CSE 1201
Merge Sort

Md. Shahid Uz Zaman
Dept of CSE, RUET

MergeSort

- MergeSort is a *divide and conquer* method of sorting

MergeSort Algorithm

- To sort an array of **n** elements, we perform the following steps in sequence:
- If **n < 2** then the array is already sorted.
- Otherwise, **n > 1**, and we perform the following three steps in sequence:
  1. **Sort** the **<u>left</u> <u>half</u>** of the the array using MergeSort.
  2. **Sort** the **<u>right</u> <u>half</u>** of the the array using MergeSort.
  3. **<u>Merge</u>** the sorted left and right halves.

How to Merge

Here are two lists to be merged:

First:	(12, 16, 17, 20, 21, 27)
Second:	(9, 10, 11, 12, 19)

Compare12 and 9

First:	(12, 16, 17, 20, 21, 27)
Second:	(10, 11, 12, 19)
New:	(9)

Compare 12 and 10

First:	(12, 16, 17, 20, 21, 27)
Second:	(11, 12, 19)
New:	(9, 10)

Merge Example

Compare 12 and 11

First:	(12, 16, 17, 20, 21, 27)
Second:	(12, 19)
New:	(9, 10, 11)

Compare 12 and 12

First:	(16, 17, 20, 21, 27)
Second:	(12, 19)
New:	(9, 10, 11, 12)

Merge Example

Compare **16** and **12**

First:	(16, 17, 20, 21, 27)
Second:	(19)
New:	(9, 10, 11, 12, 12)

Compare **16** and **19**

First:	(17, 20, 21, 27)
Second:	(19)
New:	(9, 10, 11, 12, 12, 16)

Merge Example

Compare **17** and **19**

First:	(20, 21, 27)
Second:	(19)
New:	(9, 10, 11, 12, 12, 16, 17)

Compare **20** and **19**

First:	(20, 21, 27)
Second:	( )
New:	(9, 10, 11, 12, 12, 16, 17, 19)

Merge Example

Checkout **20** and **empty list**

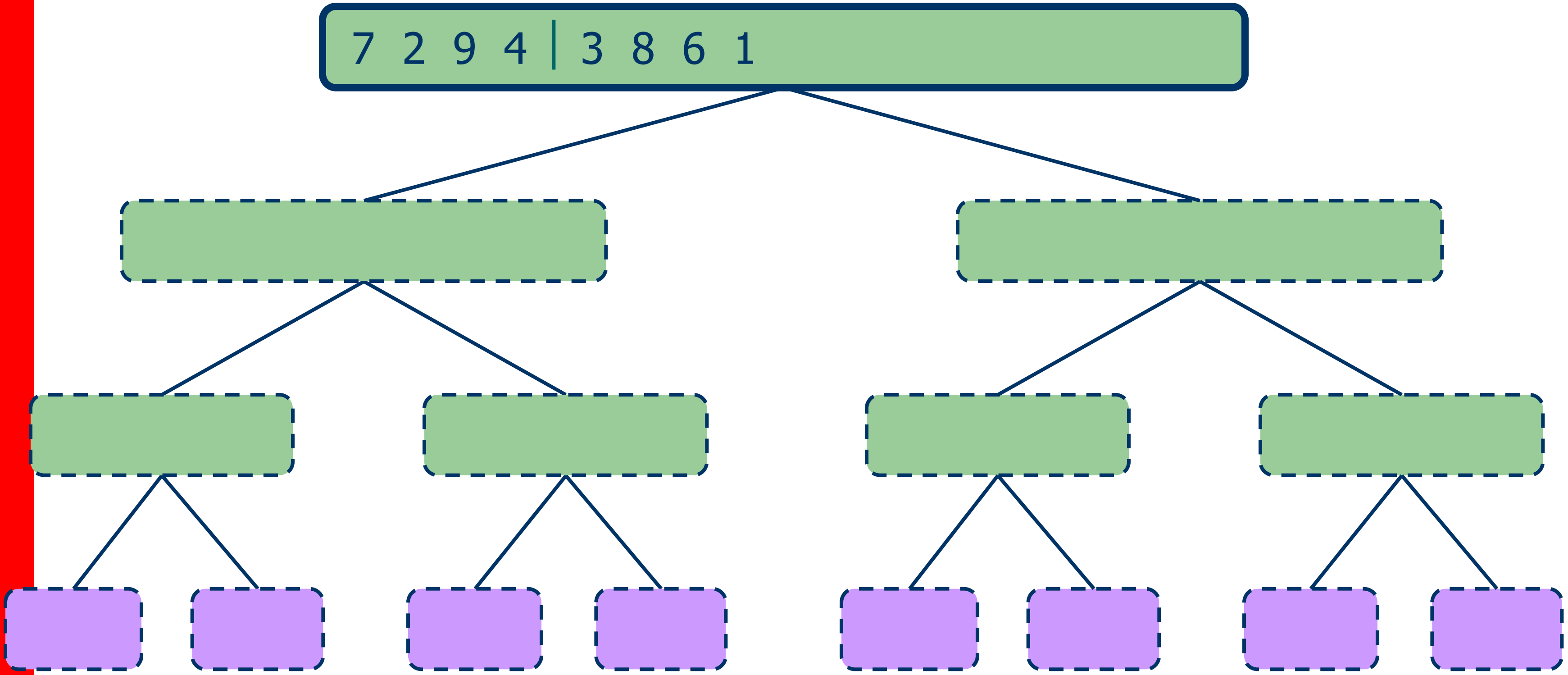
- **First: ( )**
- **Second: ( )**
- **New: (9, 10, 11, 12, 12, 16, 17, 19,** 20, 21, 27**)**

MergeSort

Original	24	13	26	1	12	27	38	15							
Divide in 2	24	13	26	1		12	27	38	15						
Divide in 4	24	13		26	1		12	27		38	15				
Divide in 8	24		13		26		1		12		27		38		15
Merge 2	13	24			1	26			12	27			15	38	
Merge 4	1	13	24	26					12	15	27	38			
Merge 8	1	12	13	15	24	26	27	38							

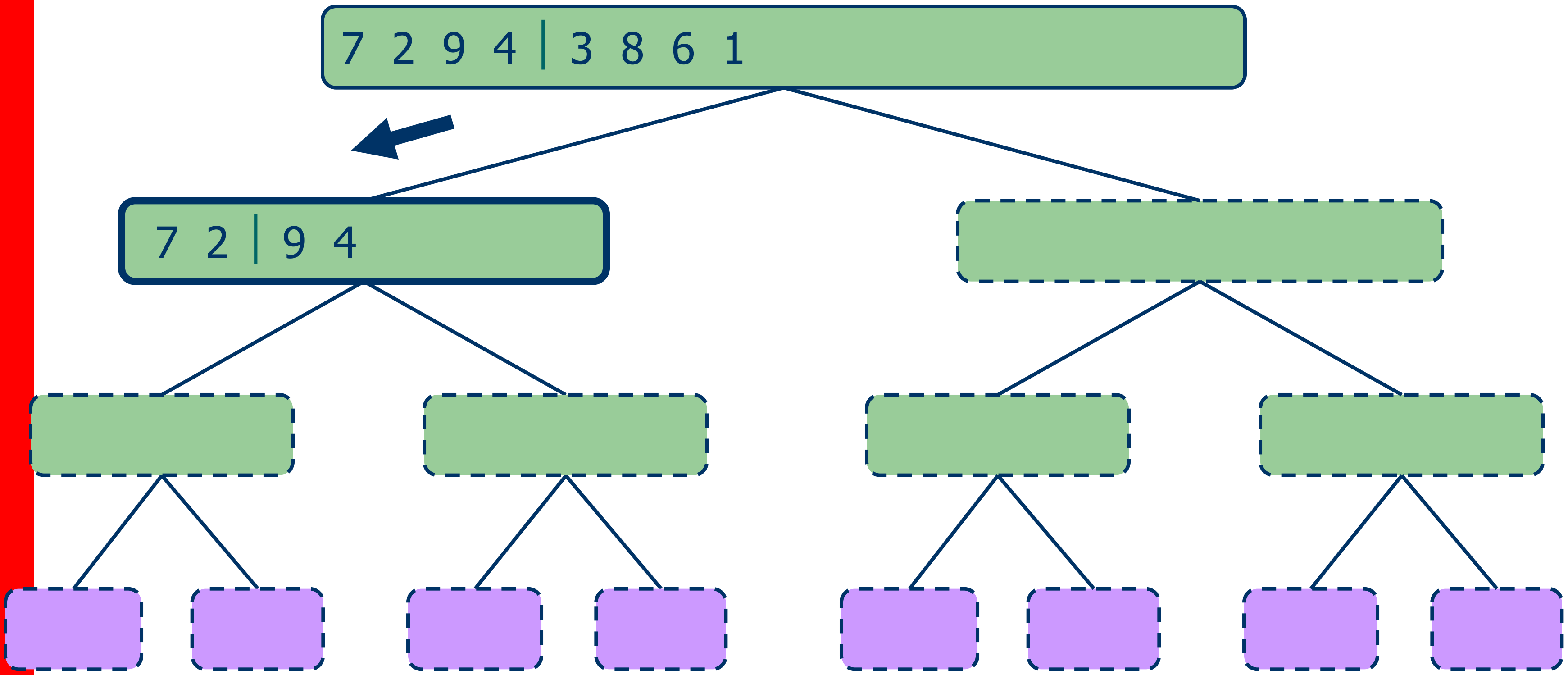
Execution Example

- Partition



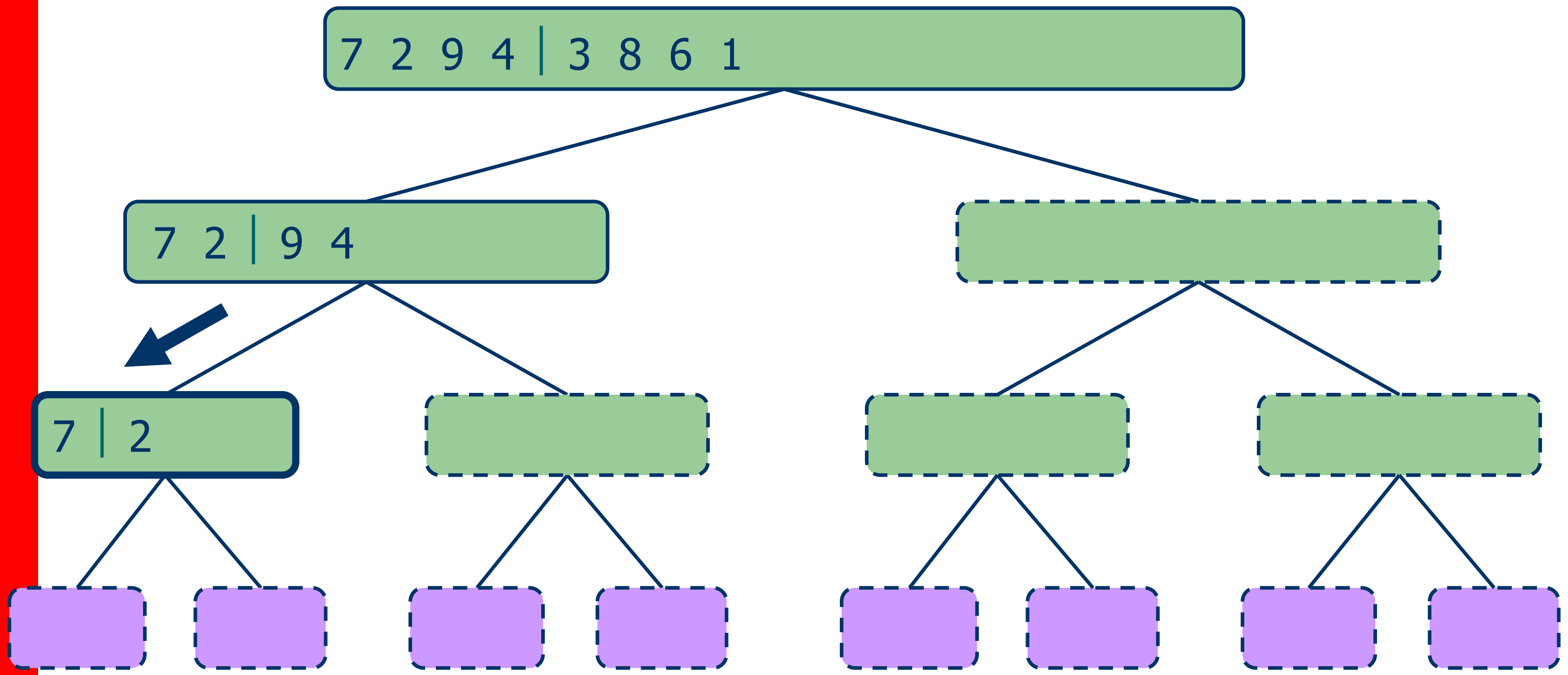
Execution Example (cont.)

