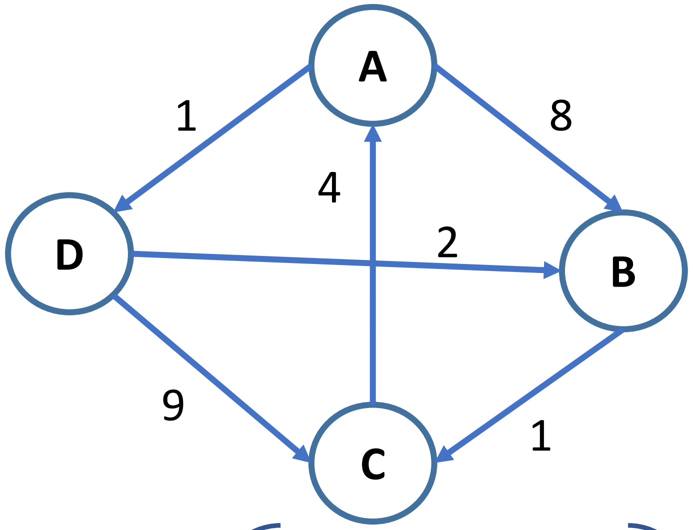
CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

Exercise: Floyd-Warshall → APSP

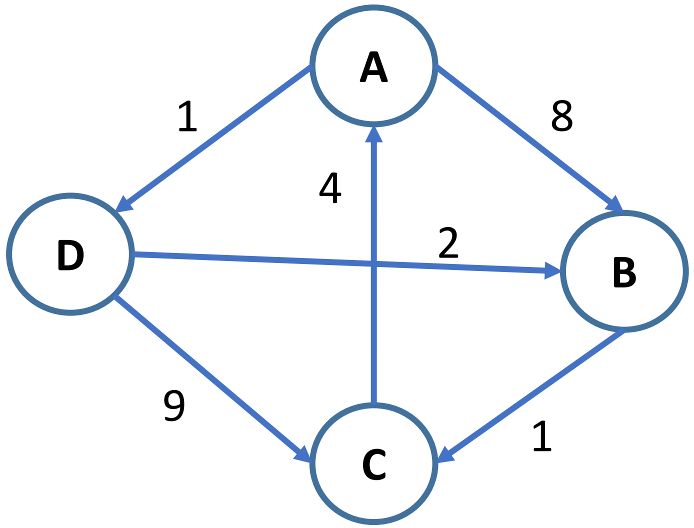


$$D^{(4)} = \begin{pmatrix} 0 & & & \\ & 0 & & \\ & & 0 & \\ & & & 0 \end{pmatrix} \quad \pi^{(4)} = \begin{pmatrix} \text{NIL} & & & \\ & \text{NIL} & & \\ & & \text{NIL} & \\ & & & \text{NIL} \end{pmatrix}$$

$$D^{(0)} = W \begin{pmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{pmatrix} \quad \pi^{(0)} = \begin{pmatrix} \text{NIL} & \text{A} & \text{NIL} & \text{A} \\ \text{NIL} & \text{NIL} & \text{B} & \text{NIL} \\ \text{C} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{D} & \text{D} & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{pmatrix} \quad \pi^{(1)} = \begin{pmatrix} \text{NIL} & \text{A} & \text{NIL} & \text{A} \\ \text{NIL} & \text{NIL} & \text{B} & \text{NIL} \\ \text{C} & \text{A} & \text{NIL} & \text{A} \\ \text{NIL} & \text{D} & \text{D} & \text{NIL} \end{pmatrix}$$

Exercise: Floyd-Warshall → ASAP



$$D^{(4)} = W \begin{pmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{pmatrix} \qquad \pi^{(4)} = \begin{pmatrix} \text{NIL} & \text{D} & \text{D} & \text{A} \\ \text{C} & \text{NIL} & \text{B} & \text{C} \\ \text{C} & \text{D} & \text{NIL} & \text{A} \\ \text{C} & \text{D} & \text{B} & \text{NIL} \end{pmatrix}$$

Exercise: Master Theorem (1)

$T(n) = 0.5\,T\left(\frac{n}{2}\right) + cn^2$

Solution:

$T(n) = 2T\left(\frac{n}{0.8}\right) + n$

Solution:

$T(n) = 2T\left(\frac{n}{2}\right) + 1/\mathrm{n}$

Solution:

- The Master theorem method is used for solving the following types of recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k log^p n)$$

Where, a ≥ 1, b > 1, k ≥ 0 and *p is a real number*, *n* is the size of the problem, *a* is the number of sub problems in the recursion, and *n/b* is the size of each sub problem.

Using the three following cases, it solves $T(n)$

Case1: If $a > b^k$, then $T(n) = \Theta(n^{log_b^a})$

Case2: If $a = b^k$

a. If p > -1, then $T(n) = \Theta(n^{log_b^a}\, log^{p+1} n)$
b. If p = -1, then $T(n) = \Theta(n^{log_b^a}\, \log log\, n)$
c. If p < -1, then $T(n) = \Theta(n^{log_b^a})$

Case3: If $a < b^k$

a. If p ≥ 0, then $T(n) = \Theta(n^k log^p n)$
b. If p < 0, then $T(n) = \Theta(n^k)$

Exercise: Master Theorem

$T(n) = 3T\left(\frac{n}{2}\right) + n^2$

Solution:
a=3; b=2; k=2; p=0

$a < b^k$ => 3<4, p=0, then apply *case 3. a*

$\Theta(n^2)$

$T(n) = 6T\left(\frac{n}{3}\right) + n^2 logn$

Solution:
a=6; b=3; k=2; p=1

$a < b^k$ => 6<9,
p=1 then apply *case 3. a*

$\Theta(n^2\, logn)$

- The Master theorem method is used for solving the following types of recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k log^p n)$$

Using the three following cases, it solves $T(n)$

Case1: If $a > b^k$, then $T(n) = \Theta(n^{log_b^a})$

Case2: If $a = b^k$

a. If p > -1, then $T(n) = \Theta(n^{log_b^a}\, log^{p+1} n)$
b. If p = -1, then $T(n) = \Theta(n^{log_b^a}\, \log log\, n)$
c. If p < -1, then $T(n) = \Theta(n^{log_b^a})$

Case3: If $a < b^k$

a. If p ≥ 0, then $T(n) = \Theta(n^k log^p n)$
b. If p < 0, then $T(n) = \Theta(n^k)$

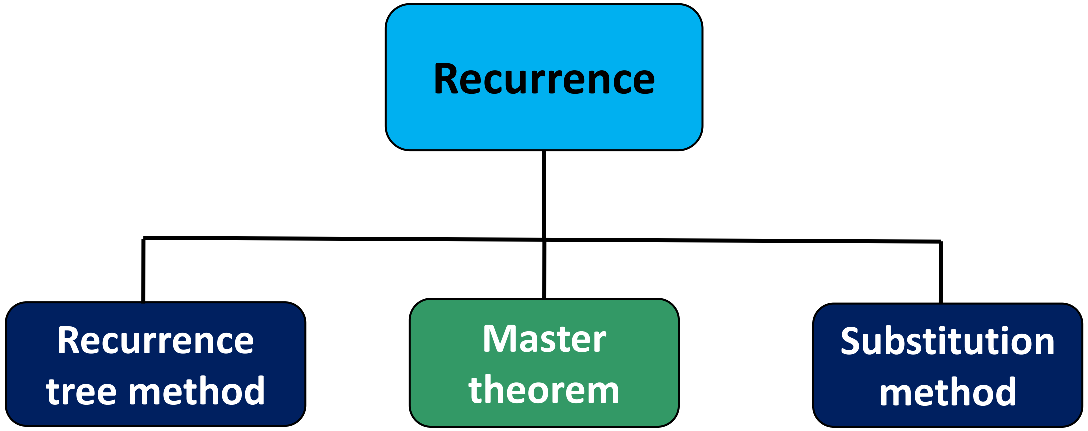
Master Theorem: Subtract and Conquer recurrences

- The Master theorem method is used for solving the following types of recurrence

$$T(n) = \begin{cases} c, & if\ n \le 1 \\ a\,T(n-b) + f(n), & if\ n > 1 \end{cases}$$

For some constants c, a > 0, b > 0, k ≥ 0, if f(n) is in O (n^k)

$$T(n) = \begin{cases} O(n^k), if\ a < 1 \\ O(n^{k+1}), if\ a = 1 \\ O\left(n^k a^{\frac{n}{b}}\right), if\ a > 1 \end{cases}$$



Exercise: Master Theorem: Subtract and Conquer recurrences (1)

$$T(n) = \begin{cases} 1, & if\ n \leq 0 \\ 3\,T(n-1), & if\ n > 0 \end{cases}$$

Solution: a = 3, b =1, k = 0, f(n) → O(n^0)

T(n) = O(3^n)

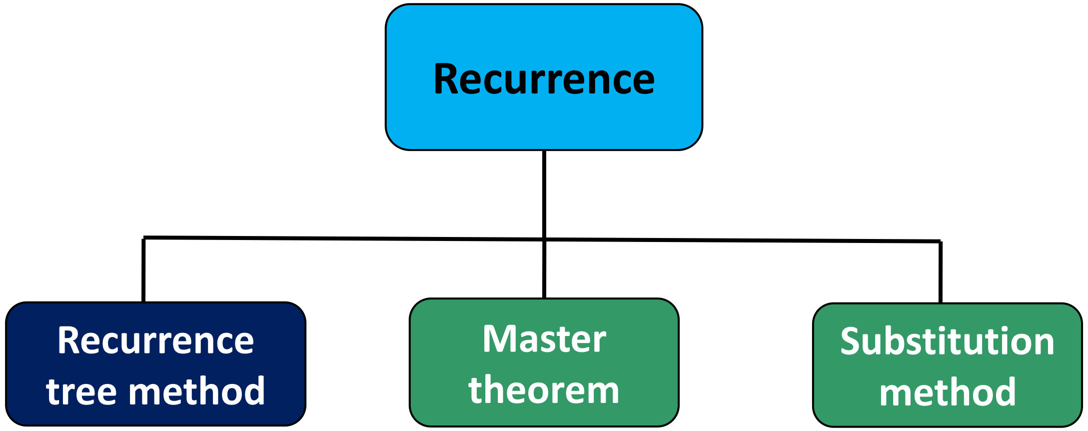
T (n) = 3T(n-1)

- The Master theorem method is used for solving the following types of recurrence

$$T(n) = \begin{cases} c, & if\ n \leq 1 \\ a\,T(n-b) + f(n), & if\ n > 1 \end{cases}$$

For some constants c, a > 0, b > 0, k ≥ 0, if f(n) is in O (n^k)

$$T(n) = \begin{cases} O(n^k), if\ a < 1 \\ O(n^{k+1}), if\ a = 1 \\ O(n^k a^{\frac{n}{b}}), if\ a > 1 \end{cases}$$



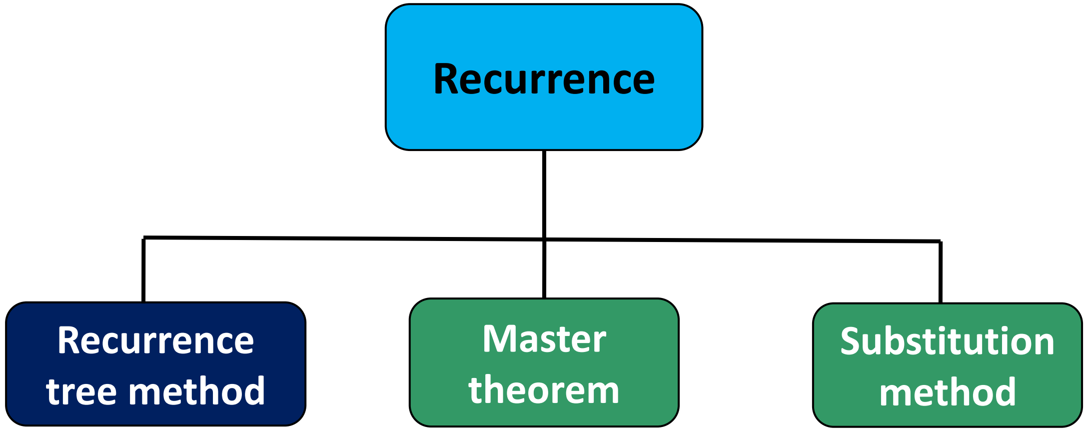
Exercise: Master Theorem: Subtract and Conquer recurrences (2)

- The Master theorem method is used for solving the following types of recurrence

$$T(n) = \begin{cases} c, & if\ n \le 1 \\ a\,T(n-b) + f(n), & if\ n > 1 \end{cases}$$

For some constants c, a > 0, b > 0, k ≥ 0, if f(n) is in O (n^k)

$$T(n) = \begin{cases} O(n^k), if\ a < 1 \\ O(n^{k+1}), if\ a = 1 \\ O\left(n^k a^{\frac{n}{b}}\right), if\ a > 1 \end{cases}$$



$$T(n) = \begin{cases} 1, & if\ n \le 0 \\ 2\,T(n-1) + 1, & if\ n > 0 \end{cases}$$

Solution: a = 2, b =1, k = 0, f(n) → O(n^0)

T(n) = O(2^n)

T (n) = 2T(n-1) +1

Sorting

What is Sorting?

