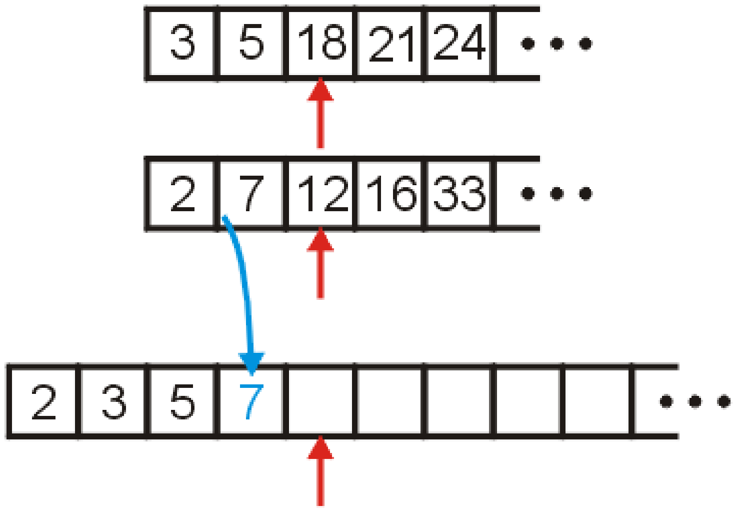
- Copy 5 down
- Increment the appropriate indices



Merging Example

We compare 18 and 7

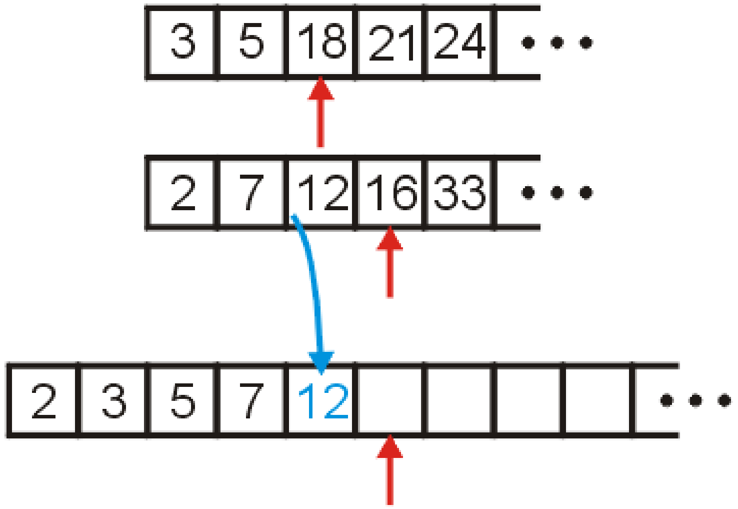
- Copy 7 down
- Increment...



Merging Example

We compare 18 and 12

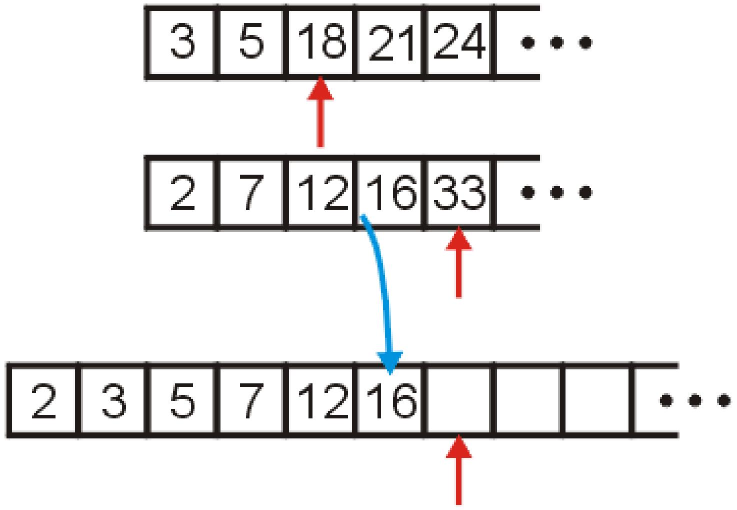
- – Copy 12 down
- – Increment...