- Recursive call, partition



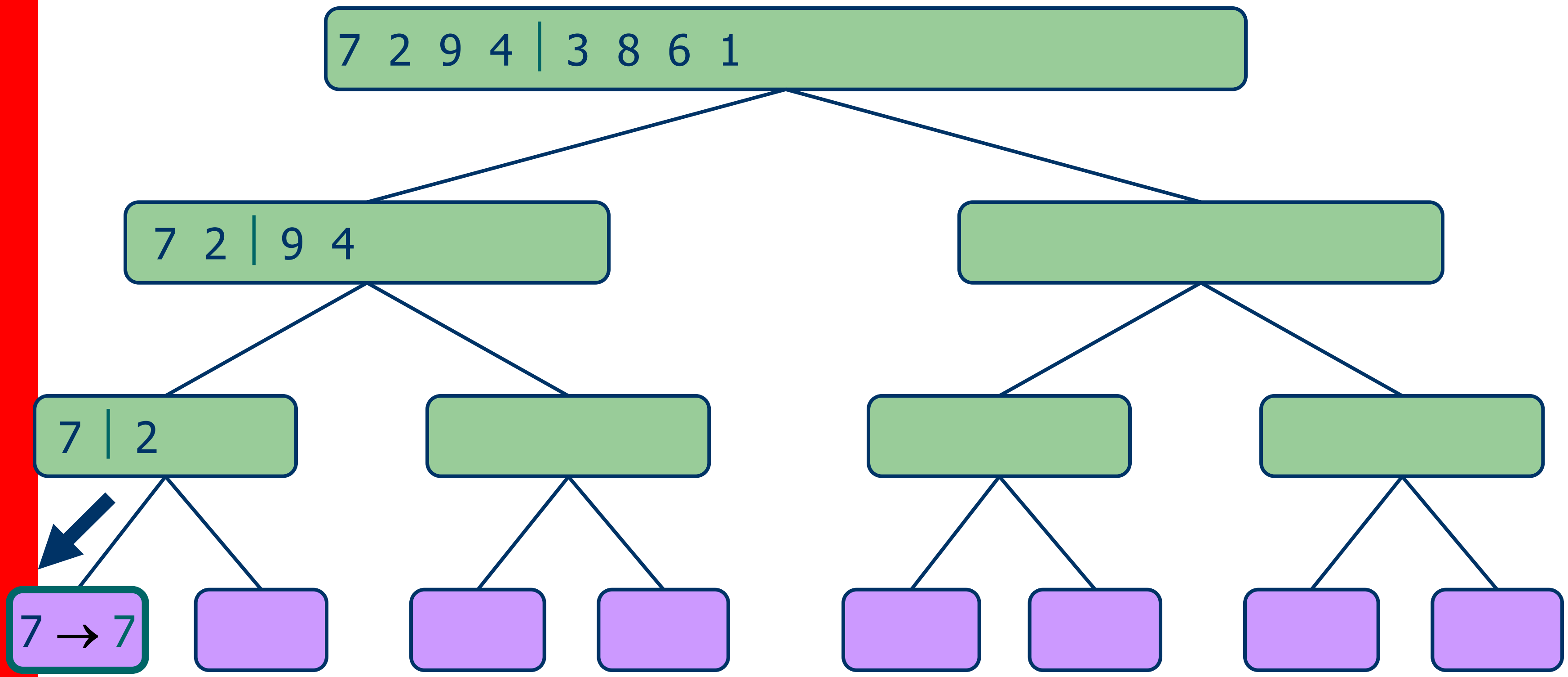
Execution Example (cont.)

- Recursive call, partition



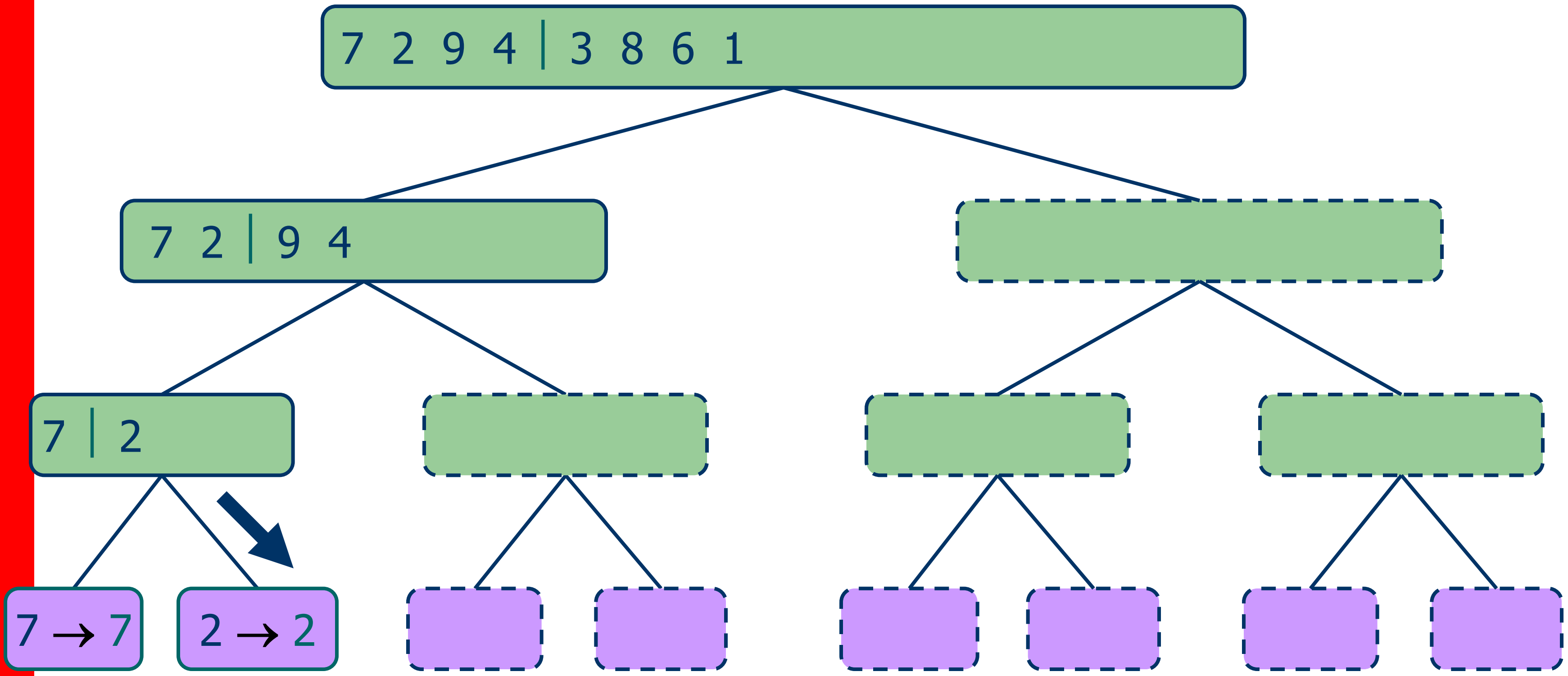
Execution Example (cont.)

- Recursive call, base case



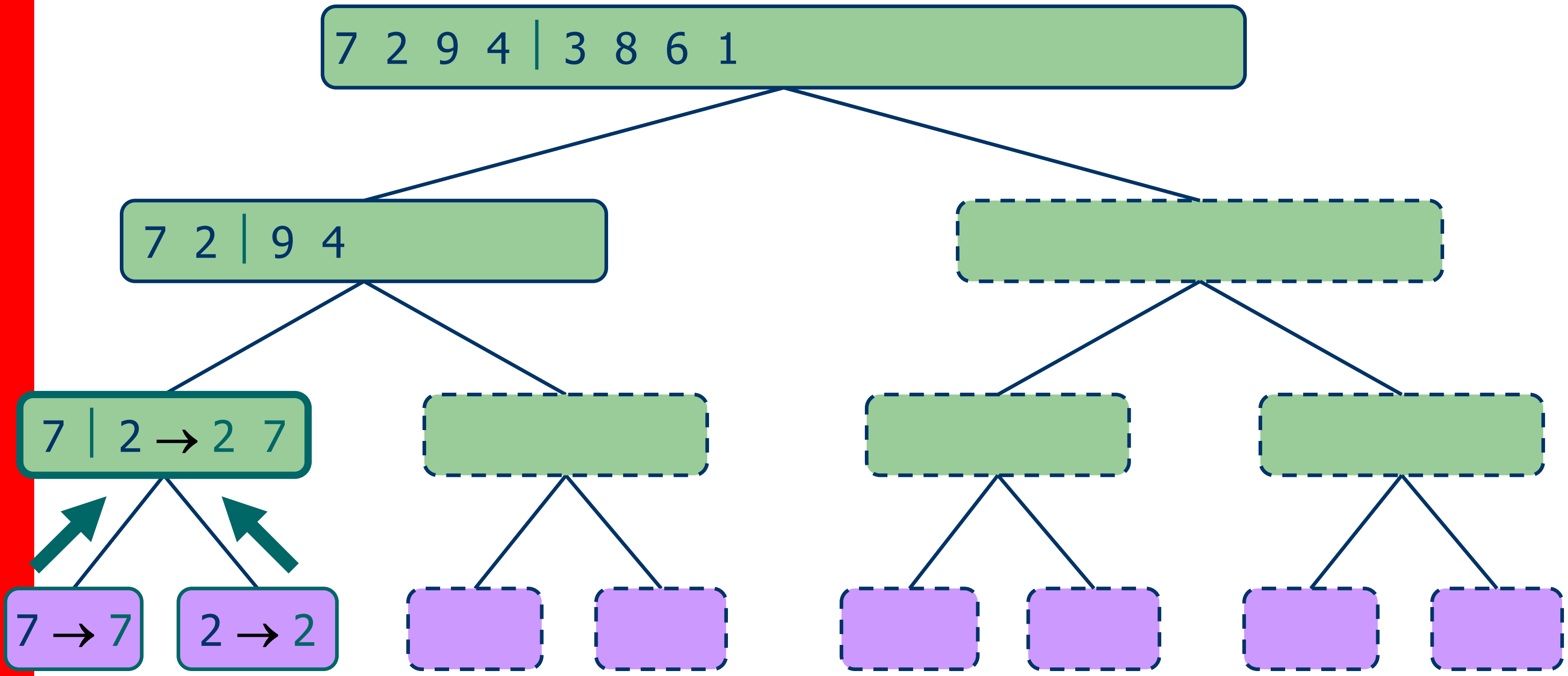
Execution Example (cont.)

- Recursive call, base case



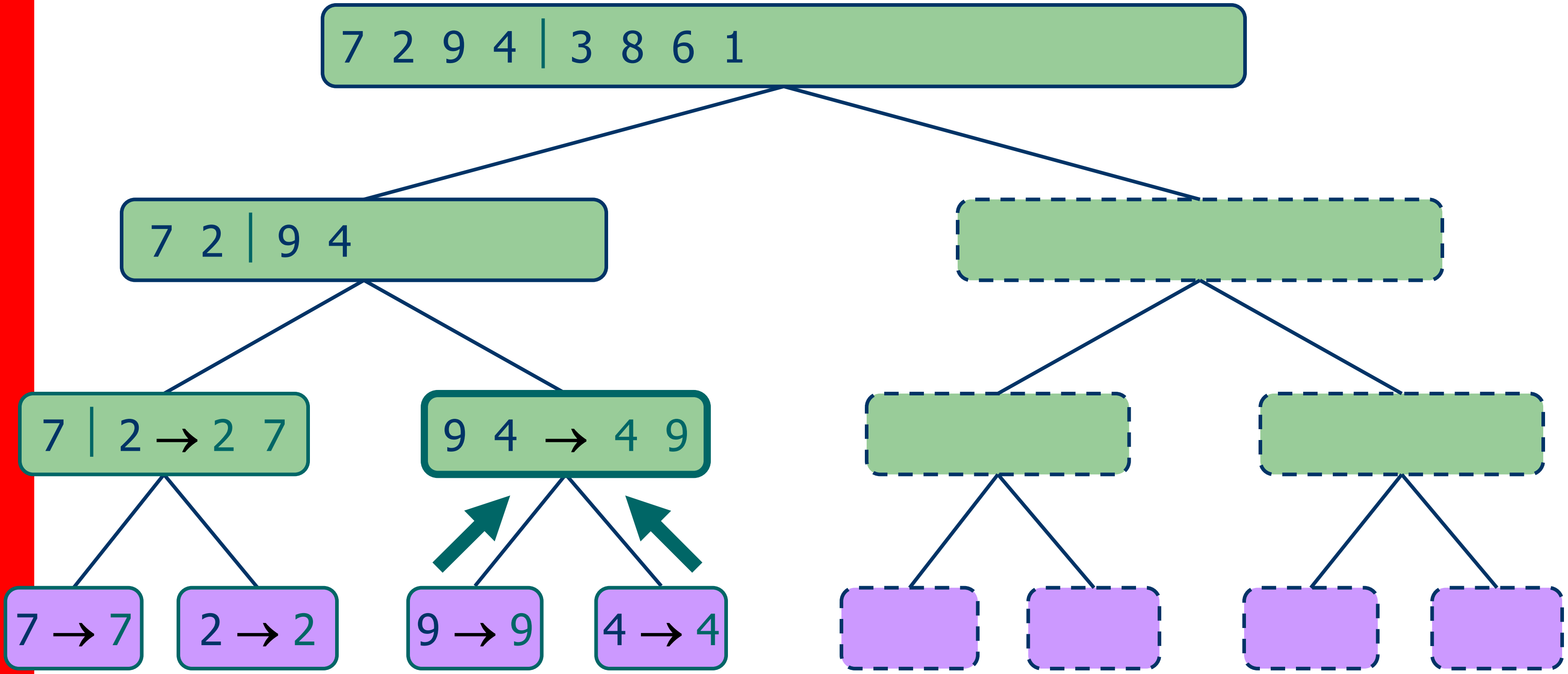
Execution Example (cont.)

- Merge



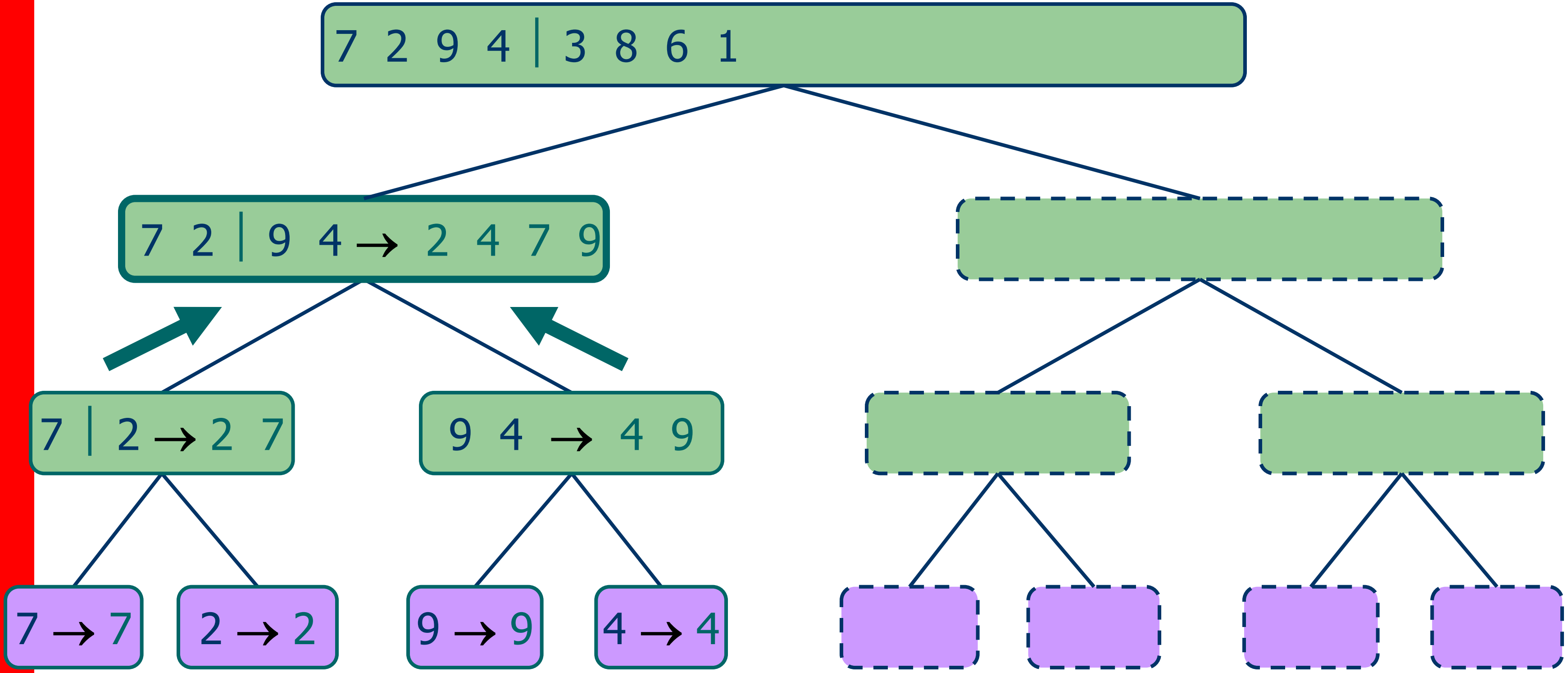
Execution Example (cont.)