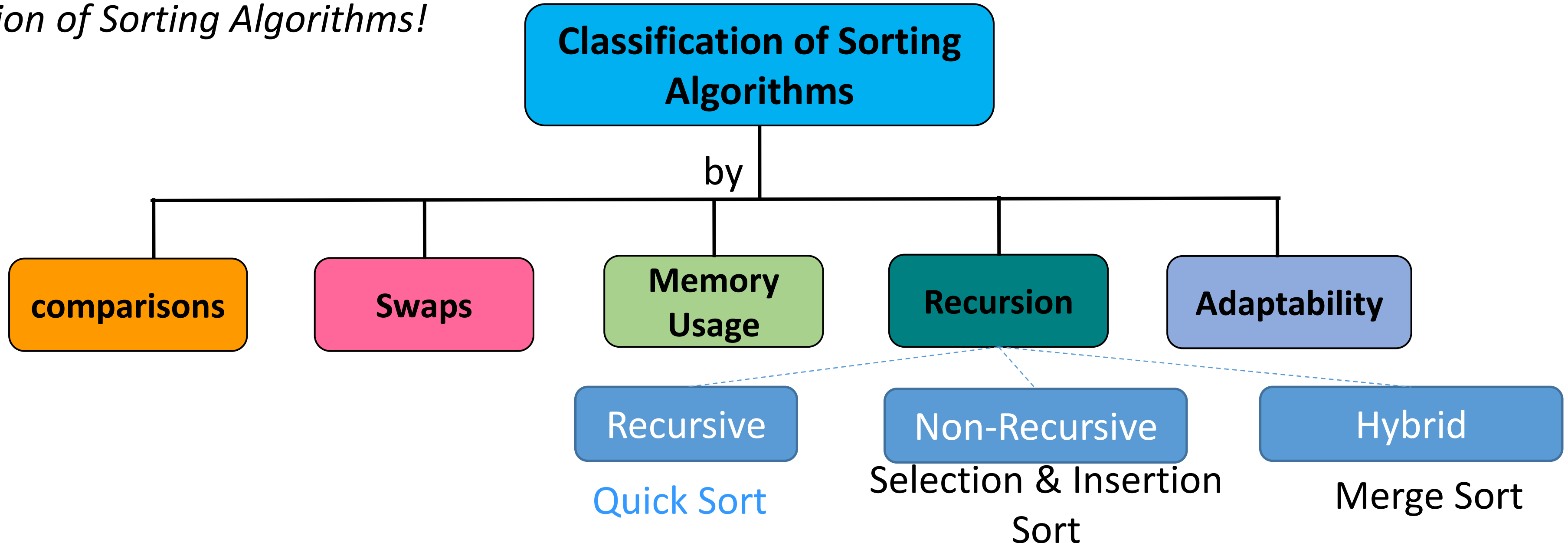
Sorting refers to arranging the elements of a list in a certain order [either ascending or descending]

Why is Sorting necessary or so important?

Sorting is one of important principles of algorithm design.

- Sorting helps to reduce the complexity of the problem.
- Sorting is used as a technique to reduce the search complexity (e.g., binary search)

Classification of Sorting Algorithms!

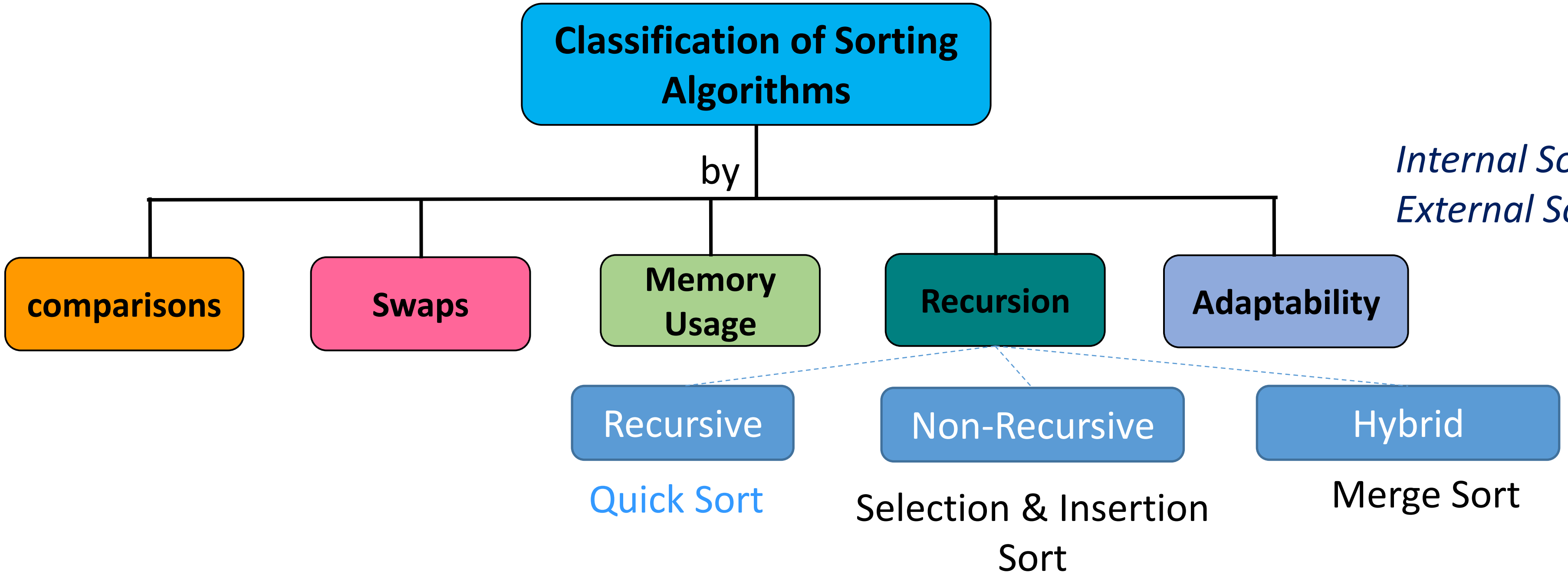


Internal Sort
External Sort

Sorting

- *Heap Sort*
- *Quick Sort*

- *Merge Sort*
- *Radix Sort*
- *Selection Sort*
- *Insertion Sort*



Internal Sort
External Sort

Divide-and-Conquer

- Design Principle: Merge sort follows the divide-and-Conquer design approach
- Divide-and-Conquer:
  - Divide: if the problem input size is too large → divide the problem into two or more smaller instances (i.e., sub-problems)
  - Conquer: the sub-problems are solved recursively by following divide-and-conquer approach again
  - Combine [or Merge]: the solutions of each sub-problems to obtain the solution for the original problem

Merge Sort Algorithm

- Divide: (i) if the given input size S has zero or one element, nothing need to be done;

  (ii) if the given input size S contains at least two elements, then divide and put them into two subsequences L (where L contain first half of the elements) and R (where R contains remaining elements)

L → Left
R → Right

- Conquer: Sort the sequence L and R using merge sort
- Combine [or Merge]: Put back the elements into S by merging the sorted sequences L and R into one sorted sequence

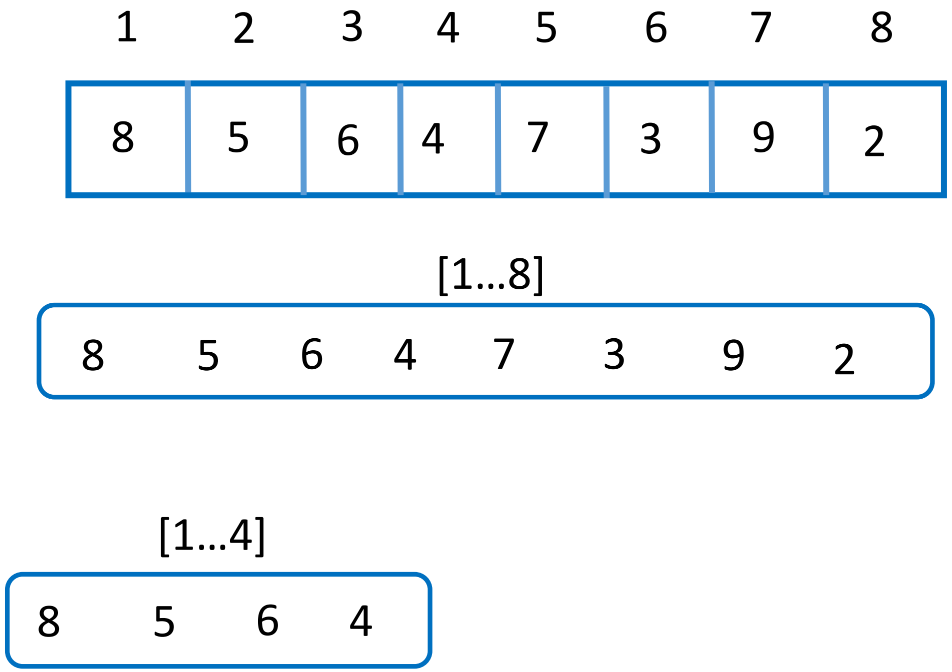
Merge Sort → Algorithm

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}

void merge(A, l, mid, h)
  // (i) Take the smallest elements of among
two sub-sequences A [l…mid] and A[mid+1…h]and
put into resulting sequence.