Merging Example

We compare 18 and 16

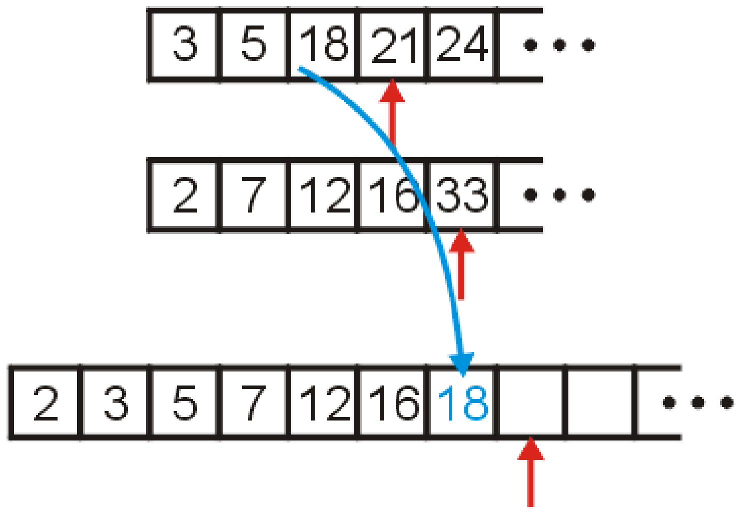
- Copy 16 down
- Increment...



Merging Example

We compare 18 and 33

- Copy 18 down
- Increment...

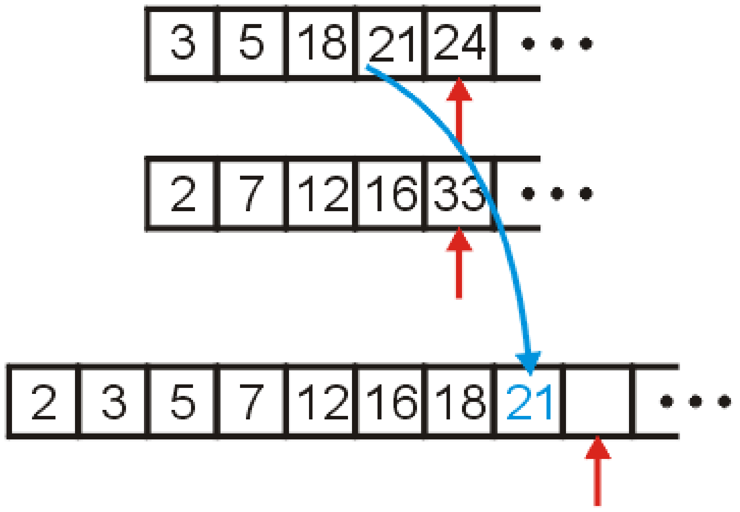


Merging Example

We compare 21 and 33

– Copy 21 down

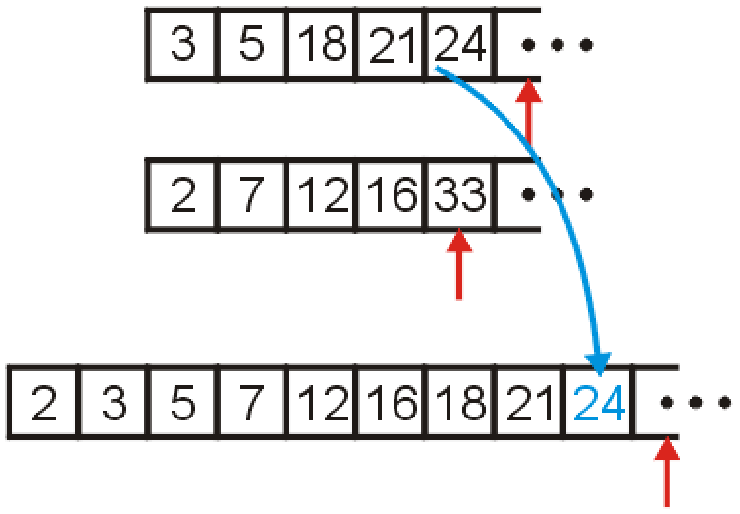
– Increment...