- Recursive call, …, base case, merge



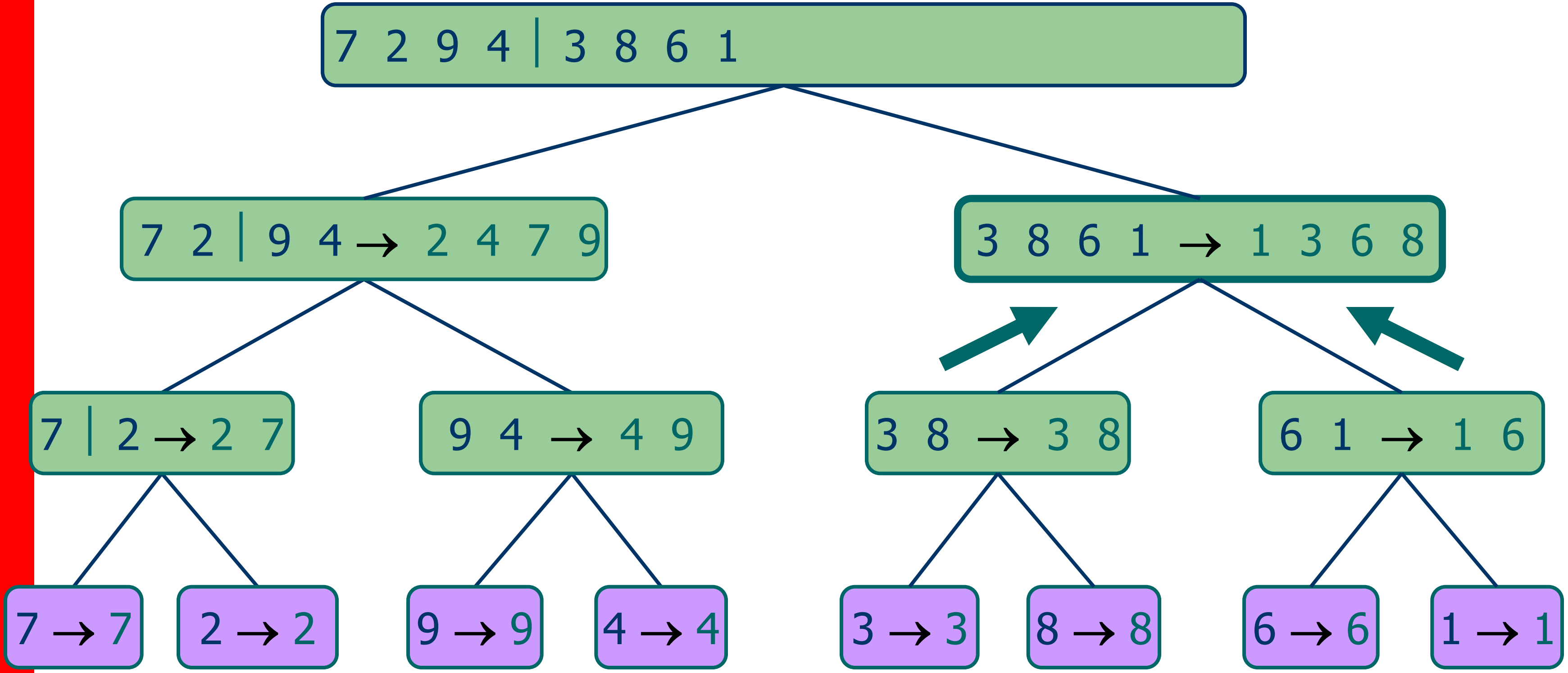
Execution Example (cont.)

- Merge



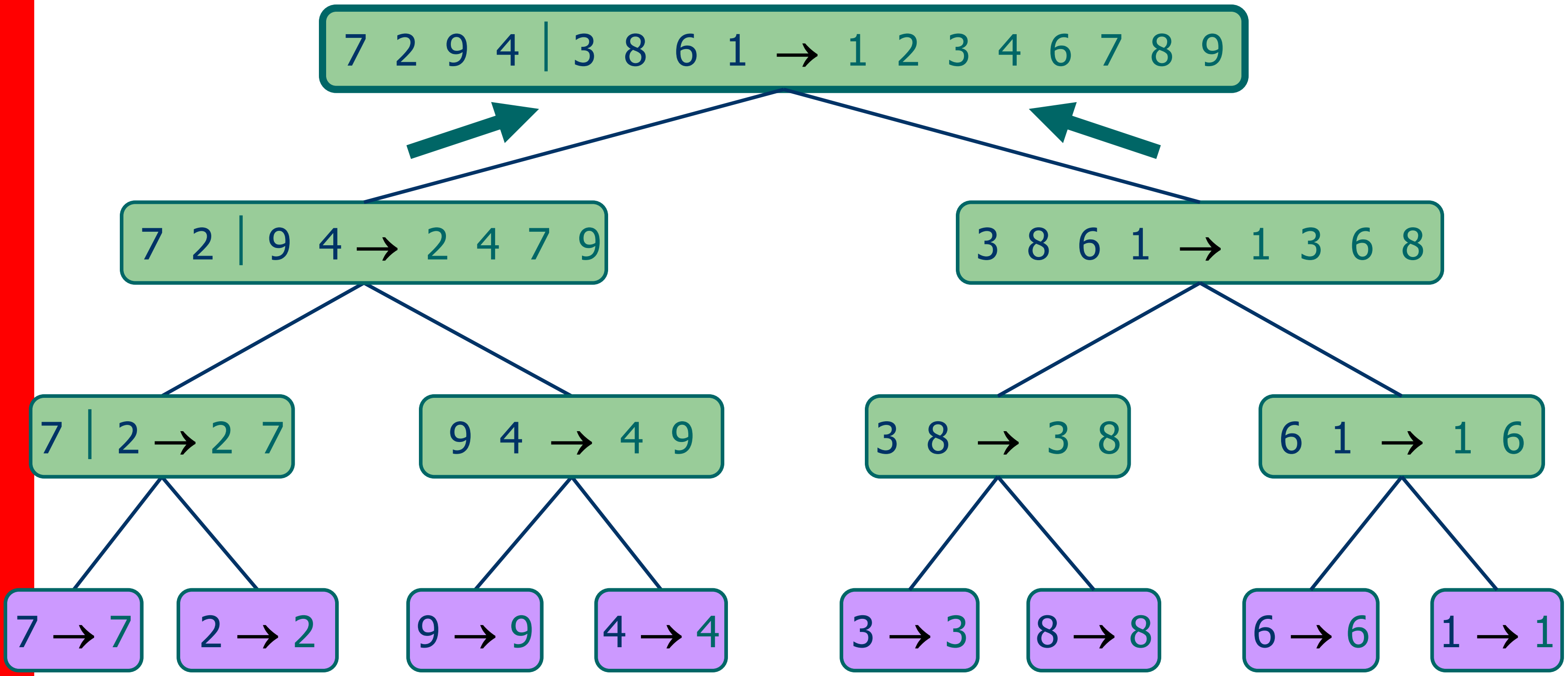
Execution Example (cont.)

- Recursive call, …, merge, merge

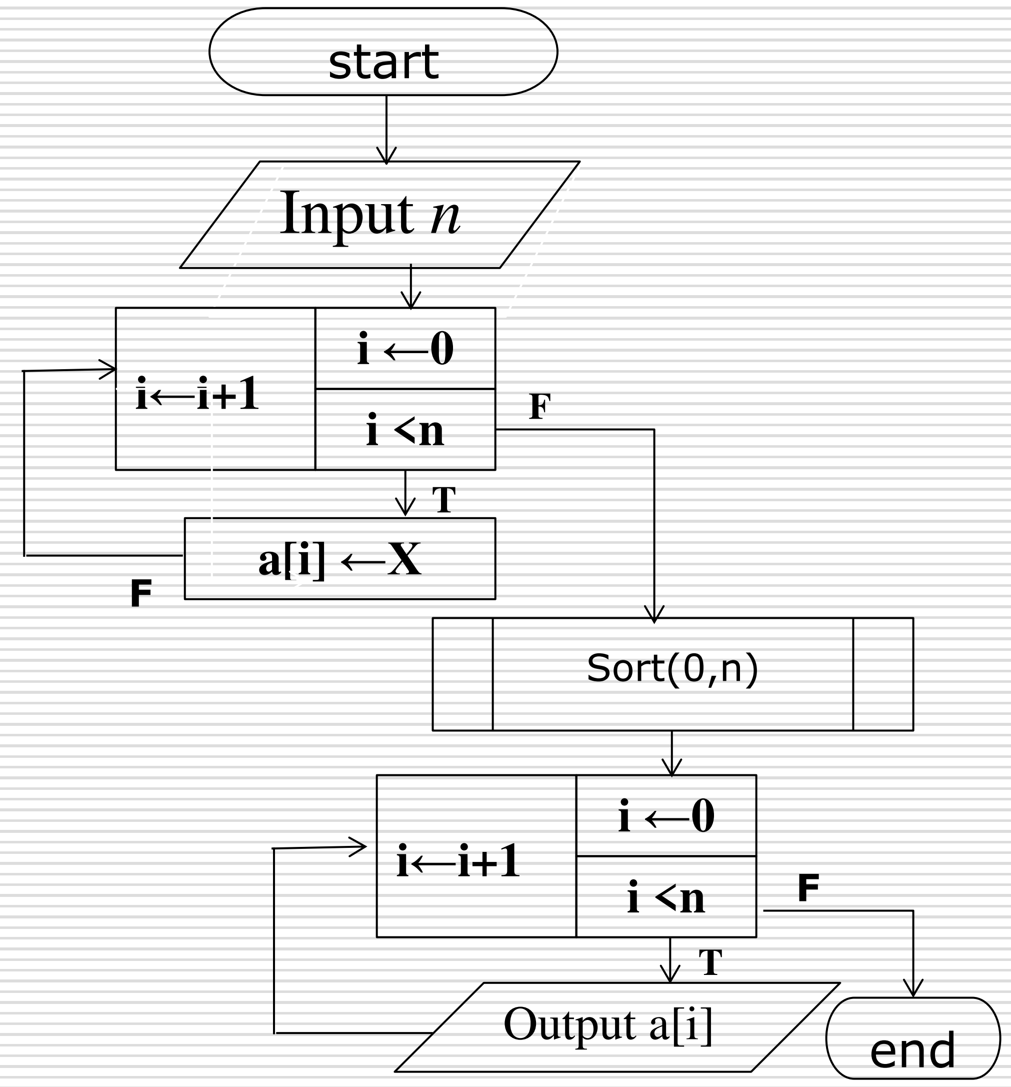


Execution Example (cont.)