 //  (ii)Repeat the process until both are
sub-sequences are empty

}
```

1	2	3	4	5	6	7	8
8	5	6	4	7	3	9	2

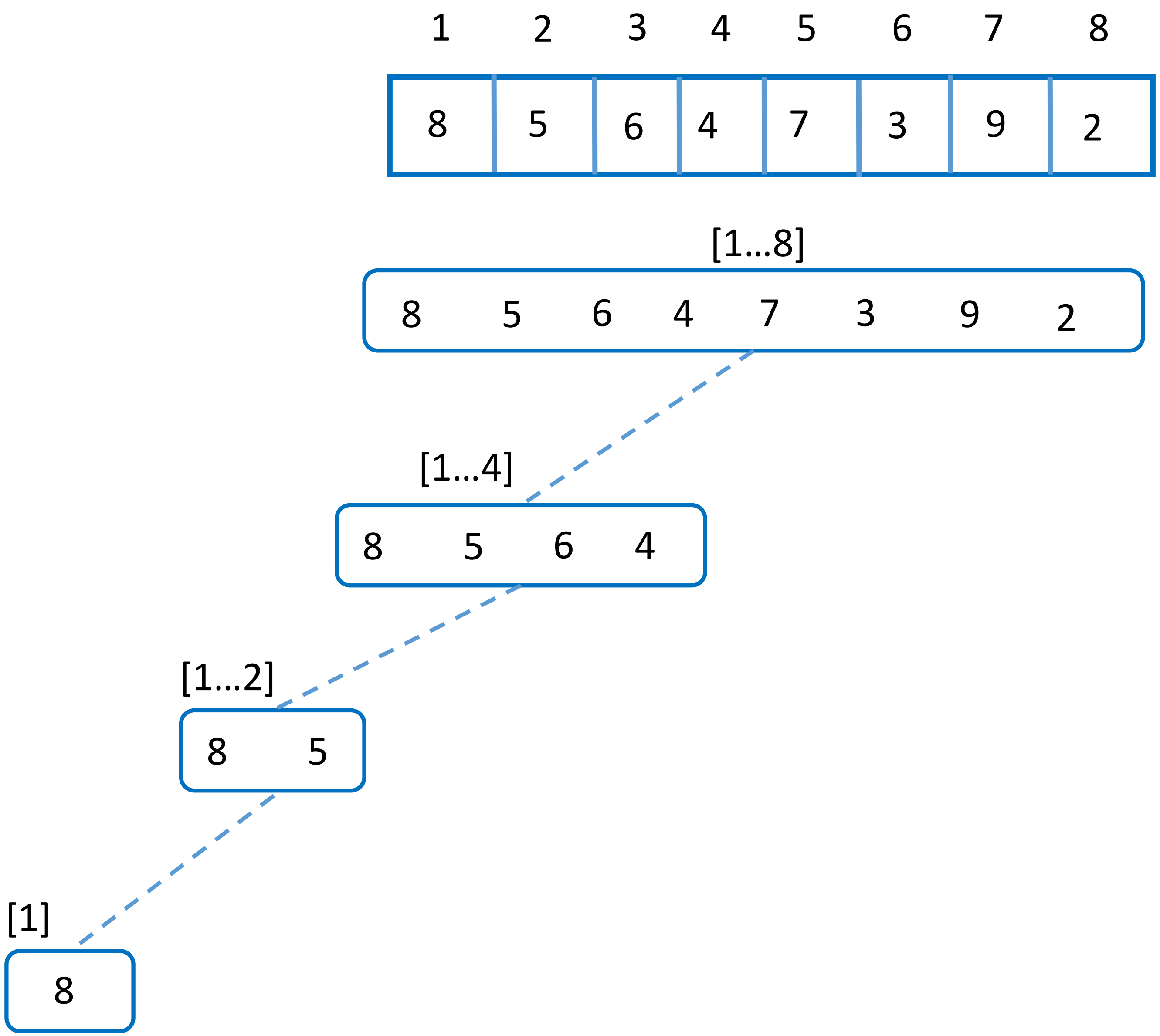
Example: Merge Sort

```
void mergesort(A, l, h)
{
   if (l < r)
      {
      mid = (l + h) / 2;
      mergesort(A, l, mid);
      mergesort(A, mid + 1, h);
      merge(A, l, mid, h);
      }
}
```



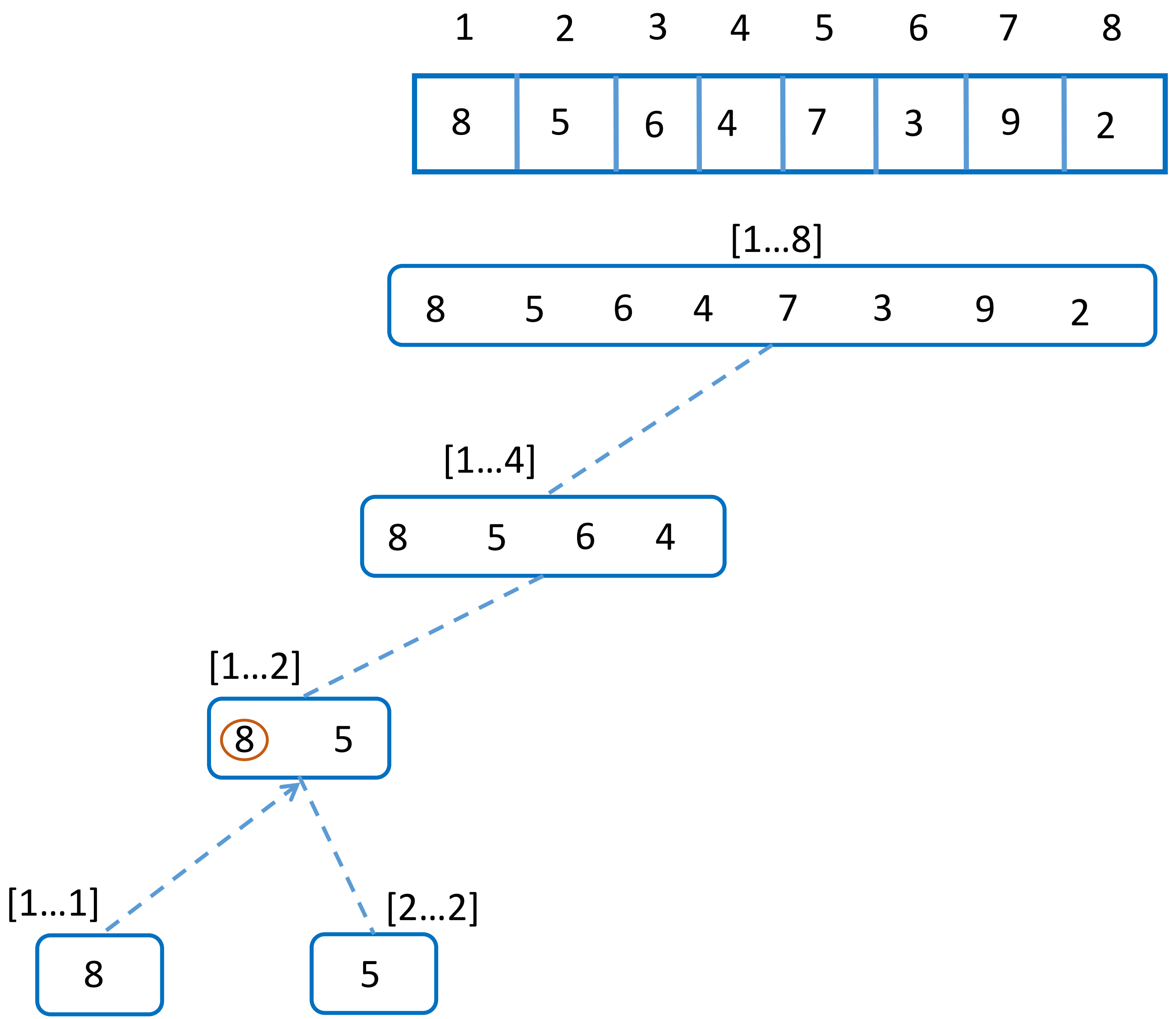
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



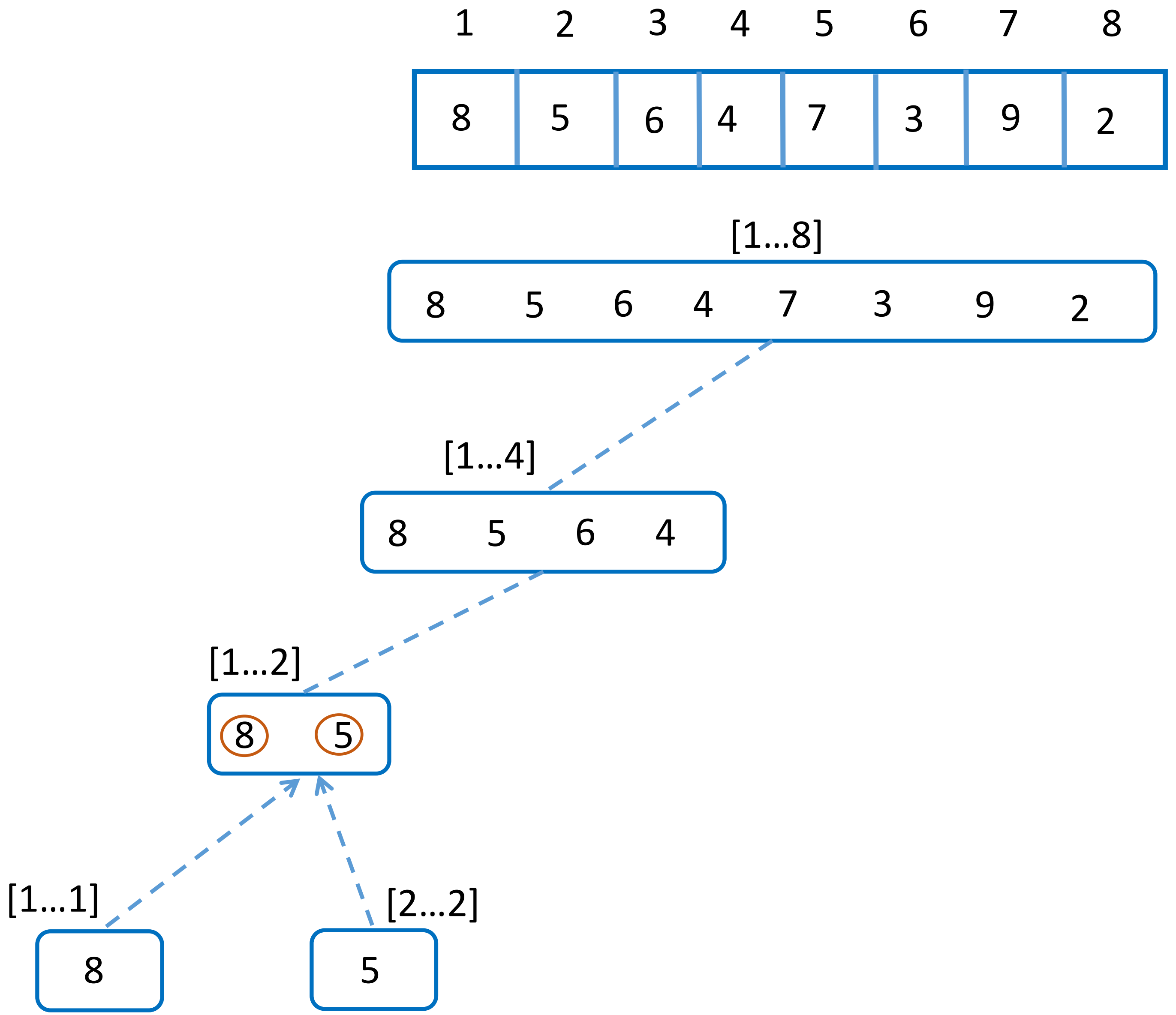
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



Example: Merge Sort

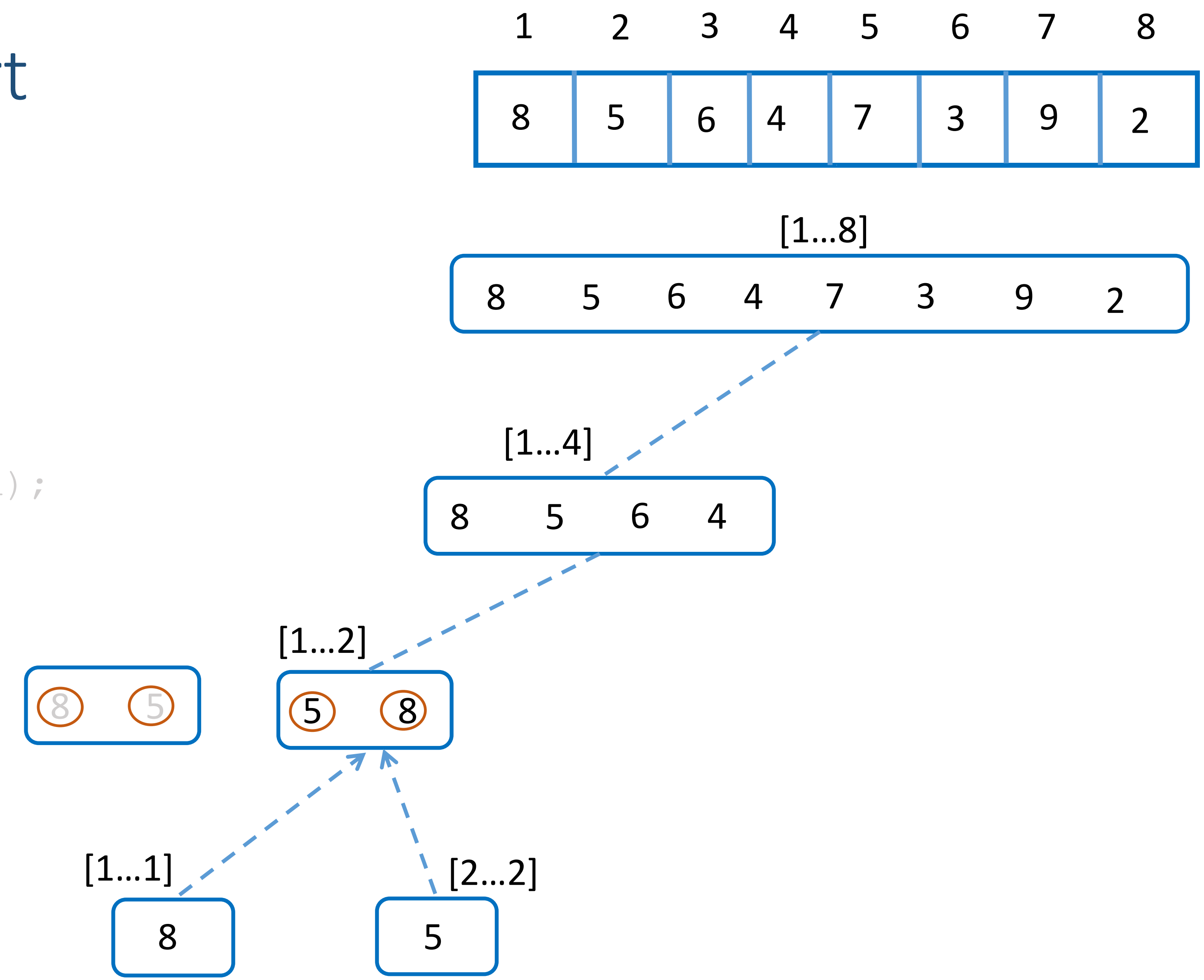
```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mergesort(A, l, mid);
    mid = (l + h) / 2;
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```