Merging Example

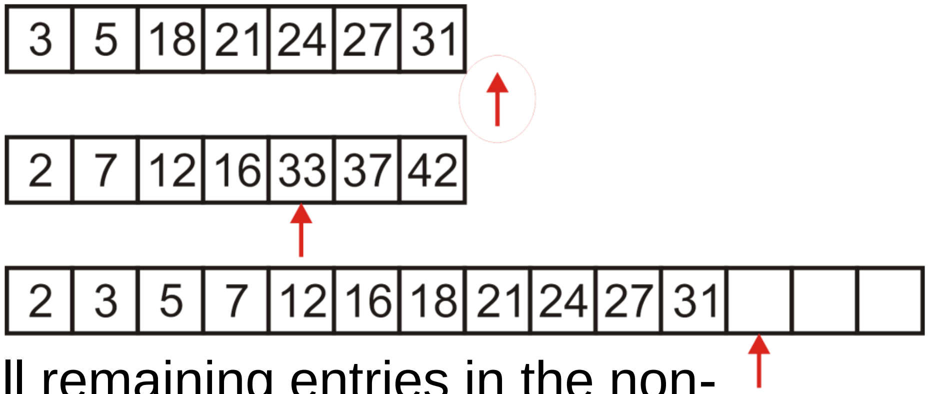
We compare 24 and 33

- Copy 24 down
- Increment...

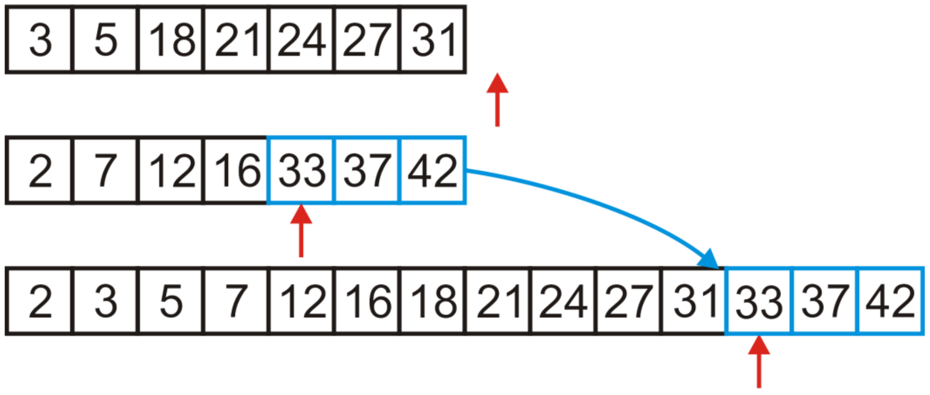


Merging Example

We would continue until we have passed beyond the limit of one of the two arrays



After this, we simply copy over all remaining entries in the non-empty array



Merging Two Lists

Programming a merge is straight-forward:

- the sorted arrays, `array1` and `array2`, are of size `n1` and `n2`, respectively, and
- we have an empty array, `arrayout`, of size `n1 + n2`

Define three variables

```
    int i1 = 0, i2 = 0, k = 0;
```

which index into these three arrays

Merging Two Lists

We can then run the following loop:

```
int i1 = 0, i2 = 0, k = 0;

while ( i1 < n1 && i2 < n2 ) {
    if ( array1[i1] < array2[i2] ) {
        arrayout[k] = array1[i1];
        ++i1;
    } else {
        arrayout[k] = array2[i2];
        ++i2;
    }
    ++k;
}
```

Merging Two Lists

We’re not finished yet, we have to empty out the remaining array

```
for ( ; i1 < n1; ++i1, ++k ) {
    arrayout[k] = array1[i1];
}

for ( ; i2 < n2; ++i2, ++k ) {
    arrayout[k] = array2[i2];
}
```

The Algorithm

The algorithm:

- Split the list into two approximately equal sub-lists
- Recursively call merge sort on both sub lists
- Merge the resulting sorted lists

Implementation

The actual body is quite small:

```
void merge_sort( Type *array, int first, int last ) {
    if ( last - first <= N ) {
        insertion_sort( array, first, last );
    } else {
        int midpoint = (first + last)/2;

        merge_sort( array, first, midpoint );
        merge_sort( array, midpoint, last );
        merge( array, first, midpoint, last );
    }
}
```

Example

Consider the following is of unsorted array of 25 entries

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

We will call insertion sort if the list being sorted of size $N = 6$ or less

Example

We call `merge_sort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

```
merge_sort( array,  0, 25 )
```