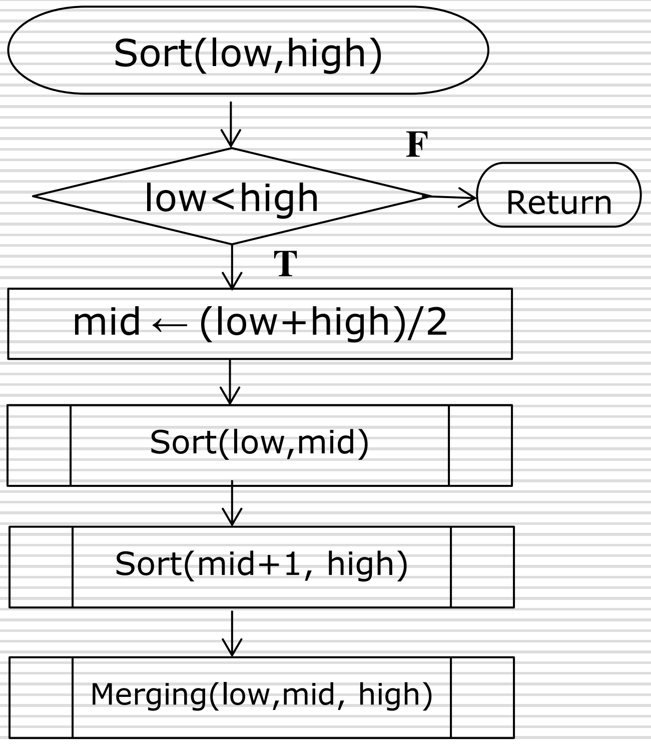
- Merge



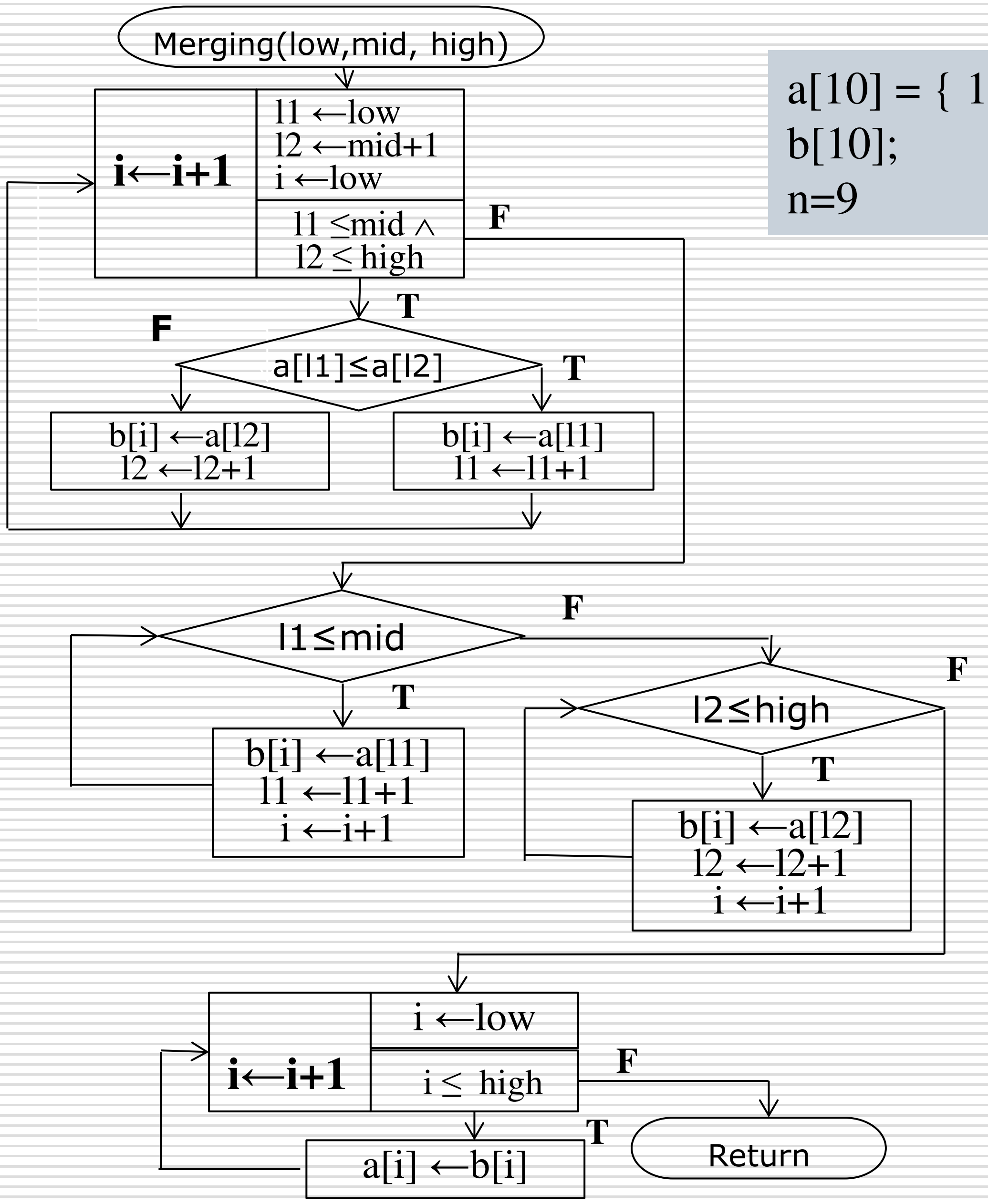
Merge Sort: Algorithm



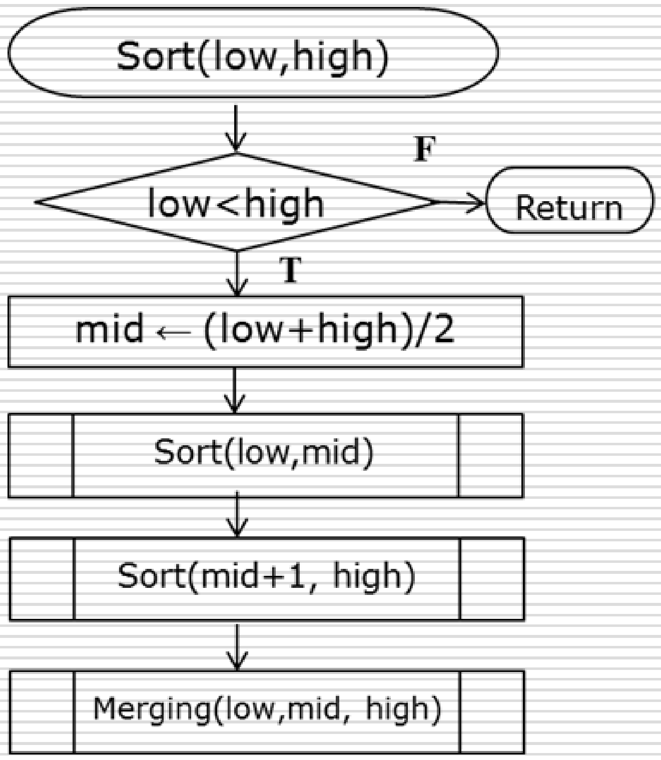
Flowchart



Merge Sort: Algorithm



a[10] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44 };
b[10];
n=9

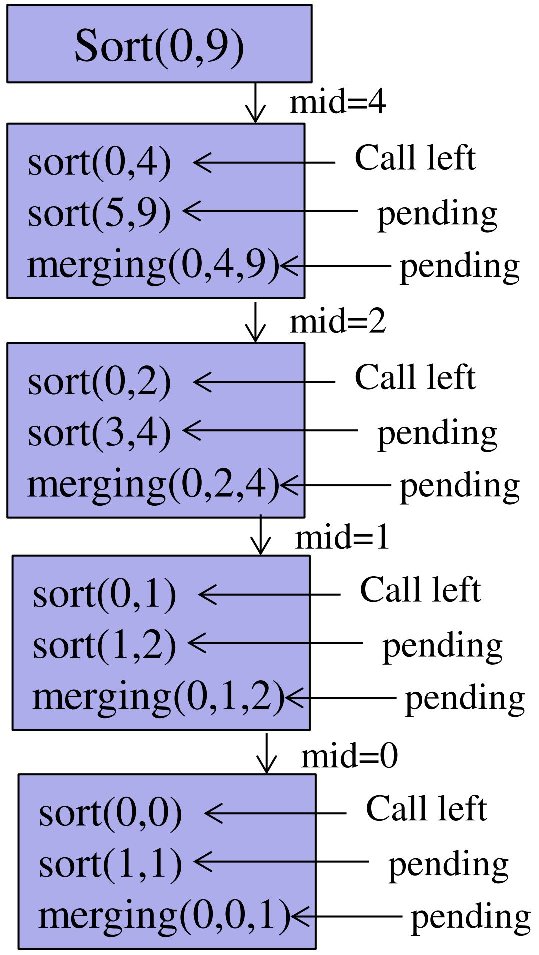


Merge Sort: Program

```
#include <stdio.h>
#define n 9
int a[10] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44};
int b[10];

void merging(int low, int mid, int high) {
   int l1, l2, i;
   for(l1 = low, l2 = mid + 1, i = low; l1  <= mid &&
l2 <= high; i++) {
      if(a[l1] <= a[l2])
         b[i] = a[l1++];
      else
         b[i] = a[l2++];
   }
   while(l1  <= mid)
      b[i++] = a[l1++];

   while(l2  <= high)
      b[i++] = a[l2++];
   for(i = low; i <= high; i++)
      a[i] = b[i];
}
```

```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}

int main() {
   int i;
   printf("List before sorting\n");
   for(i = 0; i <= n; i++)
      printf("%d ", a[i]);
   sort(0, max);
   printf("\nList after sorting\n");

   for(i = 0; i <= n; i++)
      printf("%d ", a[i]);
}
```

Merge Sort: Program Execution

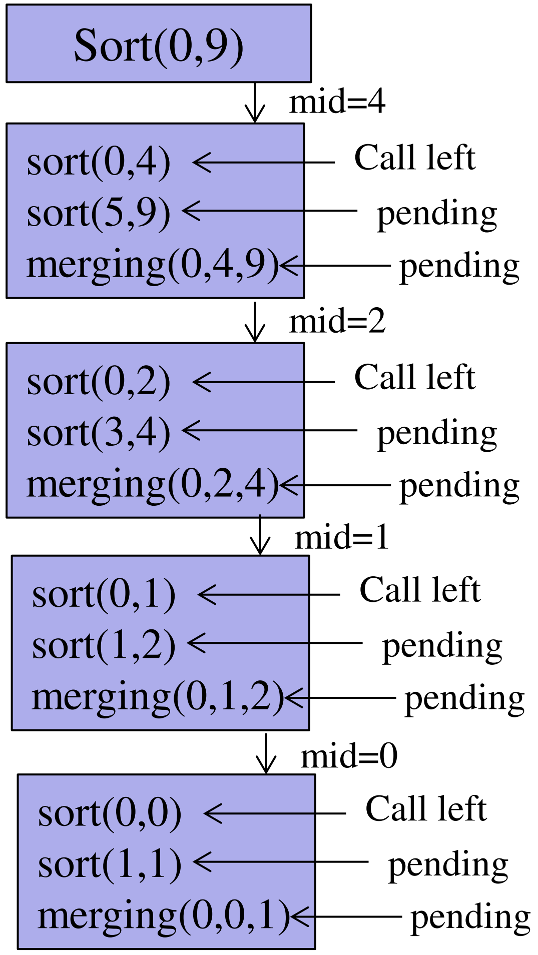


```
int a[10] = { 44, 32, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```

```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
```

Merge Sort: Program Execution

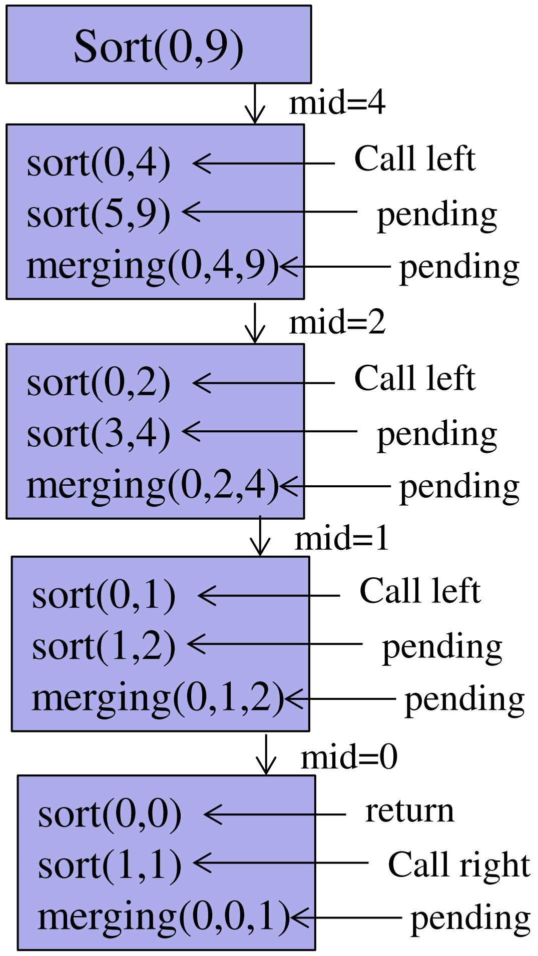
```
int a[10] = { 44, 32, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```



```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
```

Merge Sort: Program Execution

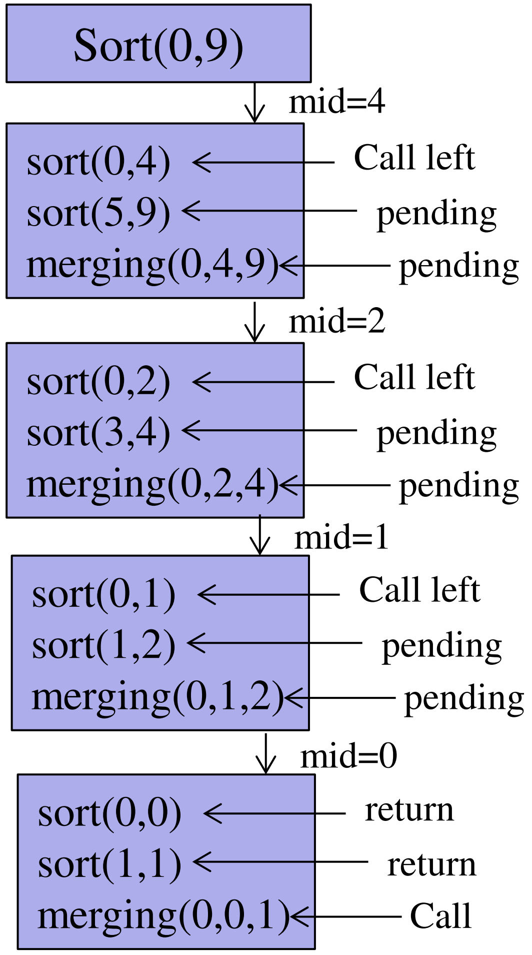
```
int a[10] = { 44, 32, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```



```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
```

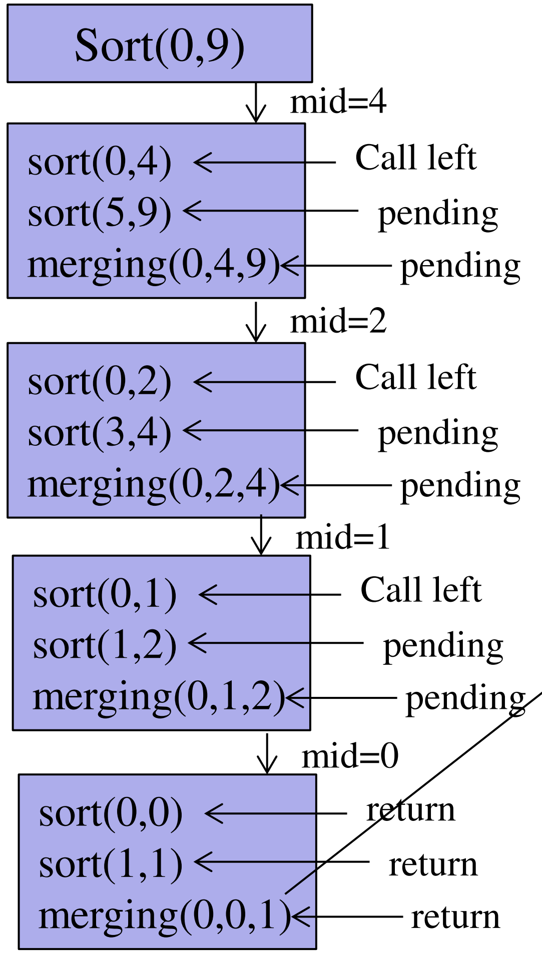
Merge Sort: Program Execution