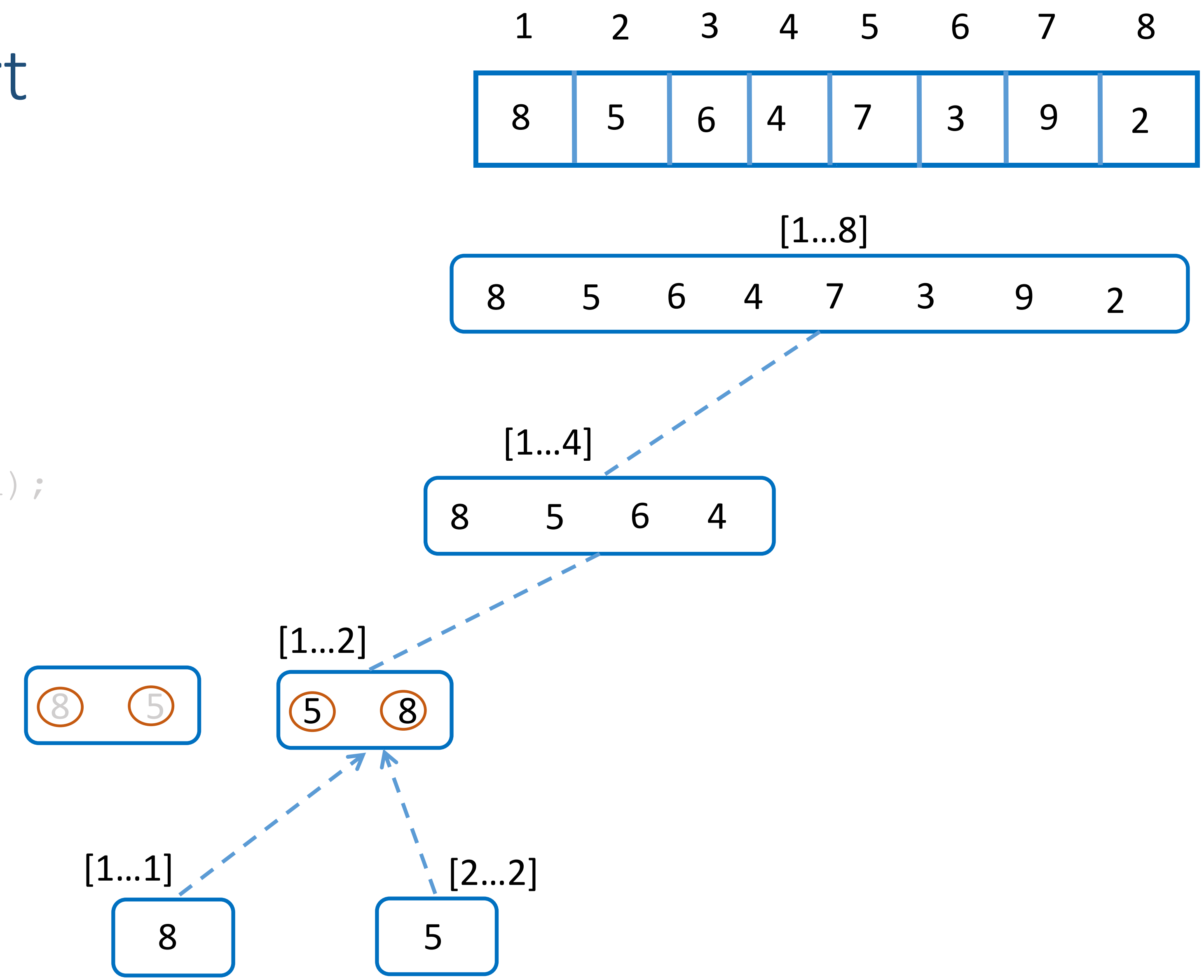
(i) Take the smallest elements of among two sub-sequences A [1…1] and A[2…2]and put into resulting sequence.