Example

We are calling `merge_sort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

First, $25 - 0 > 6$, so find the midpoint and call `merge_sort` recursively

```
midpoint = (0 + 25)/2; // == 12
merge_sort( array, 0, 12 );
```

```
merge_sort( array,  0, 25 )
```

Example

We are now executing `merge_sort( array, 0, 12 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

First, $12 - 0 > 6$, so find the midpoint and call `merge_sort` recursively

```
midpoint = (0 + 12)/2; // == 6
merge_sort( array, 0, 6 );
```

```
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

We are now executing `merge_sort( array, 0,  6 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

Now, $6 - 0 \le 6$, so find we call insertion sort

```
merge_sort( array,  0,  6 )
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

Insertion sort just sorts the entries from `0` to `5`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

```
insertion_sort( array, 0, 6 )
merge_sort( array,  0,  6 )
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

Insertion sort just sorts the entries from `0` to `5`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	35	48	49	61	77	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

– This function call completes and so we exit

```
insertion_sort( array, 0, 6 )
merge_sort( array,  0,  6 )
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

This call to `merge_sort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	35	48	49	61	77	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

```
merge_sort( array,  0,  6 )
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

We return to continue executing `merge_sort( array, 0, 12 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	35	48	49	61	77	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

We continue calling

```
midpoint = (0 + 12)/2; // == 6
merge_sort( array, 0, 6 );
merge_sort( array, 6, 12 );
```

```
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

We are now executing `merge_sort( array, 6,  12 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	35	48	49	61	77	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

Now, $12 - 6 \le 6$, so find we call insertion sort

```
merge_sort( array,  6, 12 )

merge_sort( array,  0, 12 )

merge_sort( array,  0, 25 )
```

Example

Insertion sort just sorts the entries from 6 to 11

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	35	48	49	61	77	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

```
insertion_sort( array, 6, 12 )

merge_sort( array,  6, 12 )

merge_sort( array,  0, 12 )

merge_sort( array,  0, 25 )
```

Example

Insertion sort just sorts the entries from 6 to 11

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	35	48	49	61	77	3	23	37	73	89	95	57	99	17	32	94	28	15	55	7	51	88	97	62

- This function call completes and so we exit

```
insertion_sort( array, 6, 12 )
merge_sort( array,  6, 12 )
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

This call to `merge_sort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	35	48	49	61	77	3	23	37	73	89	95	57	99	17	32	94	28	15	55	7	51	88	97	62

```
merge_sort( array,  6, 12 )
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

We return to continue executing `merge_sort( array, 0, 12 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	35	48	49	61	77	3	23	37	73	89	95	57	99	17	32	94	28	15	55	7	51	88	97	62

We continue calling

```
midpoint = (0 + 12)/2; // == 6
merge_sort( array, 0, 6 );
merge_sort( array, 6, 12 );
merge( array, 0, 6, 12 );
```

```
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

We are executing `merge( array, 0, 6, 12 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	35	48	49	61	77	3	23	37	73	89	95	57	99	17	32	94	28	15	55	7	51	88	97	62

These two sub-arrays are merged together

```
merge( array, 0, 6, 12 )
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

We are executing `merge( array, 0, 6, 12 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	57	99	17	32	94	28	15	55	7	51	88	97	62

These two sub-arrays are merged together

- This function call exists

```
merge( array, 0, 6, 12 )

merge_sort( array,  0, 12 )

merge_sort( array,  0, 25 )
```

Example

We return to executing `merge_sort( array, 0, 12 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	57	99	17	32	94	28	15	55	7	51	88	97	62

We are finished calling this function as well

```
    midpoint = (0 + 12)/2; // == 6
    merge_sort( array, 0, 6 );
    merge_sort( array, 6, 12 );
    merge( array, 0, 6, 12 );
```

Consequently, we exit

```
merge_sort( array,  0, 12 )
merge_sort( array,  0, 25 )
```

Example

We return to executing `merge_sort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	57	99	17	32	94	28	15	55	7	51	88	97	62

We continue calling

```
midpoint = (0 + 25)/2; // == 12
merge_sort( array, 0, 12 );
merge_sort( array, 12, 25 );
```

```
merge_sort( array,  0, 25 )
```

Example

We are now executing `merge_sort( array, 12, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	57	99	17	32	94	28	15	55	7	51	88	97	62