```
int a[10] = { 44, 32, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```



```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
```

Merge Sort: Program Execution

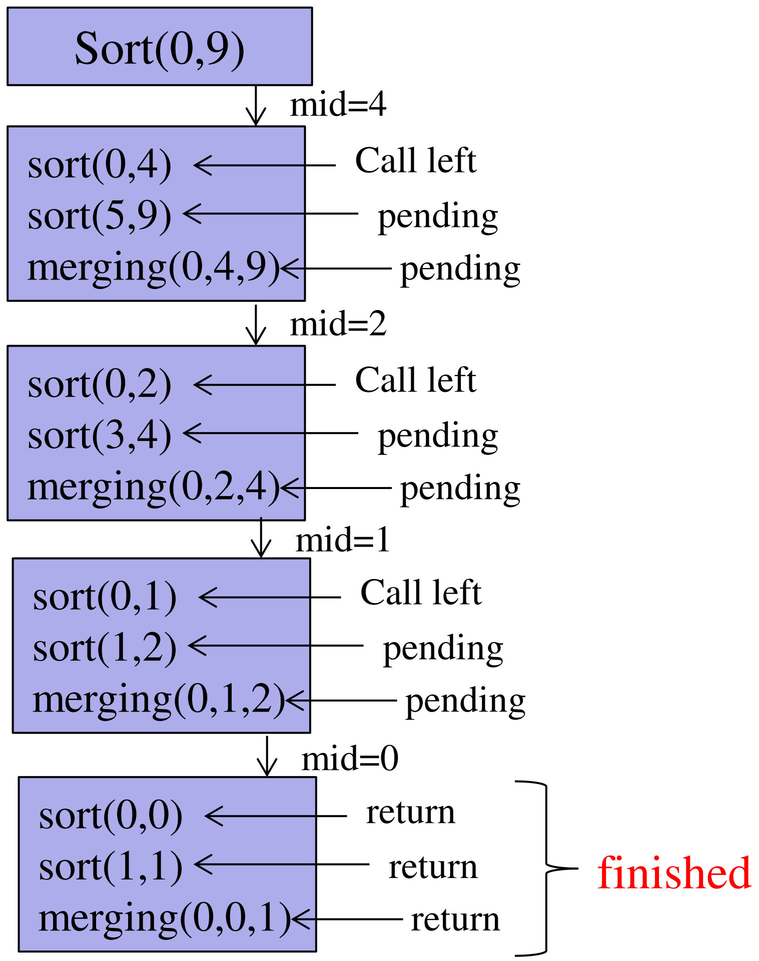


```
int a[10] = { 44, 32, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```

```
int a[10] = { 32, 44, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];={ 32, 44,};
```

```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
```

Merge Sort: Program Execution

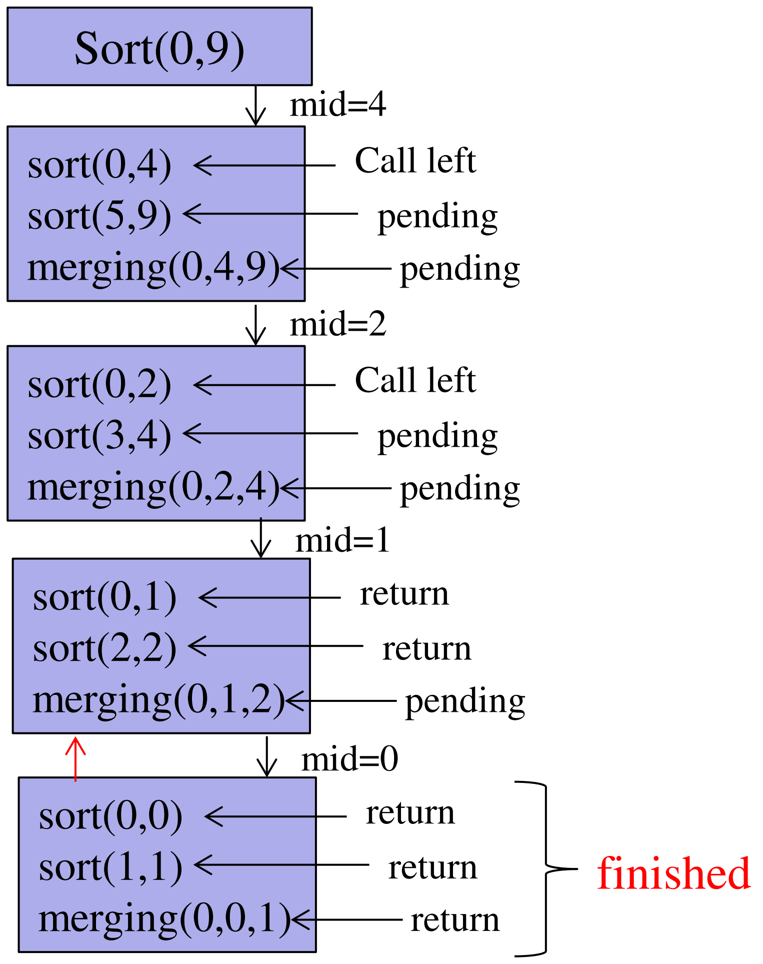


```
int a[10] = { 44, 32, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```

```
int a[10] = { 32, 44, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];={ 32, 44,};
```

```
void sort(int low, int high) {
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}
```

Merge Sort: Program Execution

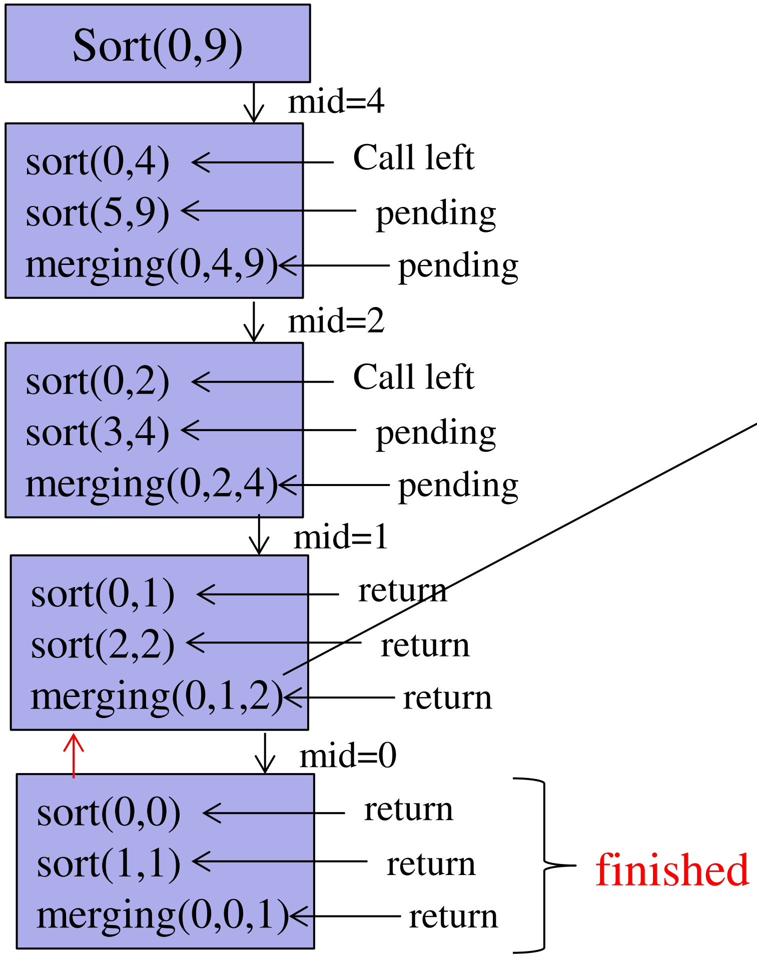


```
int a[10] = { 44, 32, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```

```
int a[10] = { 32, 44, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];={ 32, 44,};
```

```
void sort(int low, int high) {
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}
```

Merge Sort: Program Execution

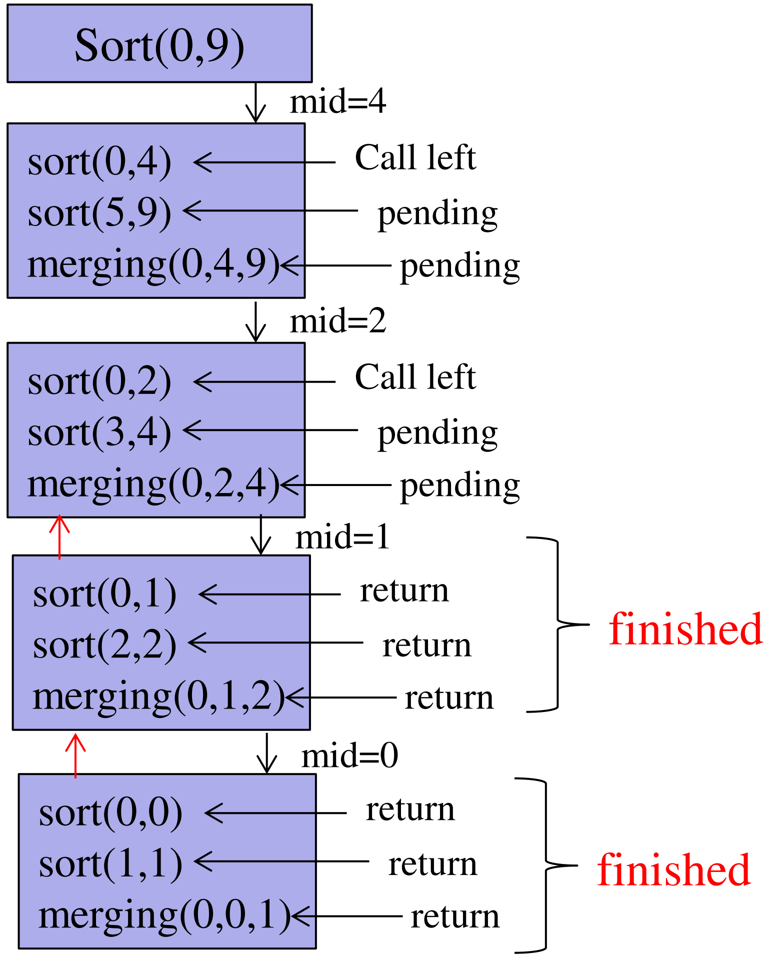


```
int a[10] = { 44, 32, 19, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```

```
int a[10] = { 19, 32, ,44, 26, 27, 36, 30, 15, 12, 24};
int b[10];={ 19, 32, 44,};
```

```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
```

Merge Sort: Program Execution

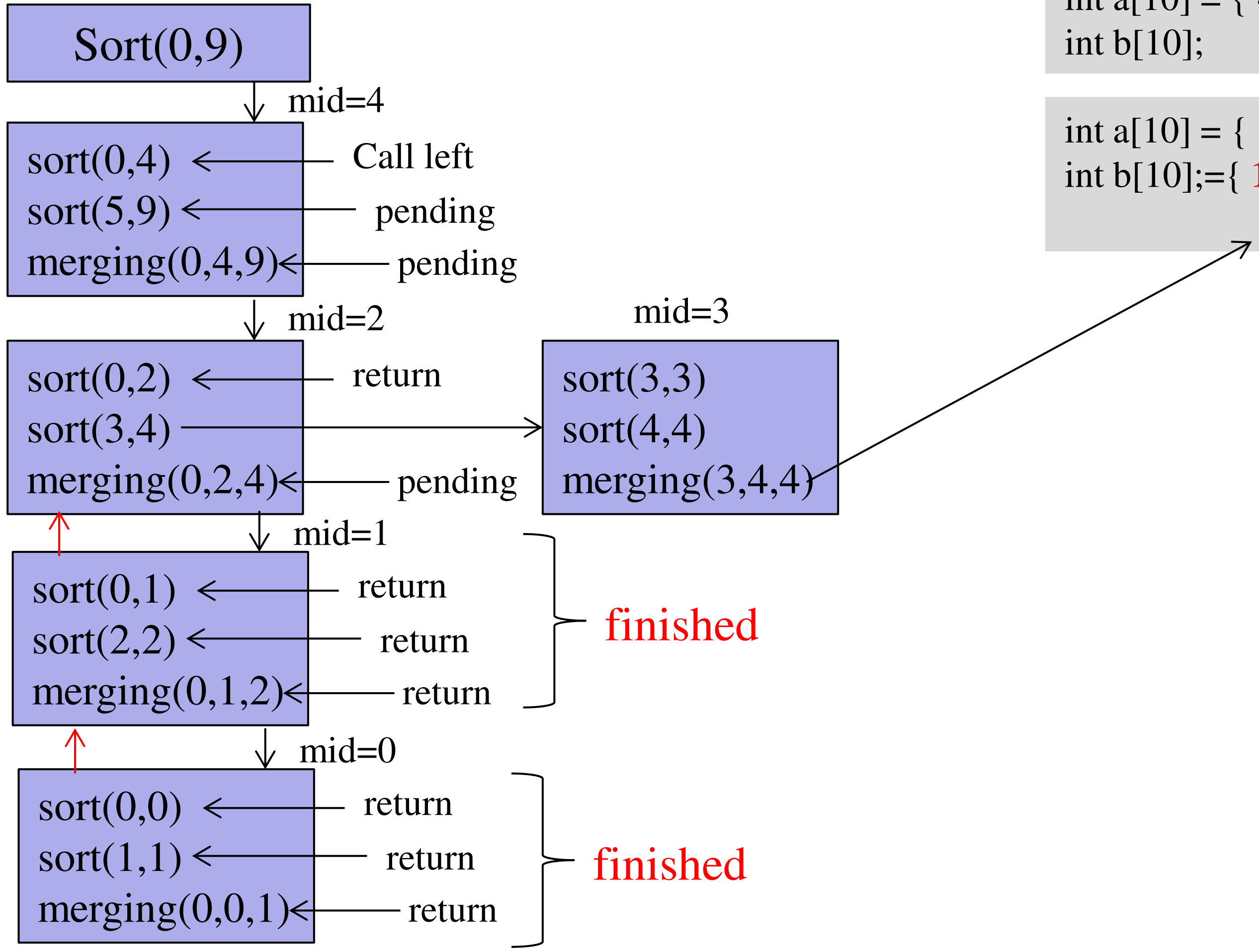


```
int a[10] = { 44, 32, 19, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```