Example: Merge Sort

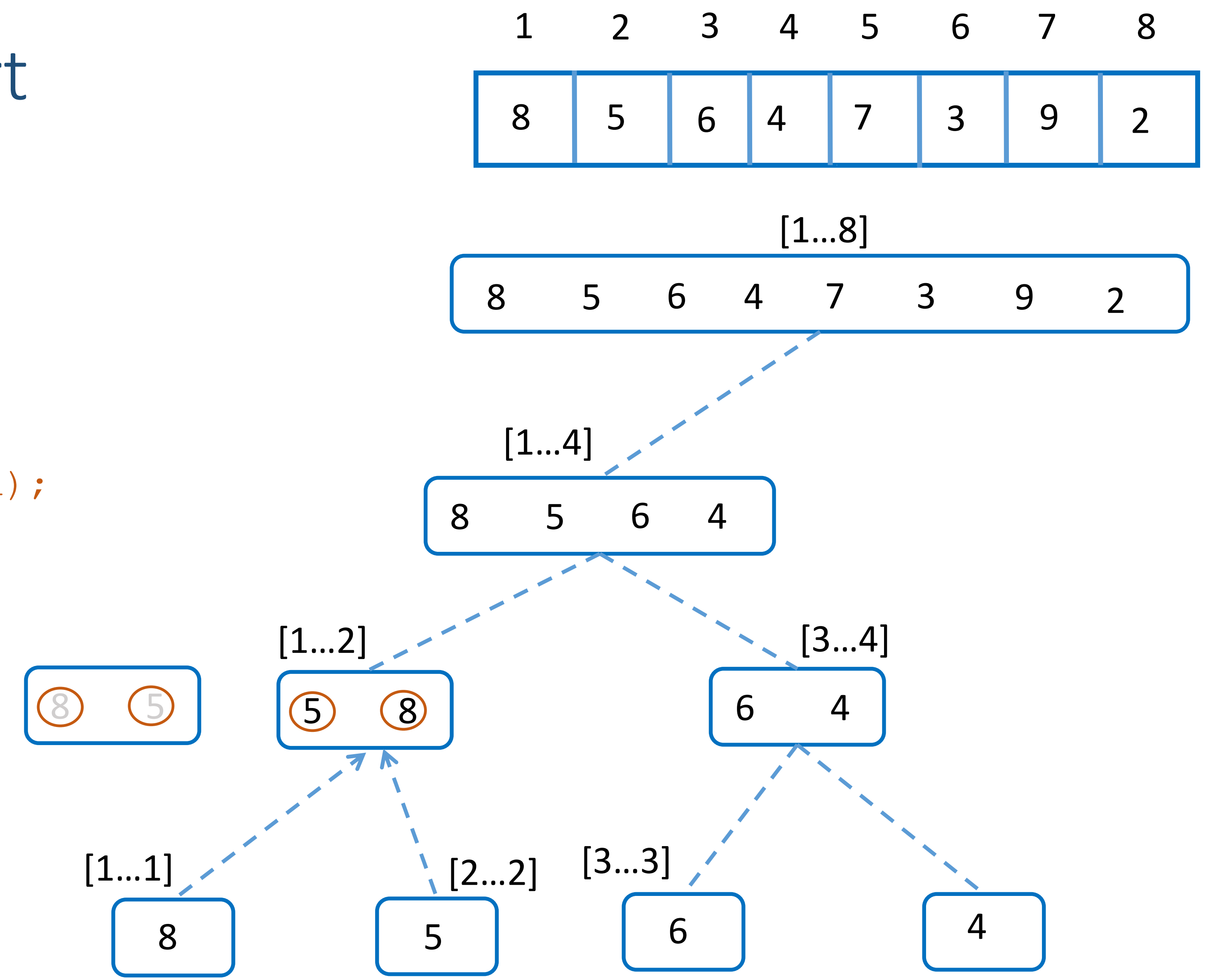
```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```

(i) Take the smallest elements of among two sub-sequences A [1…1] and A[2…2]and put into resulting sequence.



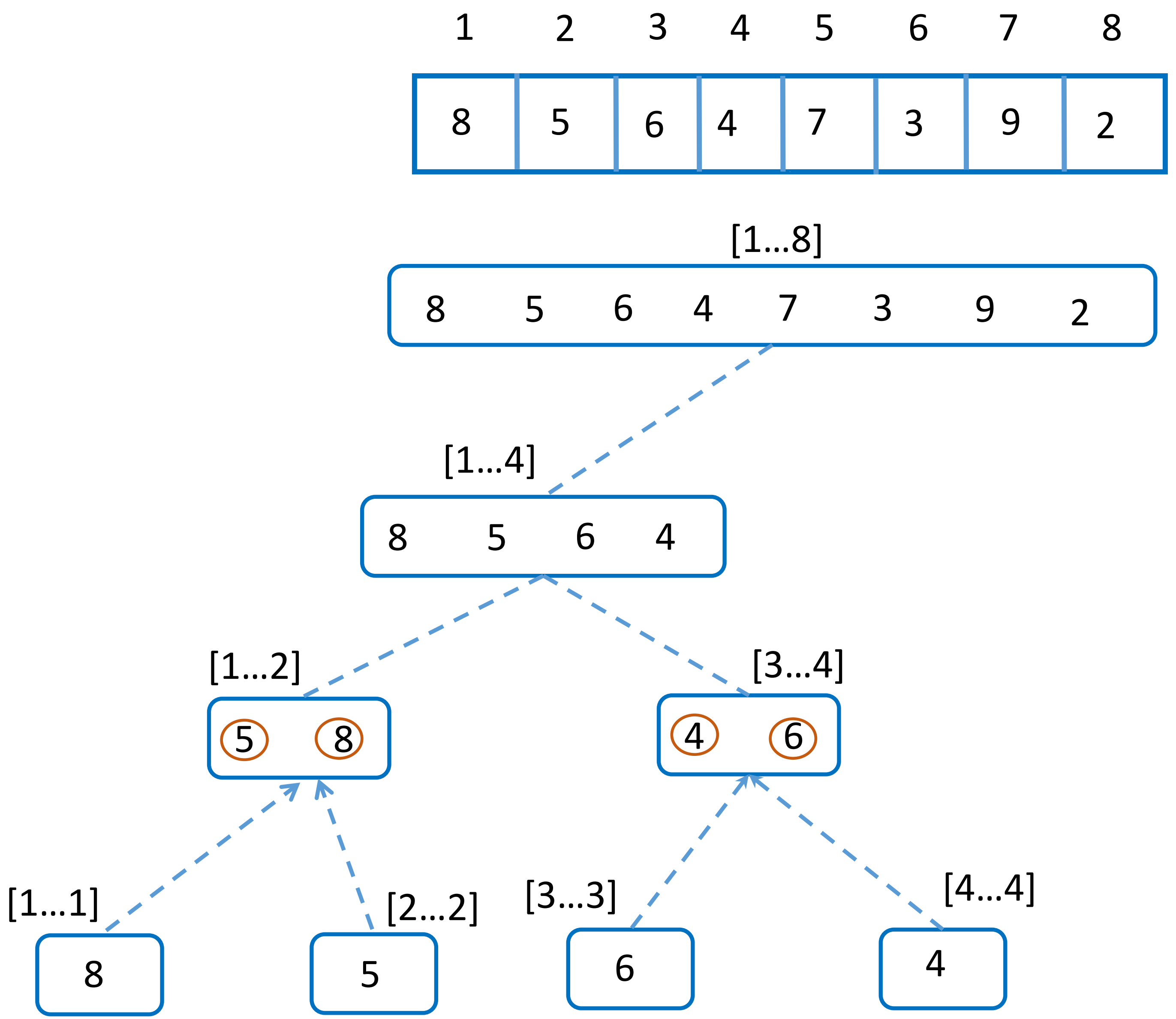
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



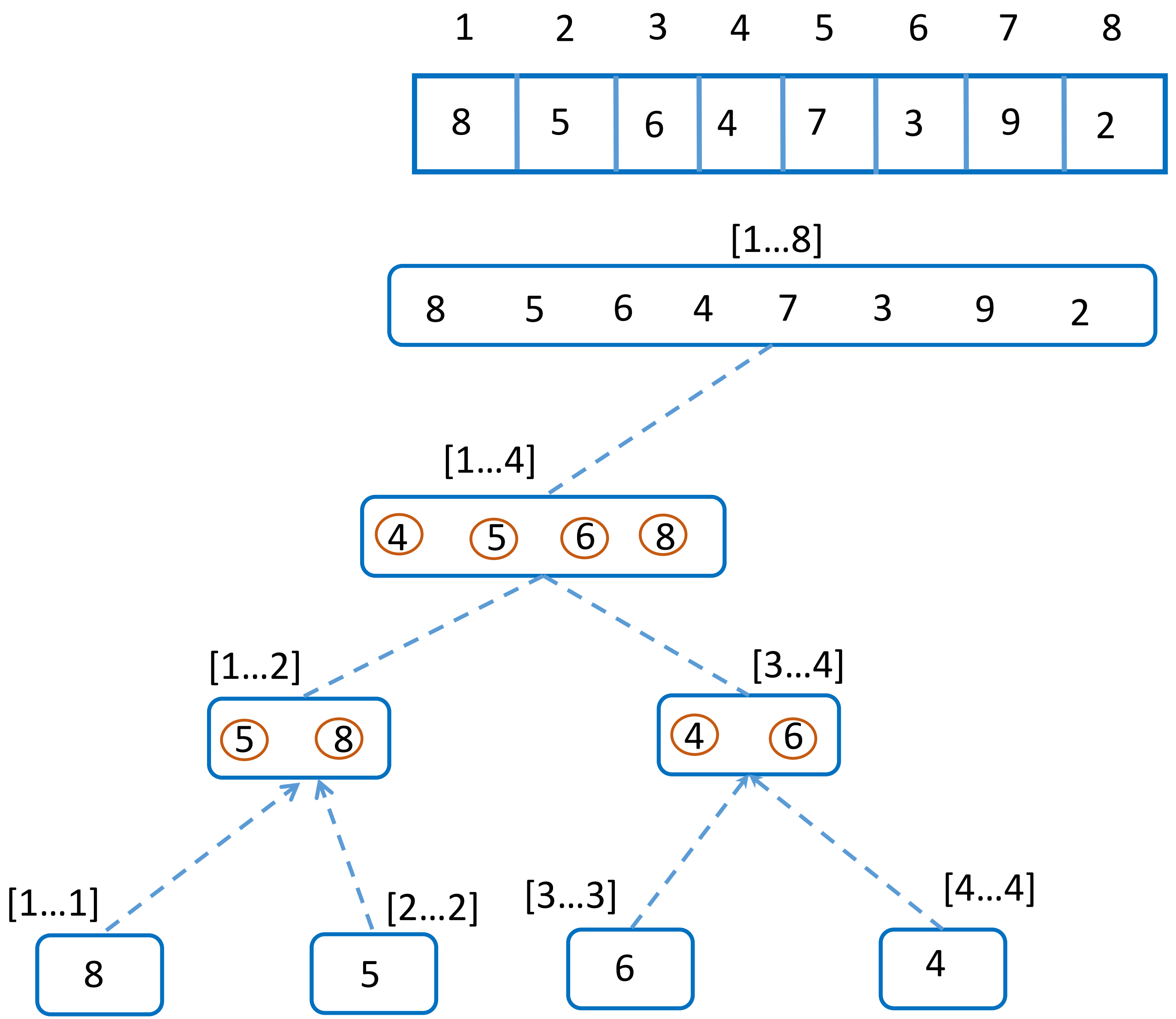