First, $25 - 12 > 6$, so find the midpoint and call `merge_sort` recursively

```
midpoint = (12 + 25)/2; // == 18
merge_sort( array, 12, 18 );
```

```
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

We are now executing `merge_sort( array, 12, 18 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	57	99	17	32	94	28	15	55	7	51	88	97	62

Now, $18 - 12 \le 6$, so find we call insertion sort

```
merge_sort( array, 12, 18 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

Insertion sort just sorts the entries from 12 to 17

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	57	99	17	32	94	28	15	55	7	51	88	97	62

```
insertion_sort( array, 12, 18 )
merge_sort( array, 12, 18 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

Insertion sort just sorts the entries from 12 to 17

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	15	55	7	51	88	97	62

– This function call completes and so we exit

```
insertion_sort( array, 12, 18 )

merge_sort( array, 12, 18 )

merge_sort( array, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

This call to `merge_sort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	15	55	7	51	88	97	62

```
merge_sort( array, 12, 18 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

We return to continue executing `merge_sort( array, 12, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	15	55	7	51	88	97	62

We continue calling

```
midpoint = (12 + 25)/2; // == 18
merge_sort( array, 12, 18 );
merge_sort( array, 18, 25 );
```

```
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

We are now executing `merge_sort( array, 18, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	15	55	7	51	88	97	62

First, $25 - 18 > 6$, so find the midpoint and call `merge_sort` recursively

```
    midpoint = (18 + 25)/2; // == 21
    merge_sort( array, 18, 21 );
```

```
merge_sort( array, 18, 25 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

We are now executing `merge_sort( array, 18, 21 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	15	55	7	51	88	97	62

Now, $21 - 18 \le 6$, so find we call insertion sort

```
merge_sort( array, 18, 21 )
merge_sort( array, 18, 25 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

Insertion sort just sorts the entries from 18 to 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	15	55	7	51	88	97	62

```
insertion_sort( array, 18, 21 )

merge_sort( array, 18, 21 )

merge_sort( array, 18, 25 )

merge_sort( array, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

Insertion sort just sorts the entries from `18` to `20`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	55	51	88	97	62

– This function call completes and so we exit

```
insertion_sort( array, 18, 21 )
merge_sort( array, 18, 21 )
merge_sort( array, 18, 25 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

This call to `merge_sort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	55	51	88	97	62

```
merge_sort( array, 18, 21 )
merge_sort( array, 18, 25 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

We return to executing `merge_sort( array, 18, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	55	51	88	97	62

We continue calling

```
midpoint = (18 + 25)/2; // == 21
merge_sort( array, 18, 21 );
merge_sort( array, 21, 25 );
```

```
merge_sort( array, 18, 25 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

We are now executing `merge_sort( array, 21, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	55	51	88	97	62

Now, $25 - 21 \leq 6$, so find we call insertion sort

```
merge_sort( array, 21, 25 )
merge_sort( array, 18, 25 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

Insertion sort just sorts the entries from 21 to 24

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	55	51	88	97	62

```
insertion_sort( array, 21, 25 )

merge_sort( array, 21, 25 )

merge_sort( array, 18, 25 )

merge_sort( array, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

Insertion sort just sorts the entries from 21 to 24

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	55	51	62	88	97

– This function call completes and so we exit

```
insertion_sort( array, 21, 25 )

merge_sort( array, 21, 25 )

merge_sort( array, 18, 25 )

merge_sort( array, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

This call to `merge_sort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	55	51	62	88	97

```
merge_sort( array, 21, 25 )
merge_sort( array, 18, 25 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

We return to continue executing `merge_sort( array, 18, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	55	51	62	88	97

We continue calling

```
midpoint = (18 + 25)/2; // == 21
merge_sort( array, 18, 21 );
merge_sort( array, 21, 25 );
merge( array, 18, 21, 25 );
```

```
merge_sort( array, 18, 25 )
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

We are executing `merge( array, 18, 21, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	55	51	62	88	97

These two sub-arrays are merged together

```
merge( array, 18, 21, 25 )

merge_sort( array, 18, 25 )

merge_sort( array, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

We are executing `merge( array, 18, 21, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	51	55	62	88	97

These two sub-arrays are merged together

- This function call exists

```
merge( array, 18, 21, 25 )

merge_sort( array, 18, 25 )

merge_sort( array, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

We return to executing `merge_sort( array, 18, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	51	55	62	88	97

We are finished calling this function as well