```
int a[10] = { 19, 32, ,44, 26, 27, 36, 30, 15, 12, 24};
int b[10];={ 19, 32, 44,};
```

```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
```

Merge Sort: Program Execution

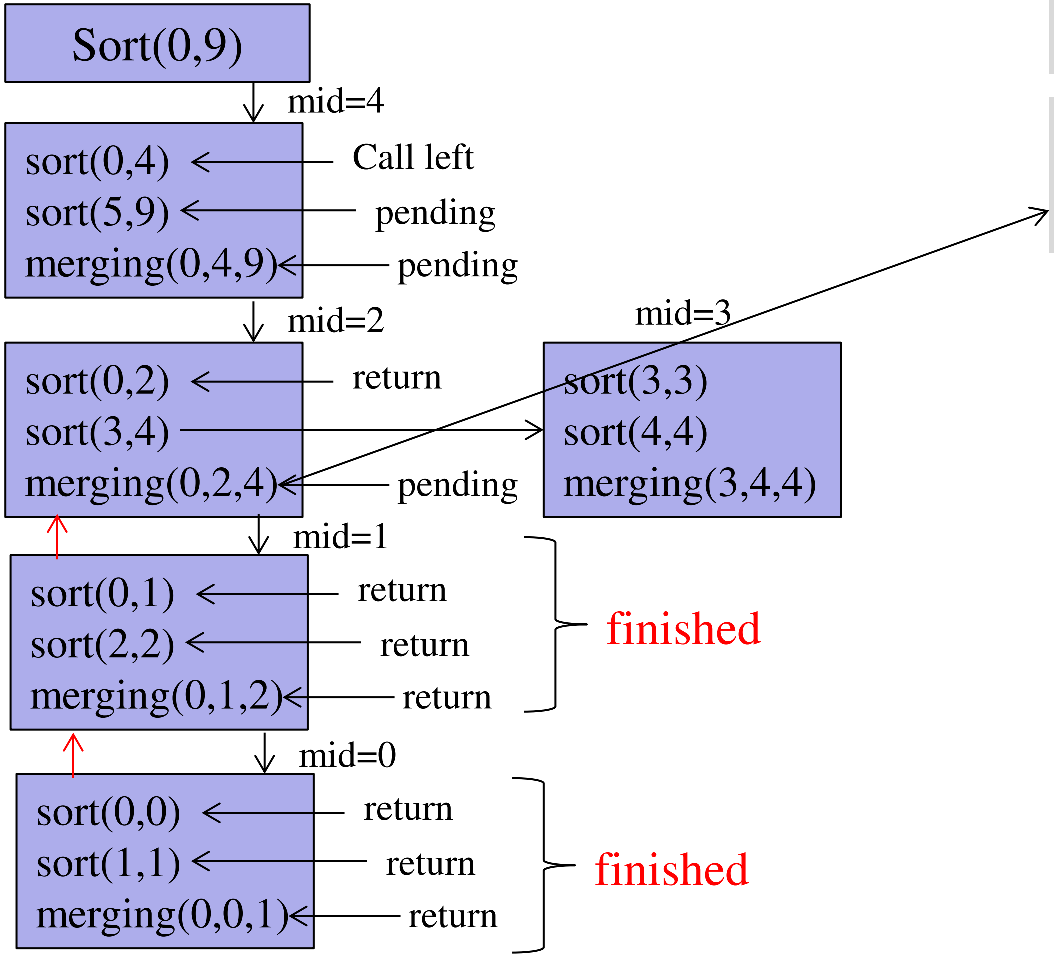


```
int a[10] = { 44, 32, 19, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```

```
int a[10] = { 19, 32, ,44, 26, 27, 36, 30, 15, 12, 24};
int b[10];={ 19, 32, 44,,26,27};
```

```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
```

Merge Sort: Program Execution

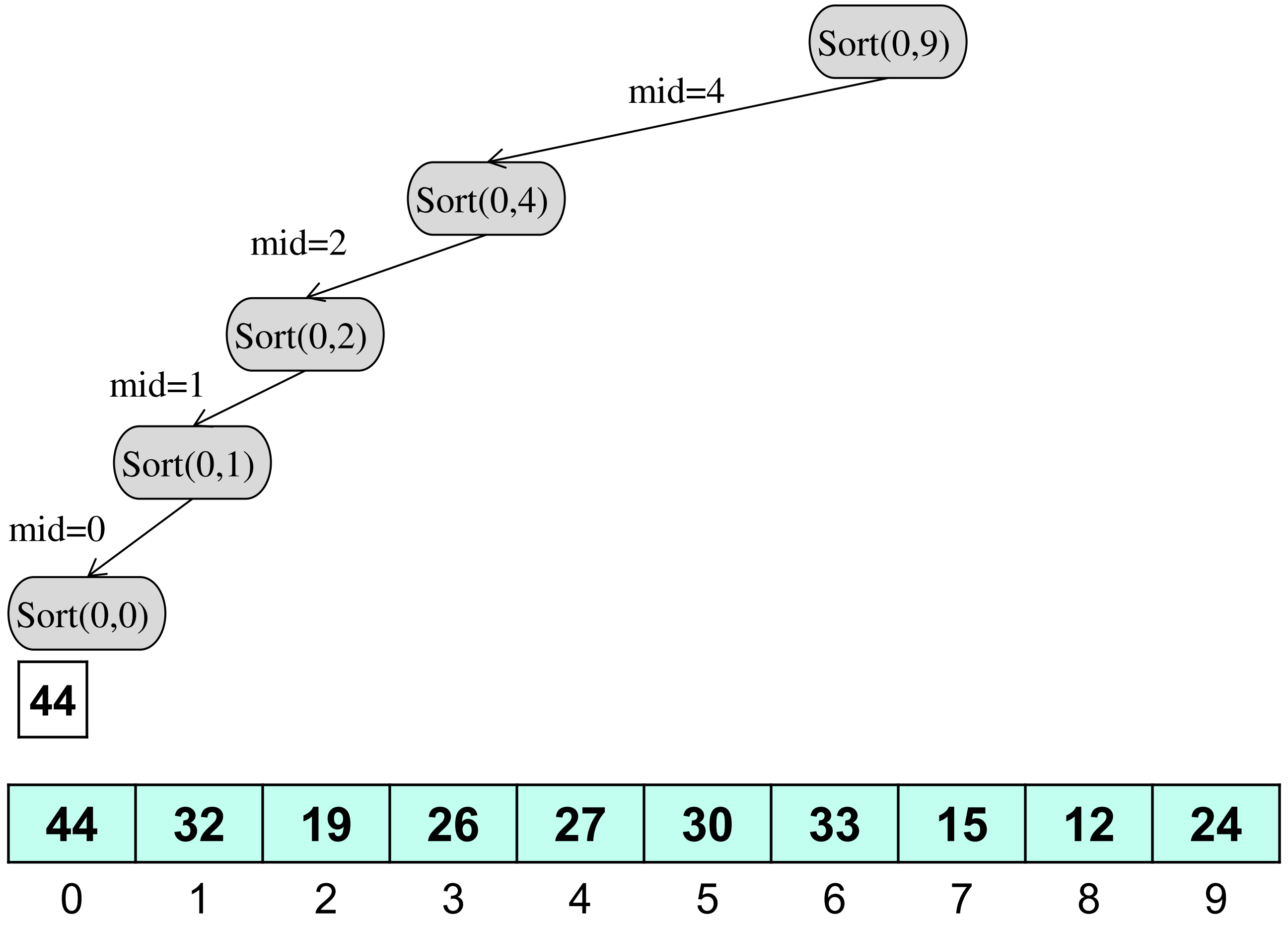


```
int a[10] = { 44, 32, 19, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```

int a[10] = {**19**, 26, 27,, **32, 44**, 36, 30, 15, 12, 24};
int b[10];={ **19**, 26, 27,, **32, 44**};

```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
```

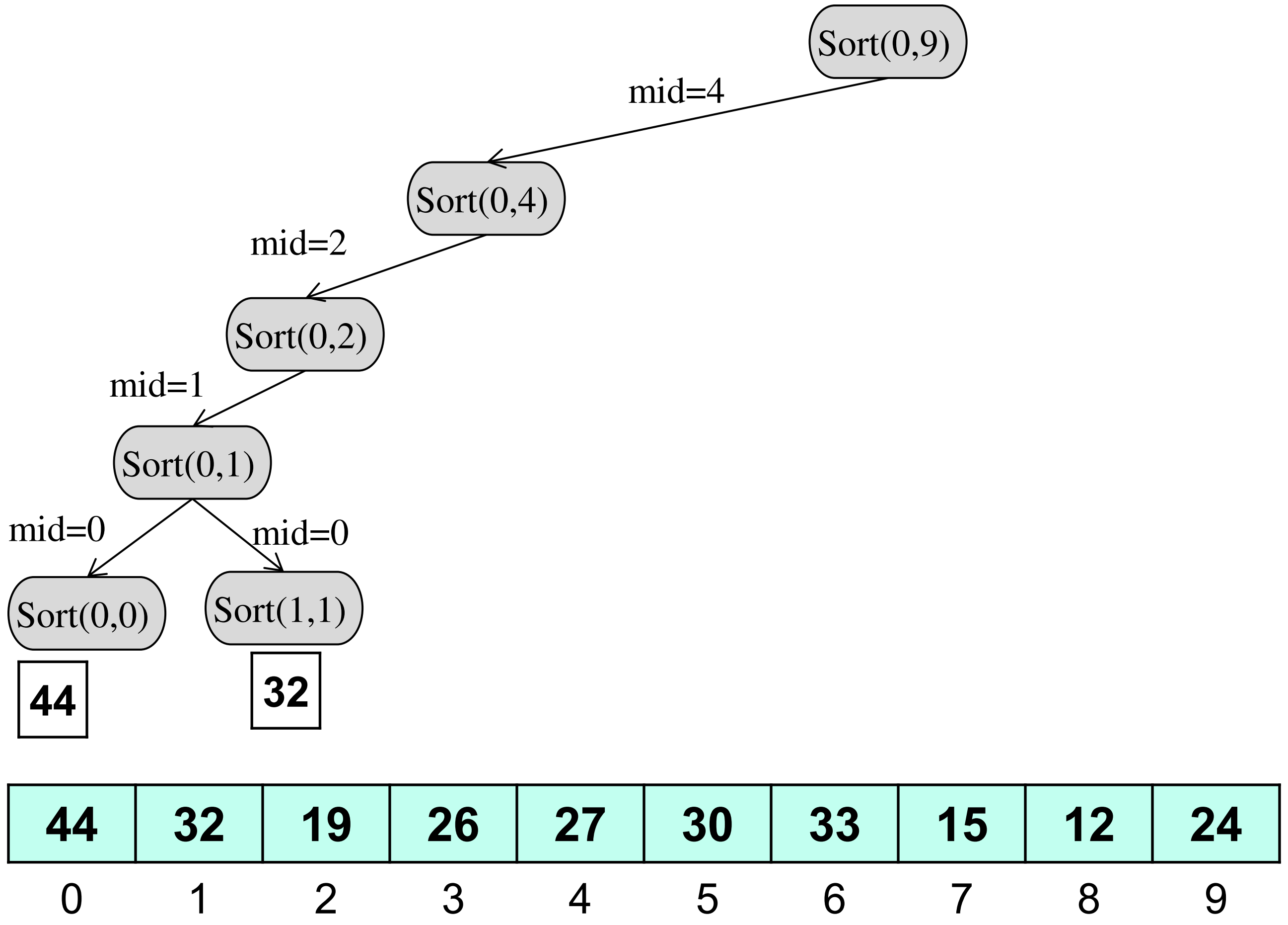
Merge Sort: Program Execution



0	1	2	3	4	5	6	7	8	9
44	32	19	26	27	30	33	15	12	24

```
void sort(int low, int high) {
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}
```

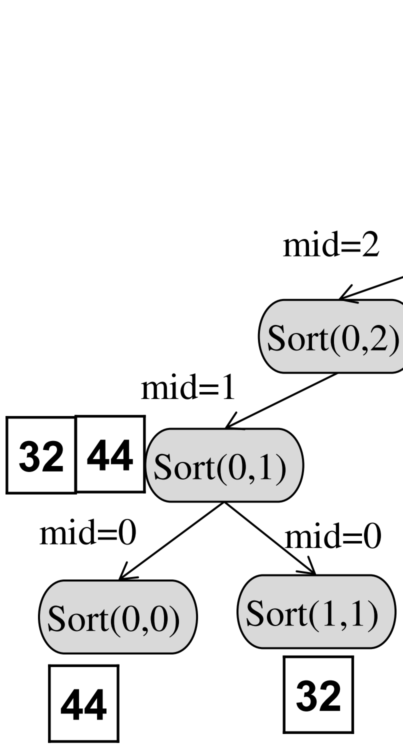
Merge Sort: Program Execution



44	32	19	26	27	30	33	15	12	24
0	1	2	3	4	5	6	7	8	9

```
void sort(int low, int high) {
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}
```

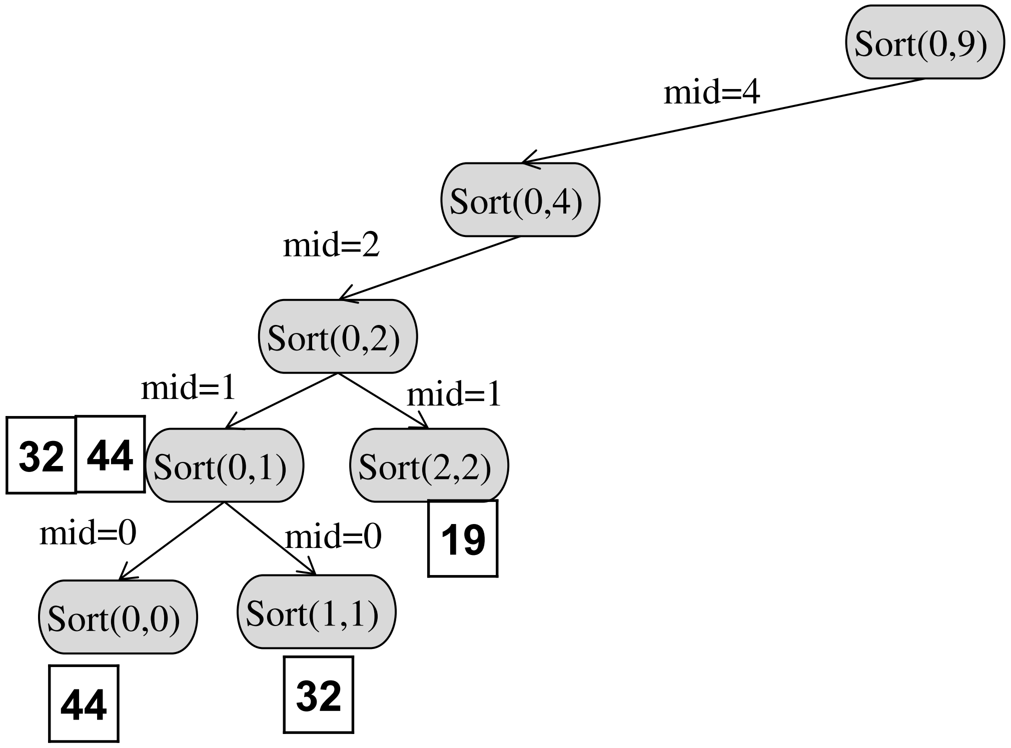
Merge Sort: Program Execution



32	44	19	26	27	30	33	15	12	24
0	1	2	3	4	5	6	7	8	9