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



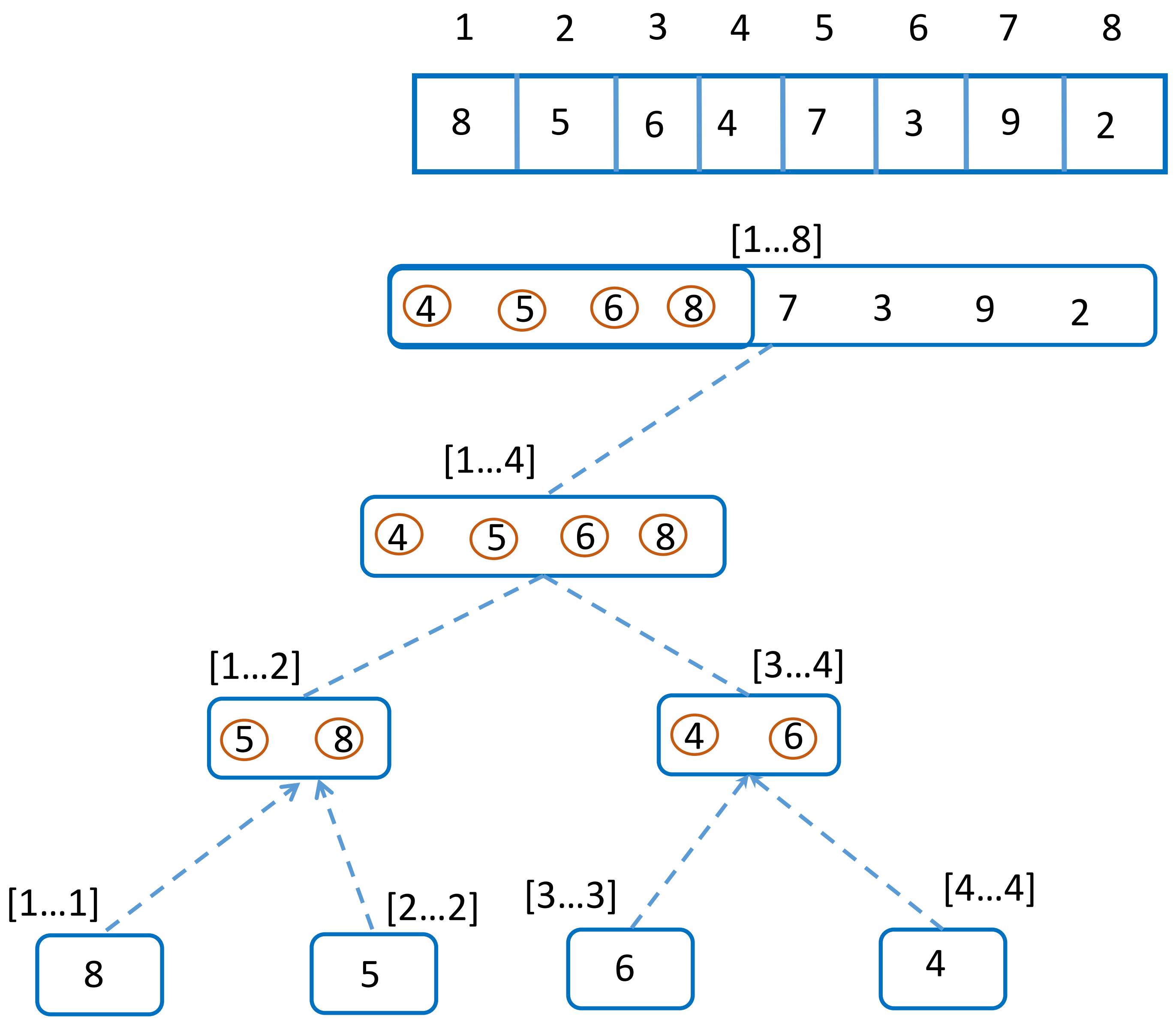
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



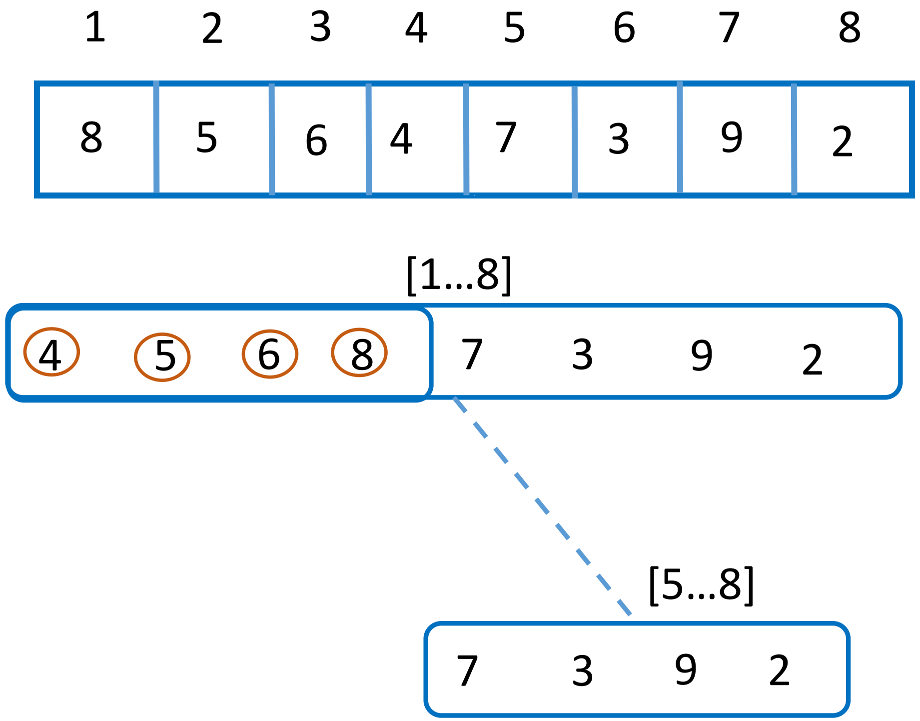
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



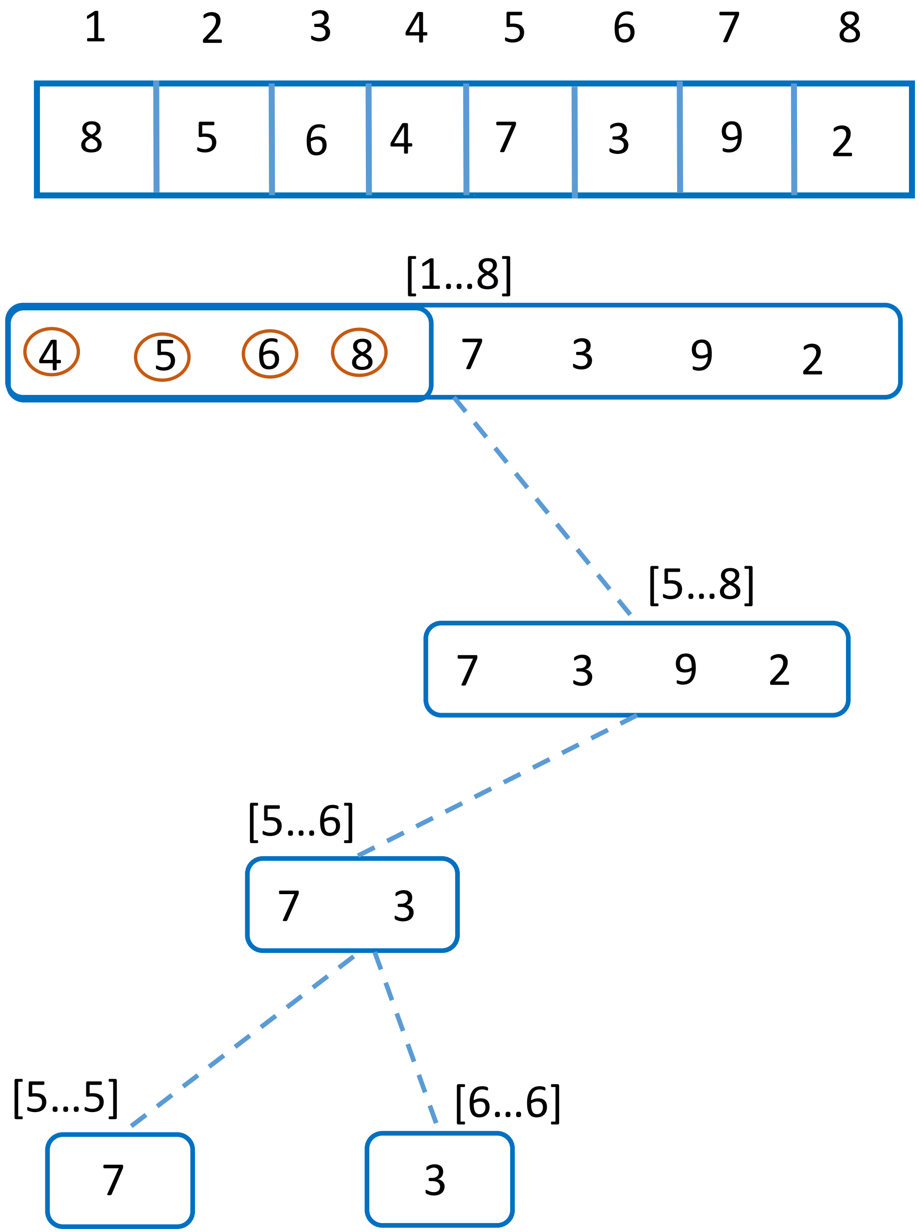
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



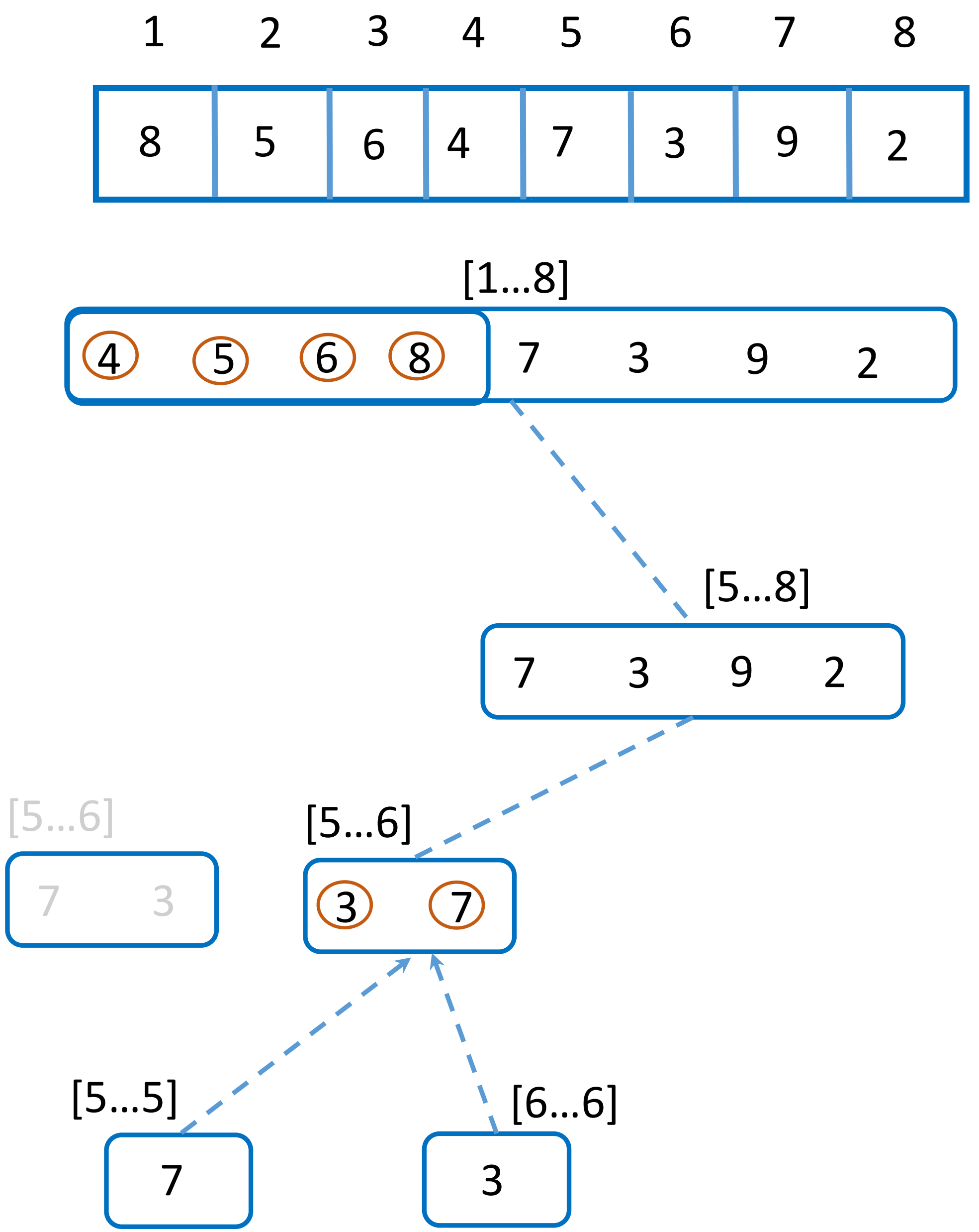
Example: Merge Sort