```
    midpoint = (18 + 25)/2; // == 21
    merge_sort( array, 18, 21 );
    merge_sort( array, 21, 25 );
    merge( array, 18, 21, 25 );
```

Consequently, we exit

```
merge_sort( array, 18, 25 )

merge_sort( array, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

We return to continue executing `merge_sort( array, 12, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	51	55	62	88	97

We continue calling

```
midpoint = (12 + 25)/2; // == 18
merge_sort( array, 12, 18 );
merge_sort( array, 18, 25 );
merge( array, 12, 18, 25 );
```

```
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

We are executing `merge( array, 12, 18, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	17	28	32	57	94	99	7	15	51	55	62	88	97

These two sub-arrays are merged together

```
merge( array, 12, 18, 25 )

merge_sort( array, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

We are executing `merge( array, 12, 18, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	7	15	17	28	32	51	55	57	62	88	94	97	99

These two sub-arrays are merged together

– This function call exists

```
merge( array, 12, 18, 25 )

merge_sort( array, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

We return to executing `merge_sort( array, 12, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	7	15	17	28	32	51	55	57	62	88	94	97	99

We are finished calling this function as well

```
midpoint = (12 + 25)/2; // == 18
merge_sort( array, 12, 18 );
merge_sort( array, 18, 25 );
merge( array, 12, 18, 25 );
```

Consequently, we exit

```
merge_sort( array, 12, 25 )
merge_sort( array,  0, 25 )
```

Example

We return to continue executing `merge_sort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	7	15	17	28	32	51	55	57	62	88	94	97	99

We continue calling

```
midpoint = (0 + 25)/2; // == 12
merge_sort( array, 0, 12 );
merge_sort( array, 12, 25 );
merge( array, 0, 12, 25 );
```

```
merge_sort( array,  0, 25 )
```

Example

We are executing `merge( array, 0, 12, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	13	23	35	37	48	49	61	73	77	89	95	7	15	17	28	32	51	55	57	62	88	94	97	99

These two sub-arrays are merged together

```
merge( array, 0, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

We are executing `merge( array, 0, 12, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

These two sub-arrays are merged together

– This function call exists

```
merge( array, 0, 12, 25 )

merge_sort( array,  0, 25 )
```

Example

We return to executing `merge_sort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

We are finished calling this function as well

```
midpoint = (0 + 25)/2; // == 12
merge_sort( array, 0, 12 );
merge_sort( array, 12, 25 );
merge( array, 0, 12, 25 );
```

Consequently, we exit

```
merge_sort( array,  0, 25 )
```

Example

The array is now sorted

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

Run-time Summary

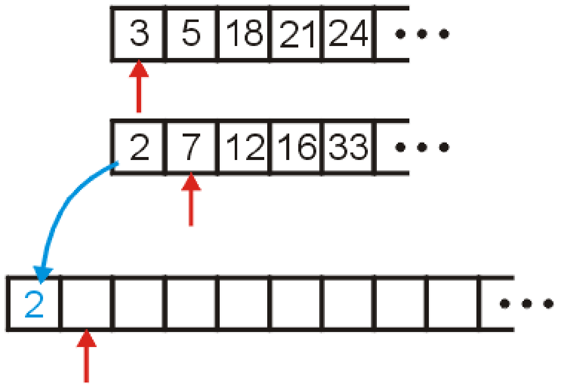
The following table summarizes the run-times of merge sort

Case	Run Time	Comments
Worst	$\Theta(n \ln(n))$	No worst case
Average	$\Theta(n \ln(n))$	
Best	$\Theta(n \ln(n))$	No best case

Why is it not $\mathrm{O}(n^2)$

When we are merging, we are comparing values

- What operation prevents us from performing $\mathrm{O}(n^2)$ comparisons?
- During the merging process, if 2 came from the second half, it was only compared to 3 and it was not compared to any other of the other $n - 1$ entries in the first array



- In this case, we remove n inversions with one comparison

Properties

- Merge sort requires an additional array
  - Not inplace
- It is adaptive and stable

Summary

This topic covered merge sort:

- Divide an unsorted list into two equal or nearly equal sub lists,
- Sorts each of the sub lists by calling itself recursively, and then
- Merges the two sub lists together to form a sorted list