```
void sort(int low, int high) {
  int mid;
  if(low < high) {
     mid = (low + high) / 2;
     sort(low, mid);
     sort(mid+1, high);
     merging(low, mid, high);
  } else {
     return;
  }
}
```

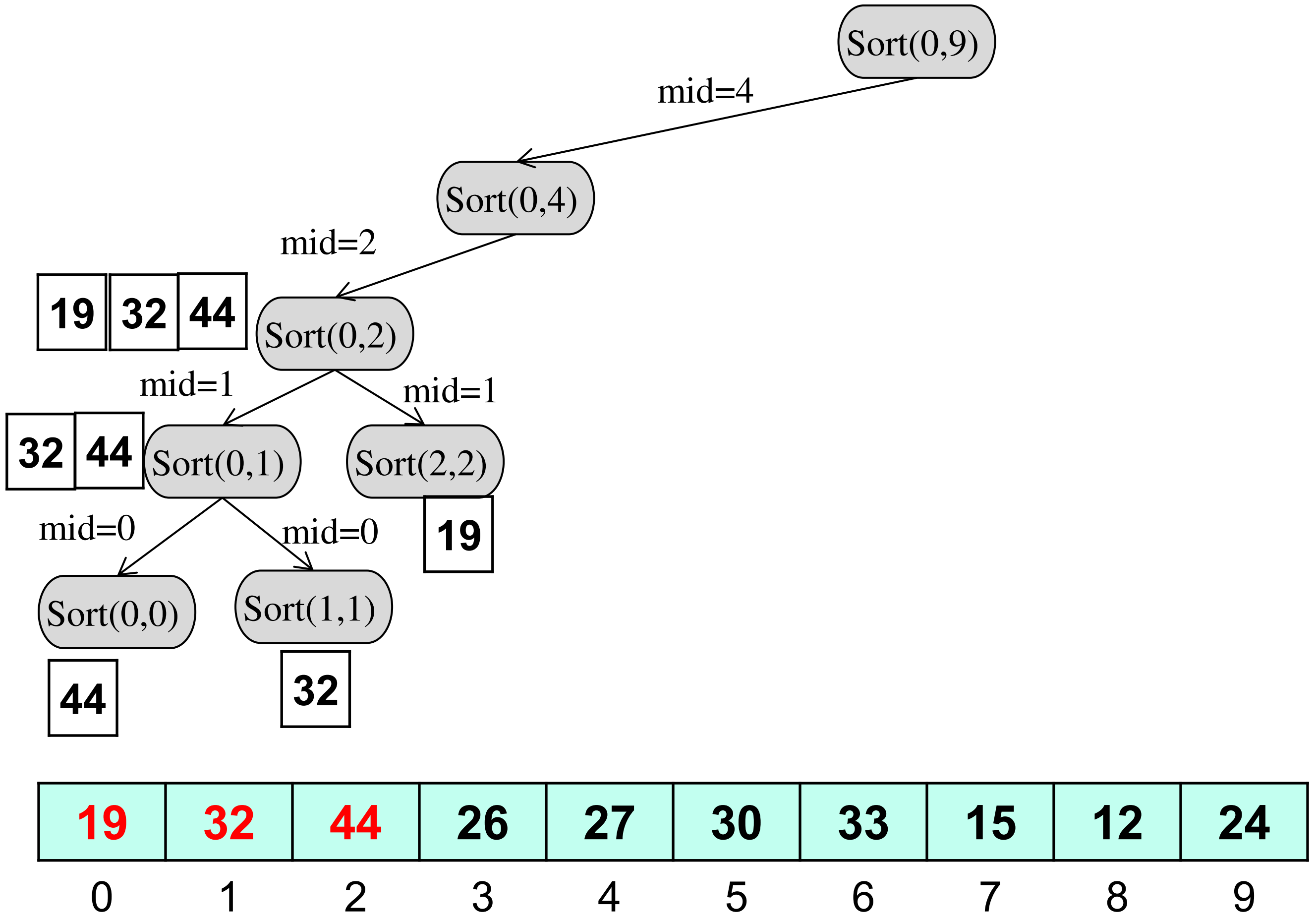
Merge Sort: Program Execution



32	44	19	26	27	30	33	15	12	24
0	1	2	3	4	5	6	7	8	9

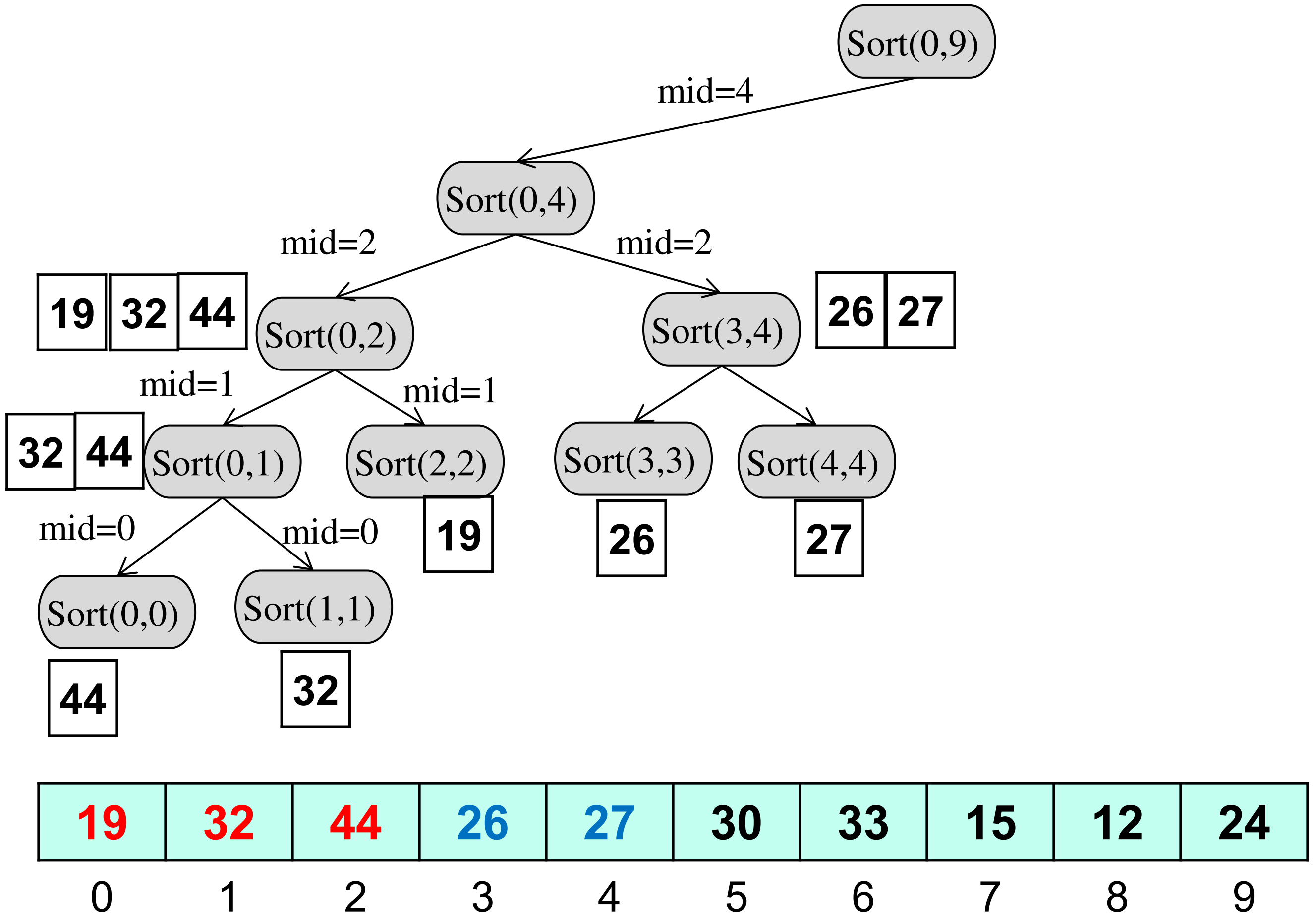
```
void sort(int low, int high) {
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}
```

Merge Sort: Program Execution



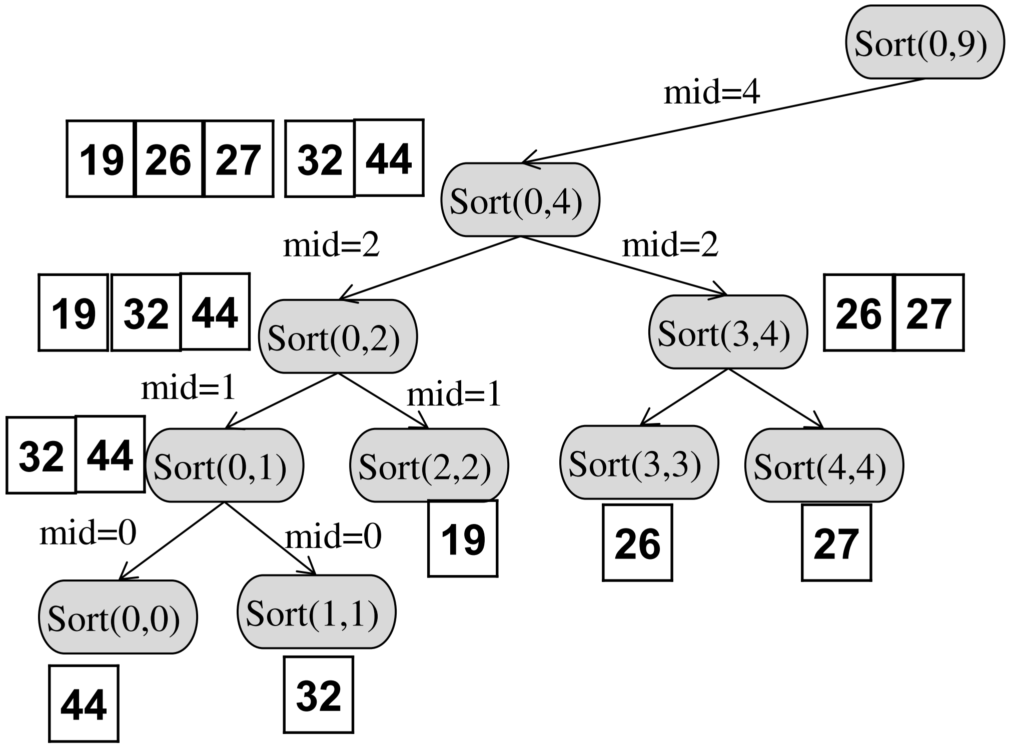
```
void sort(int low, int high) {
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}
```

Merge Sort: Program Execution



```
void sort(int low, int high) {
  int mid;
  if(low < high) {
     mid = (low + high) / 2;
     sort(low, mid);
     sort(mid+1, high);
     merging(low, mid, high);
  } else {
     return;
  }
}
```

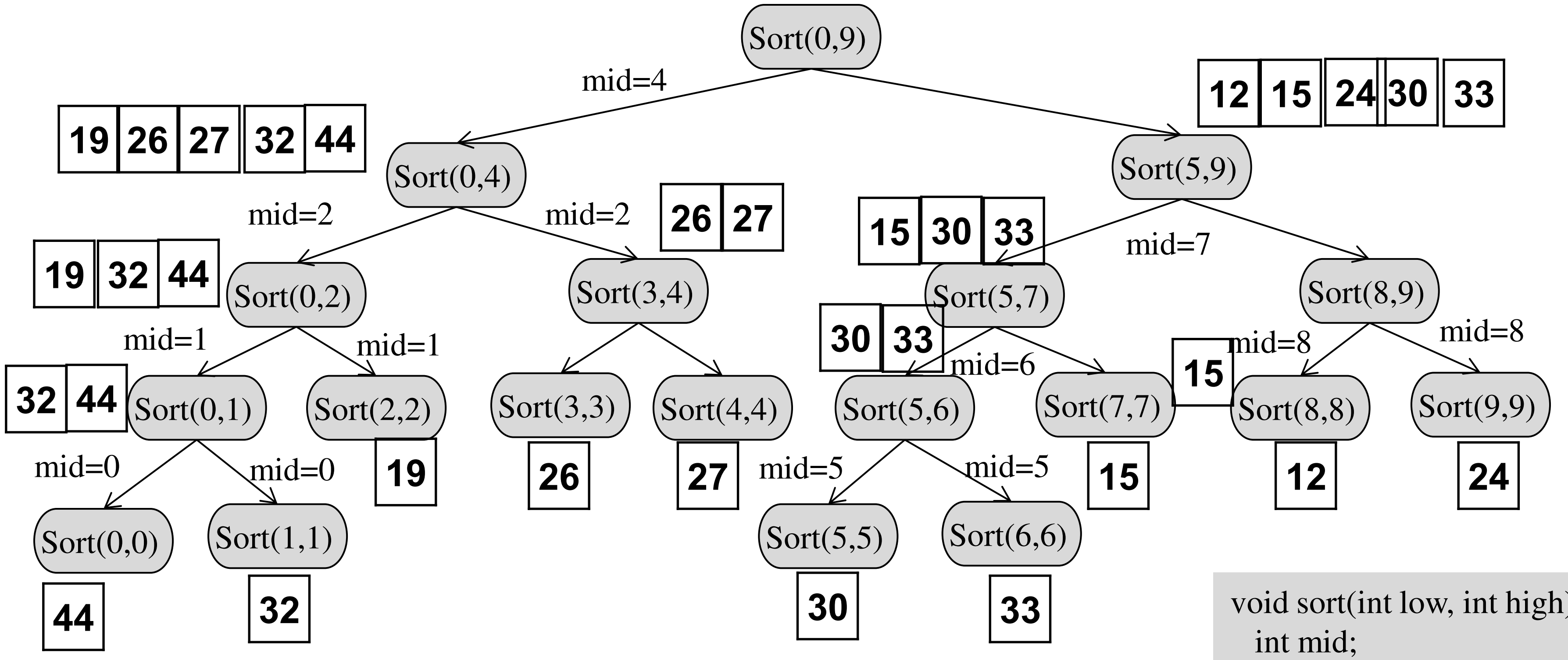
Merge Sort: Program Execution



0	1	2	3	4	5	6	7	8	9
19	26	27	32	44	30	33	15	12	24