```
void mergesort(A, l, h)
{
   if (l < r)
     {
     mid = (l + h) / 2;
     mergesort(A, l, mid);
     mergesort(A, mid + 1, h);
     merge(A, l, mid, h);
     }
}
```



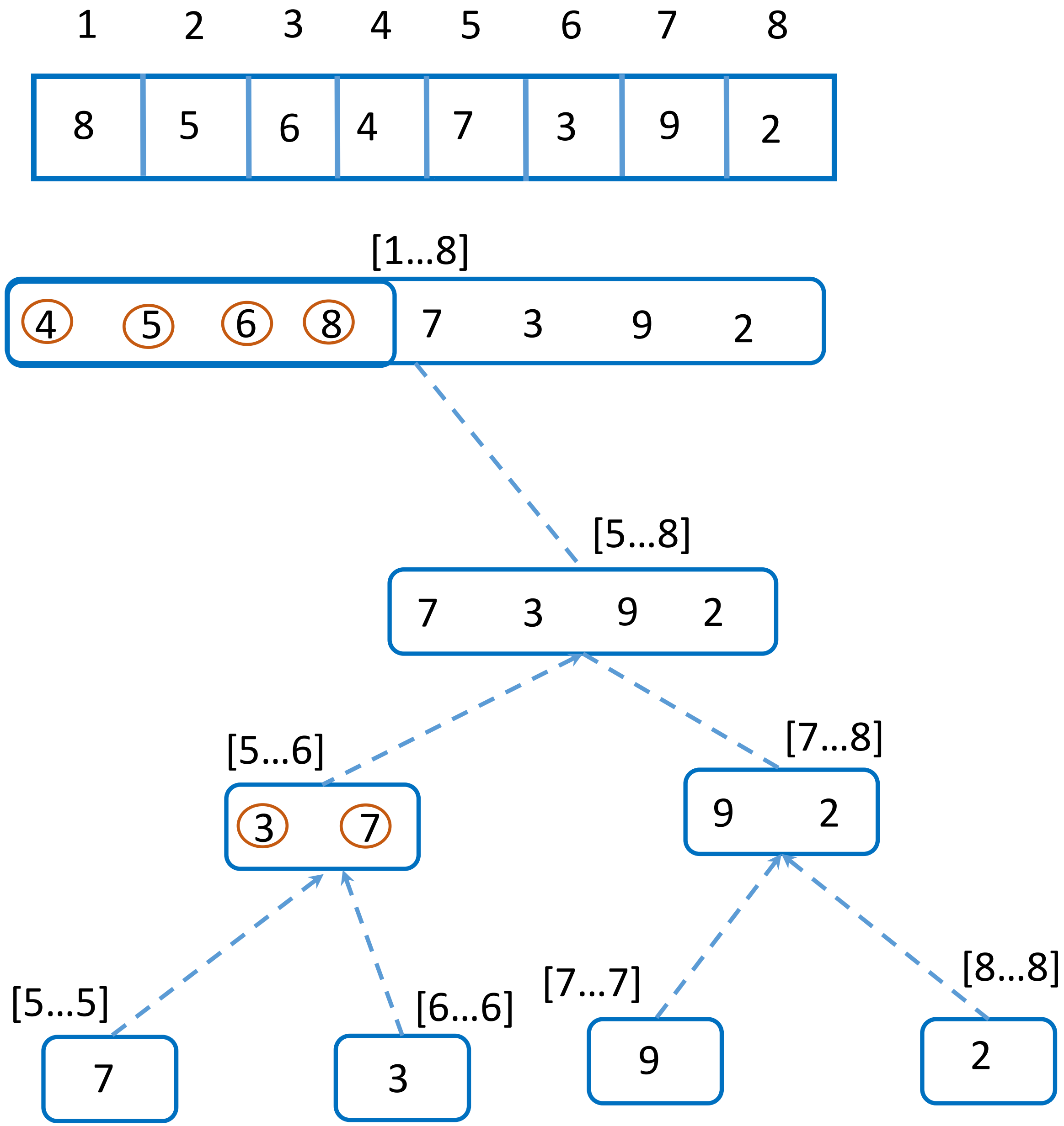
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



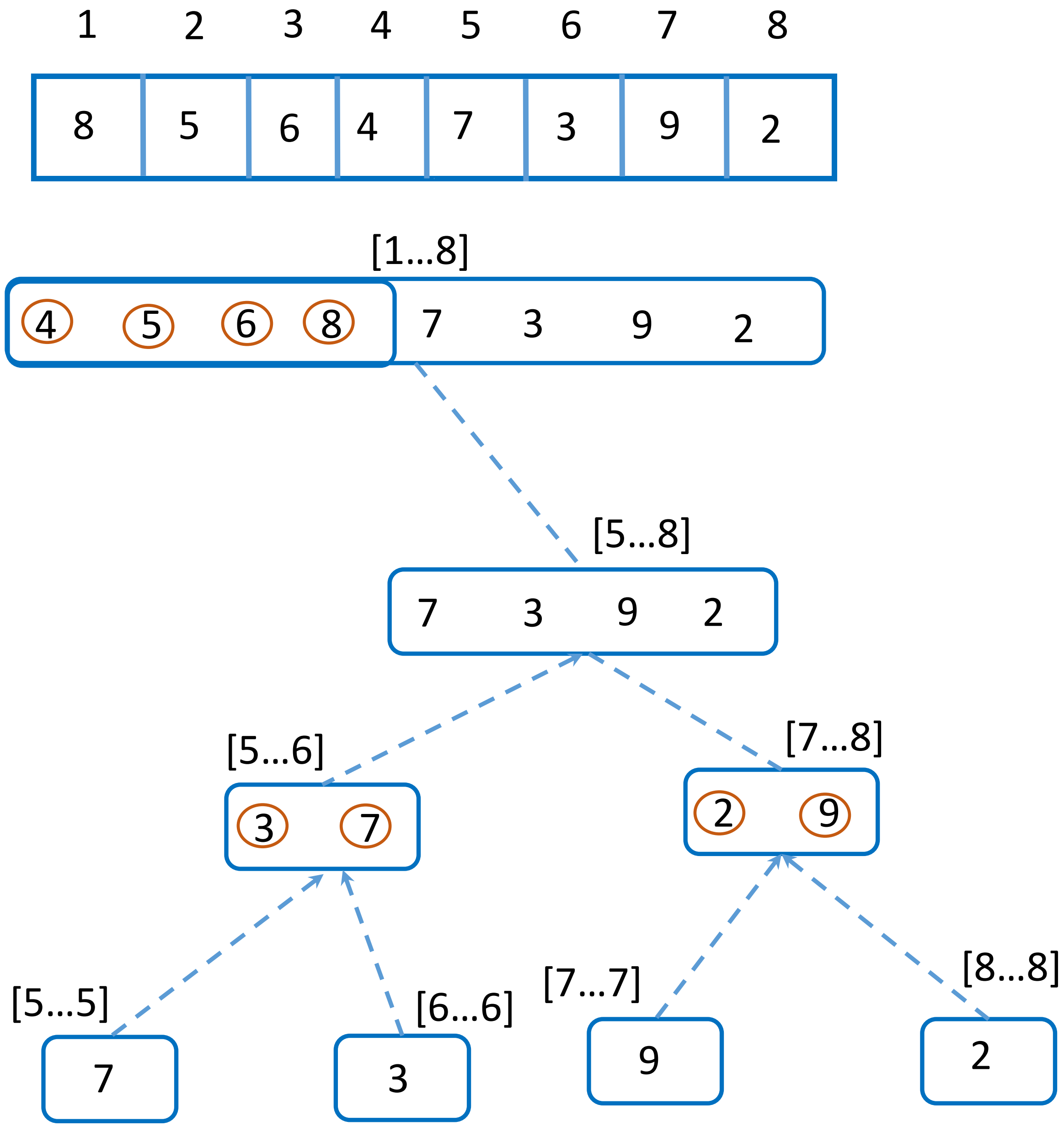
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



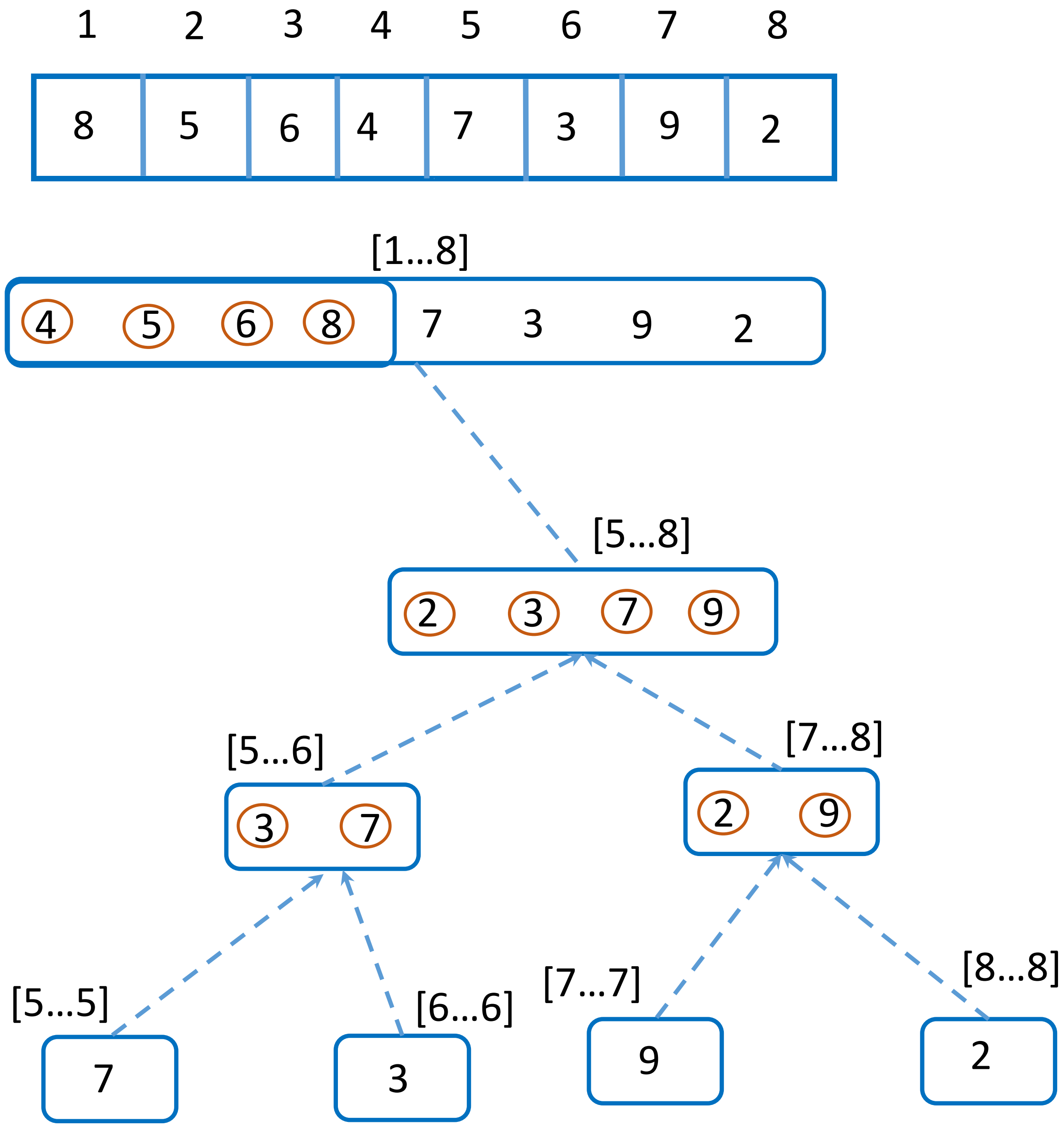
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



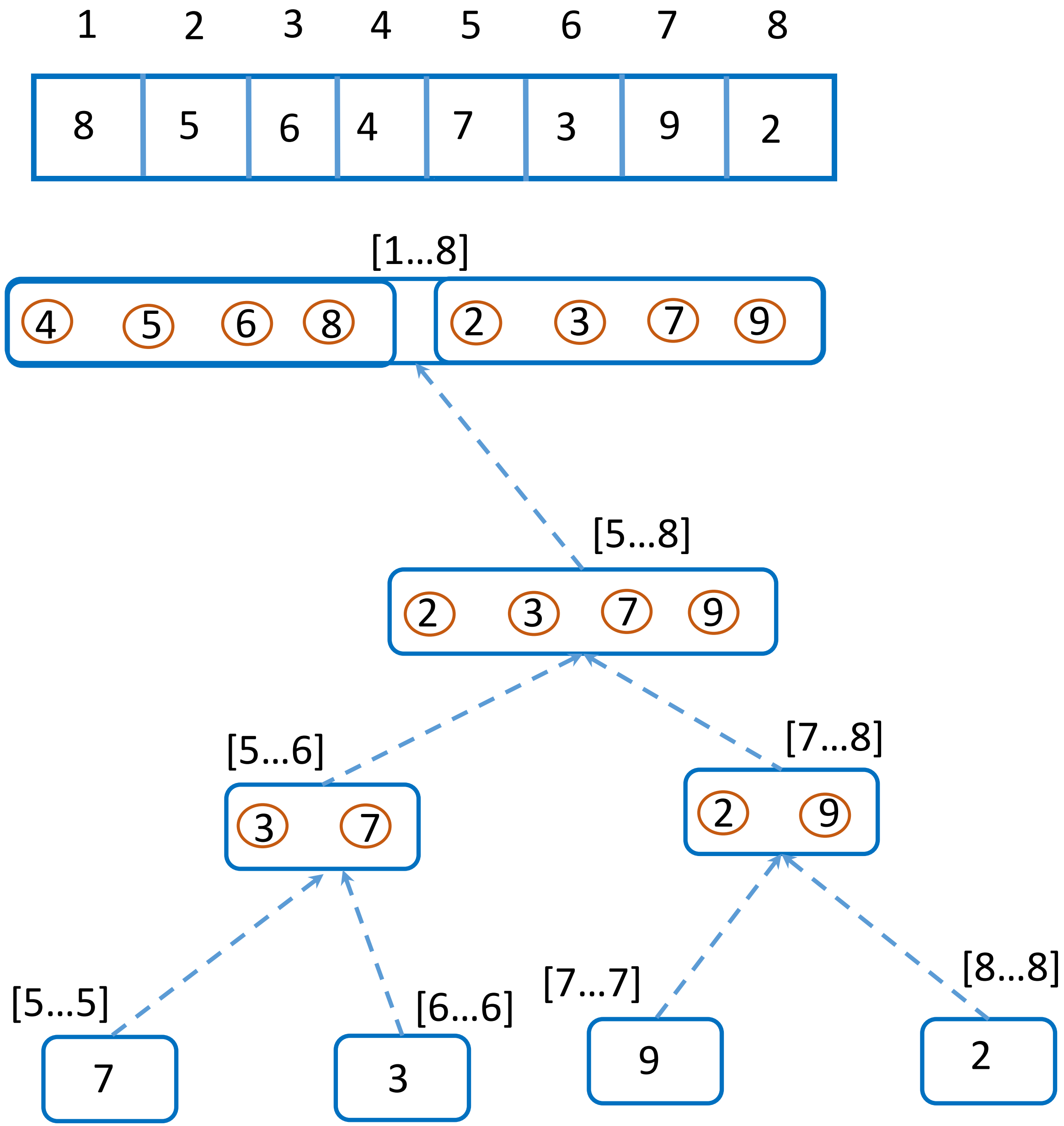
Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



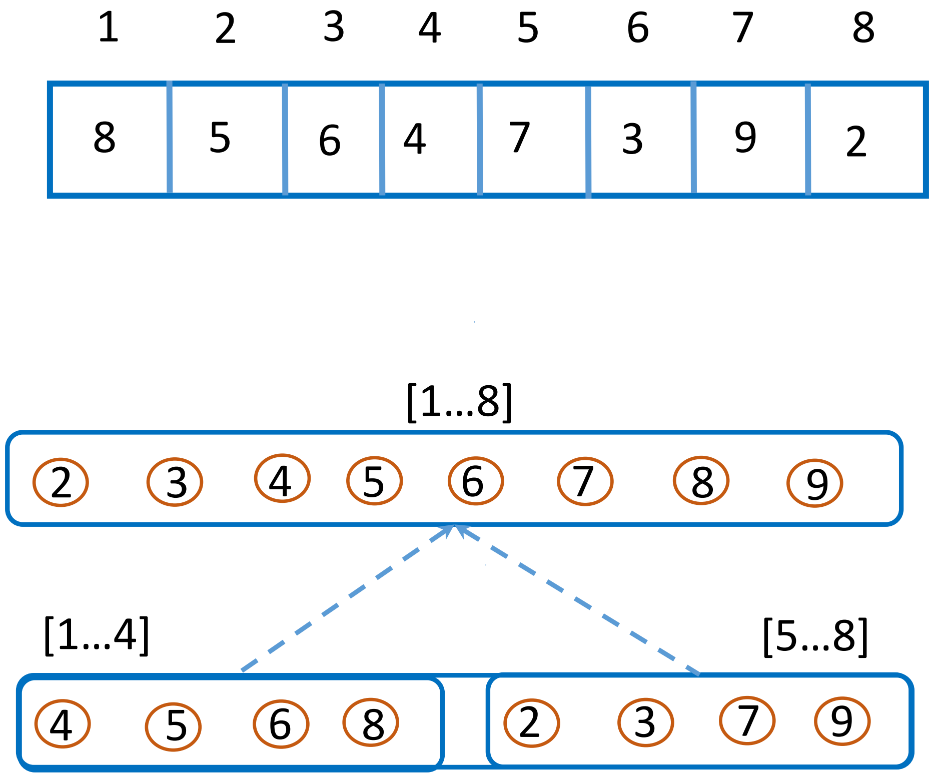
Example: Merge Sort