```
void sort(int low, int high) {
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}
```

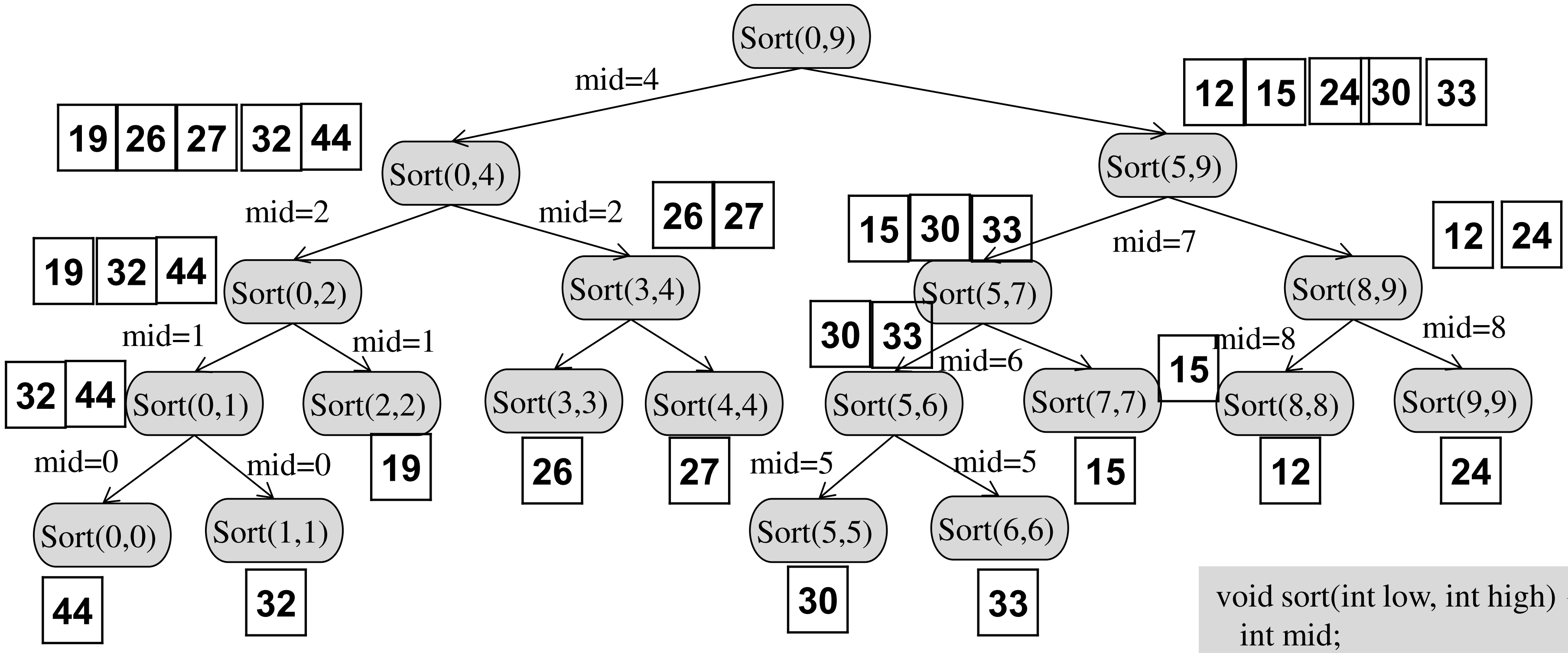
Merge Sort: Program Execution



19	26	27	32	44	30	33	15	12	24
0	1	2	3	4	5	6	7	8	9

```
void sort(int low, int high) {
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}
```

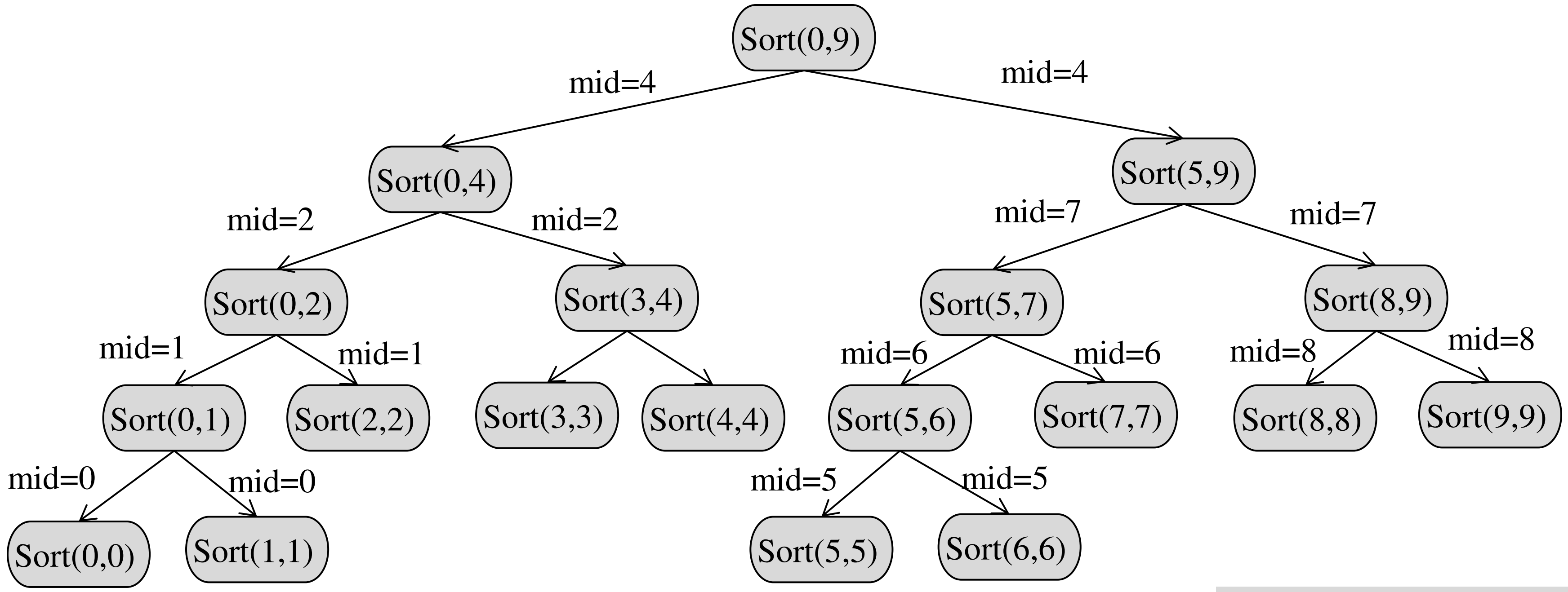
Merge Sort: Program Execution



12	15	19	24	26	27	30	32	33	44
0	1	2	3	4	5	6	7	8	9

```
void sort(int low, int high) {
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}
```

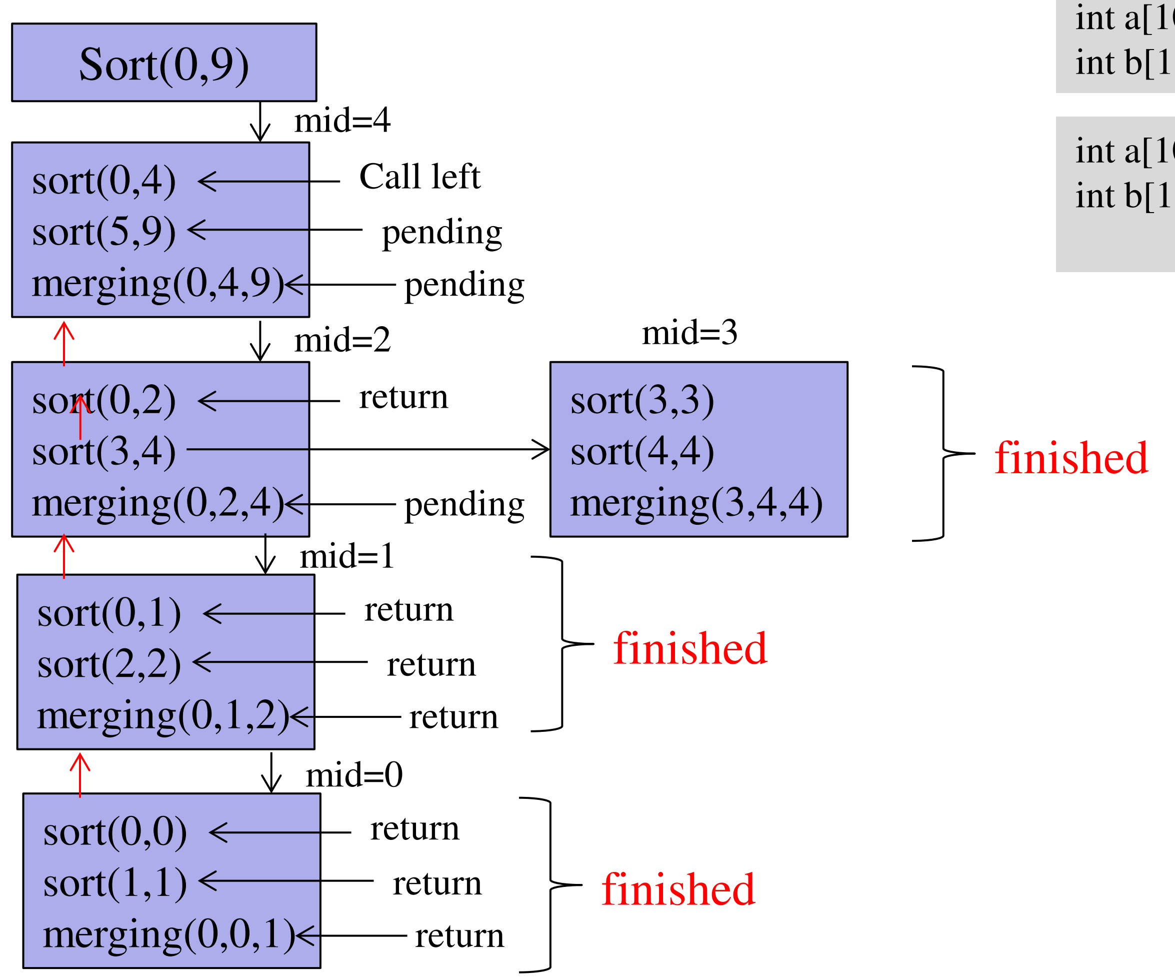
Merge Sort: Program Execution



44	32	19	26	27	30	33	15	12	24
0	1	2	3	4	5	6	7	8	9

```
void sort(int low, int high) {
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}
```

Merge Sort: Program Execution



```
int a[10] = { 44, 32, 19, 26, 27, 36, 30, 15, 12, 24};
int b[10];
```

```
int a[10] = {19, 26, 27,, 32, 44, 36, 30, 15, 12, 24};
int b[10];={ 19, 26, 27,, 32, 44};
```

```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
```

Merge Sort: Program Execution

```
#include <stdio.h>
#define n 9
int a[10] = { 44, 32, 18, 26, 27, 36, 30, 15, 12, 24};
int b[10];

void merging(int low, int mid, int high) {
   int l1, l2, i;
   for(l1 = low, l2 = mid + 1, i = low; l1 <= mid &&
l2 <= high; i++) {
      if(a[l1] <= a[l2])
         b[i] = a[l1++];
      else
         b[i] = a[l2++];
   }
   while(l1 <= mid)
      b[i++] = a[l1++];

   while(l2 <= high)
      b[i++] = a[l2++];
   for(i = low; i <= high; i++)
      a[i] = b[i];
}
```

```
void sort(int low, int high) {
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}

int main() {
   int i;
   printf("List before sorting\n");
   for(i = 0; i <= n; i++)
      printf("%d ", a[i]);
   sort(0, max);
   printf("\nList after sorting\n");

   for(i = 0; i <= n; i++)
      printf("%d ", a[i]);
}
```

Complexity of Merge Sort

Let T(n)= time taken to sort *n* elements
T(n/2)= time taken to sort *left half* elements
T(n/2)= time taken to sort *right half* elements
No. of comparisons needed to merge n/2 elements each is n

A[]	n/2	n/2

$T(n)=T(n/2)+T(n/2)+n$
$\quad =2T(n/2)+n \quad \ldots\ldots(i)$
$T(n/2)=2T(n/4)+n/2$
Substituting T(n/2) in (i)
So $T(n)=2\{2T(n/4)+n/2\}+n$
$\quad =2^2\, T(n/2^2) + 2n$
:
:
$T(n)=2^k\, T(n/2^k) + kn$

We know time needed to sort 1 element, T(1)=1
Let at k step, $n/2^k=1$
➔ $n=2^k$, $k=\log_2 n$
$T(n)=2^k\, T(n/2^k) + kn$
$\quad =nT(1)+ n*\log_2 n$
$\quad =n + n*\log_2 n$

$T(n)=O(n*\log_2 n)$

Complexity of Sorting/Searching Algorithm

Binary Search: **$O(\log_2 n)$**
Sequential Search: **$O(n)$**
Quick Sort: **$O(n\log_2 n)$**
Merge Sort: **$O(n\log_2 n)$**
Insertion Sort: **$O(n^2)$**
Bubble Sort: **$O(n^2)$**
Selection Sort: **$O(n^2)$**
Heap Sort: **$O(n\log_2 n)$**