```
void mergesort(A, l, h)
{
   if (l < r)
     {
     mid = (l + h) / 2;
     mergesort(A, l, mid);
     mergesort(A, mid + 1, h);
     merge(A, l, mid, h);
     }
}
```



Example: Merge Sort

```
void mergesort(A, l, h)
{
  if (l < r)
    {
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
    }
}
```



[8…8]

Example: Merge Sort

```
void merge(A, l, mid, h){

  i = j = 0;
  k=l;
  for k = l to h
  {
        if (L[i] <= R[j])
        {
            A[k] = L[i];
            i += 1;
        }
        else {
            A[k] = R[j];
            j += 1;
        }

    }

 }
```

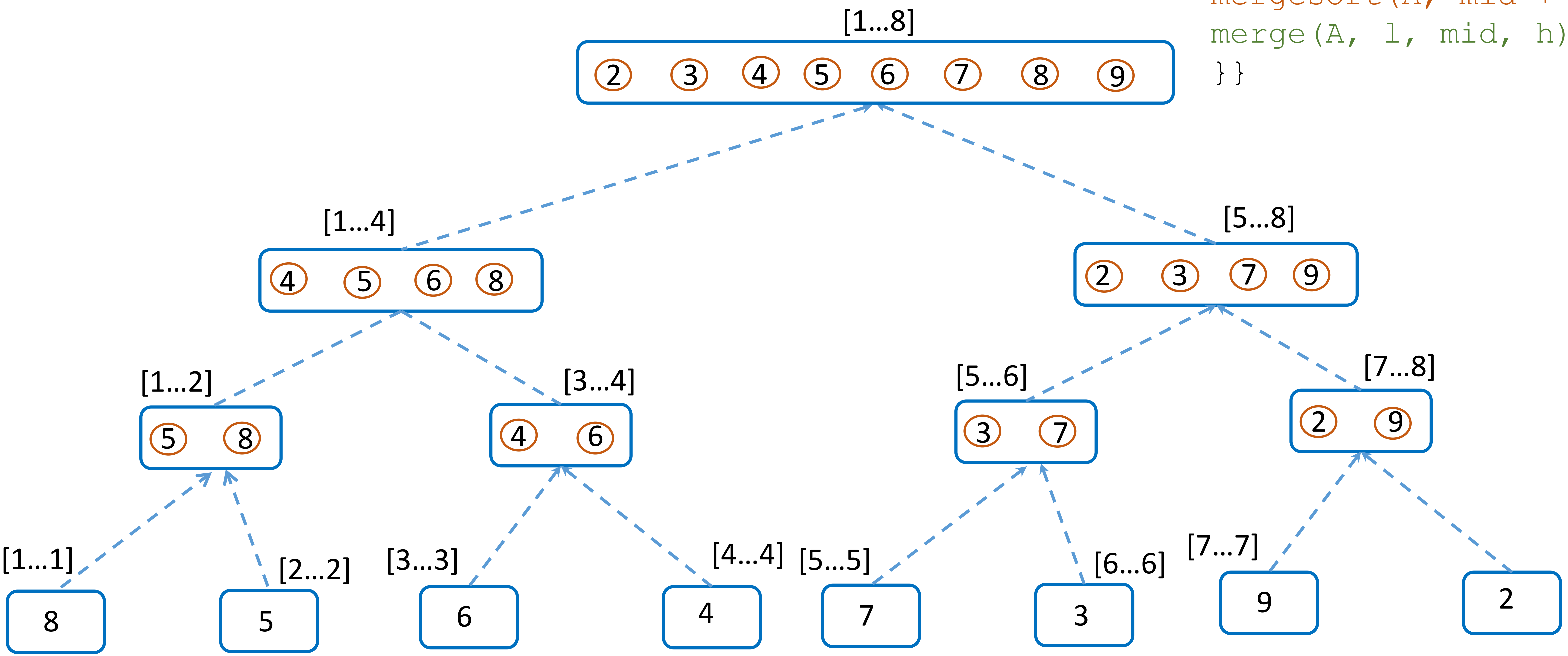
1	2	3	4	5	6	7	8
8	5	6	4	7	3	9	2

L	4	5	6	8
R	2	3	7	9

	1	2	3	4	5	6	7	8
A								

Analysis: Merge Sort

```
void mergesort(A, l, h){
  if (l < r){
    mid = (l + h) / 2;
    mergesort(A, l, mid);
    mergesort(A, mid + 1, h);
    merge(A, l, mid, h);
}}
```



Radix Sort Algorithm

- <u>Define:</u> A storing technique → implements <u>digit by digit</u> sort starting from Least Significant Digit (LSD) to Most Significant Digit (MSD)
- <u>Example:</u>

  981 → $d_3d_2d_1$

  ← direction

  Least Significate Digit (LSD) = 1; Most Significant Digit = 9

  $\underline{1}2\underline{3}$ → LSD=3; MSD=1

  - <u>Digit-by-Digit</u>:

$d_3d_2d_1$						
$88\underline{1}$	d_1 →	$8\underline{8}1$	d_2 →	$\underline{2}23$	d_3 →	142
$22\underline{3}$		$1\underline{4}2$		$\underline{1}42$		223
$14\underline{2}$		$2\underline{2}3$		$\underline{8}81$		881

Exercise: Arrays

	0	1	2	3	4	5	6	7
A	1	1	1	4	2	4	5	2

```
n=8; //length of the given array
```

```
int dc[10]; //digit count array
```

1
```
for (int i=0; i<10; i++)
       dc[i] = 0;
```

2
```
for (i=0; i<n; i++)
      dc[A[i]]++;
```

3
```
for (i=0; i<10; i++)
 printf("%d", dc[i]);
```

What is the output of Step 3, after executing Step 1 and Step 2?

Radix Sort → Algorithm

```
void radixsort( int A[],int d,int size)
{
  for(int i=1;i<=d; i++)
       countingsort(A, i, size);
}
```

$d_3 d_2 d_1$

```
// A[ ]→ the array A contains the input numbers
// d → number of digits from the largest number in the given input
// size → the size of the given input
```

Counting Sort Algorithm

Step 1: Get digits of the given number and store into a temporary array

Example: A = {123, 345, 542, 643, 111}

when d_1 is considered (i.e., d == 1):

```
A = {123, 345, 542, 643, 111}
ds = {3, 5, 2, 3, 1}
```

```
//ds: digit separation
ds [0] = A[0]%10
ds [1] = A[1]%10
……
ds [size] = A[size]%10
```

Step 2: Count the distinct elements from the digit separation array

```
for (int i=0; i<10; i++)
      dc[i] = 0;

for (i = 0; i<n; i++)
     dc[ds[i]]++;
```

0	1	2	3	4	5	6	7	8	9
0	1	1	2	0	1	0	0	0	0

Counting Sort Algorithm

Step 3: Apply the following operation!

```
for (int i=1; i<10; i++)
        dc[i] = dc[i] + dc[i-1];
```

dc original

0	1	2	3	4	5	6	7	8	9
0	1	1	2	0	1	0	0	0	0

dc updated

0	1	2	3	4	5	6	7	8	9
0	1	2	4	4	5	5	5	5	5

Step 4: Count the distinct elements from the digit separation array

```
for (int i = size-1; i>=0; i--)
   1   ax[dc[ds[i]] =A[i];
   2  //decrement the count value at
the dc array
```

Counting Sort Algorithm (1)

```
      0    1    2    3    4
ds = {3,  5,   2,  3,   1}

      0    1    2    3    4    5    6    7    8    9
dc = {0,  1,   2,  4,   4,   5,  5,   5,   5,   5}
```

Step 4: Apply the following operation!

```
for (int i = size-1; i>=0; i--)
1    ax[dc[ds[i]]-1] = A[i];
2   //decrement the count value at
the dc array
```

```
      0     1     2     3     4
A = {123, 345, 542, 643, 111}
```

When size is 5

i = 4
ds[4] → 1
1 **dc[1] → 1**
ax[1-1] ← A[4]
ax[0] = 111

ax = {111, -, -, -, -}

2 dc = {0, 0, 2, 4, 4, 5, 5, 5, 5, 5}

dc[ds[i]]--;

Counting Sort Algorithm (2)

```
        0  1   2  3   4
ds = {3, 5,  2, 3,  1}

        0  1   2  3   4   5  6   7   8   9
dc = {0, 0,  2, 4,  4,  5, 5,  5,  5,  5}
```

Step 4: Apply the following operation!

```
for (int i = size-1; i>=0; i--)
  1   ax[dc[ds[i]]-1] = A[i];
  2  //decrement the count value at
the dc array
```

```
        0     1     2     3     4
A = {123, 345, 542, 643, 111}
```

When size is 5

```
    i = 3
    ds[3] → 3
1   dc[3] → 4
    ax[4-1] ← A[3]
    ax[3] = 643
```

ax = {111, -, -, 643, -}

```
2   dc = {0, 0,  2, 3,  4,  5, 5,  5,  5,  5}
```

dc[ds[i]]--;

Counting Sort Algorithm (3)

	0	1	2	3	4
ds =	{3,	5,	2,	3,	1}

	0	1	2	3	4	5	6	7	8	9
dc =	{0,	0,	2,	3,	4,	5,	5,	5,	5,	5}

Step 4: Apply the following operation!

```
for (int i = size-1; i>=0; i--)
 1    ax[dc[ds[i]]-1] = A[i];
 2   //decrement the count value at the dc array
```

	0	1	2	3	4
A =	{123,	345,	542,	643,	111}

When size is 5

i = 2

ds[2] → 2

1 **dc[2] → 2**

ax[2-1] ← A[2]

ax[1] = 542

ax = {111, 542, -, 643, -}

2 dc = {0, 0, 1, 3, 4, 5, 5, 5, 5, 5}

dc[ds[i]]--;

Counting Sort Algorithm (4)

```
      0   1   2   3   4
ds = {3,  5,  2,  3,  1}

      0   1   2   3   4   5   6   7   8   9
dc = {0,  0,  2,  3,  4,  5,  5,  5,  5,  5}
```

Step 4: Apply the following operation!

```
for (int i = size-1; i>=0; i--)
  1    ax[dc[ds[i]]-1] = A[i];
  2   //decrement the count value at
the dc array
```

```
      0     1     2     3     4
A = {123, 345, 542, 643, 111}
```

When size is 5

```
    i = 1
    ds[1] → 5
1   dc[5] → 5
    ax[5-1] ← A[1]
    ax[4] = 345
```

```
ax = {111, 542, -, 643, 345}
```

```
2   dc = {0, 0, 1, 3, 4, 4, 5, 5, 5, 5}
```

```
dc[ds[i]]--;
```

Counting Sort Algorithm (5)

	0	1	2	3	4
ds =	{3,	5,	2,	3,	1}

	0	1	2	3	4	5	6	7	8	9
dc =	{0,	0,	2,	3,	4,	5,	5,	5,	5,	5}

Step 4: Apply the following operation!

```
for (int i = size-1; i>=0; i--)
 1    ax[dc[ds[i]]-1] = A[i];
 2   //decrement the count value at
the dc array
```

	0	1	2	3	4
A =	{123,	345,	542,	643,	111}

When size is 5

i = 0
1 **ds[0] → 3**
dc[3] → 3
ax[3-1] ← A[0]
ax[2] = 123

ax = {111, 542, 123, 643, 345}

2 dc = {0, 0, 1, 2, 4, 4, 5, 5, 5, 5}

dc[ds[i]]--;

Counting Sort Algorithm (6)

Step 5: Assig the ax array values to original array

```
for (int i=0; i<size-1; i++)
        A[i] = ax[i];
```

Original

A = {123, 345, 542, 643, 111}

ax = {111, 542, 123, 643, 345}

After Step 5 → A = {111, 542, 123, 643, 345}

Step 6: Repeat the Steps 1 to Step 5 until you visit all the digits (i.e., MSD).

Counting Sort Algorithm (7)

Step 7: Repeat the Steps 1 to Step 5 until you visit all the digits (i.e., MSD).

$A = \{12\underline{3},\ 34\underline{5},\ 54\underline{2},\ 64\underline{3},\ 11\underline{1}\}$

After d_1 $A = \{111,\ 542,\ 123,\ 643,\ 345\}$

$12\underline{3}$		$1\underline{1}1$		$\underline{1}11$		111
$34\underline{5}$	$\xrightarrow{d_1}$	$5\underline{4}2$	$\xrightarrow{d_2}$	$\underline{1}23$	$\xrightarrow{d_3}$	123
$54\underline{2}$		$1\underline{2}3$		$\underline{5}42$		345
$64\underline{3}$		$6\underline{4}3$		$\underline{6}43$		542
$11\underline{1}$		$3\underline{4}5$		$\underline{3}45$		643

Please kindly check the sorting of the d_2

thank you!

email:
k.kondepu@iitdh.ac.in