Iterative Linearithmic Sorting Algorithms

Paolo Camurati

Edited by Josie E. Rodriguez

O(NlogN)

Linearithmic Sorting Algorithms

- Comparison-based sorting algorithms whose complexity is $\Omega(n \log n)$ are **OPTIMAL**
- In general, they are recursive **(topic dealt with in the second year Course):**
  - Merge sort
  - Quick sort
  - Heap sort
- There is an iterative version of **Merge sort:**
  - **Bottom-up Merge sort**

Bottom-up Merge sort

- **An array** containing a **single item** is **sorted by definition**
- **Iteration:**
  - Merge **2 sorted subarrays** into a **sorted array**, whose **size** equals the **sum of the sizes** of the **2 subarrays**
  - Until size **N** of the array to be sorted is reached.

Example

- **Assumption:** size of array to sort is a power of **2** ($N = 2^k$)
- Starting from subarrays of size **1** (**thus sorted by definition**), apply **Merge** to get as a result at each step sorted arrays twice as big (**size m**)
- A temporary array of size **N** is required to store the result of **Merge**
- **Termination:** the **temporary sorted array** has **the same size** of the initial array.

Example

- **Assumption:** size of array to sort is a power of **2** ($N = 2^k$)
- Starting from subarrays of size **1** (**thus sorted by definition**), apply **Merge** to get as a result at each step sorted arrays twice as big (**size m**)
- A temporary array of size **N** is required to store the result of **Merge**
- **Termination:** the **temporary sorted array** has **the same size** of the initial array.

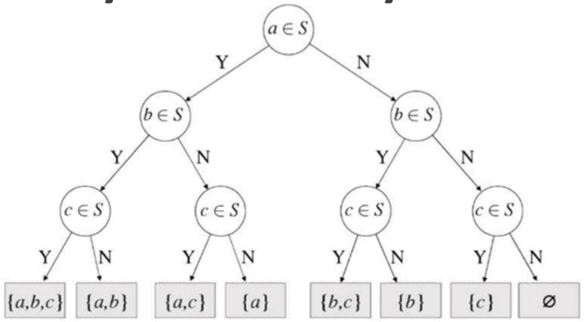


FIGURE 1.2. A decision tree for selecting a subset of $\{a, b, c\}$.

Remark. A picture like this is called a *tree*. (This is not a formal definition; that will follow later.) If you want to know why the tree is growing upside down, ask the computer scientists who introduced this convention. (The conventional wisdom is that they never went out of the room, and so they never saw a real tree.)

Example

12	6	4	5	9	2	3	1

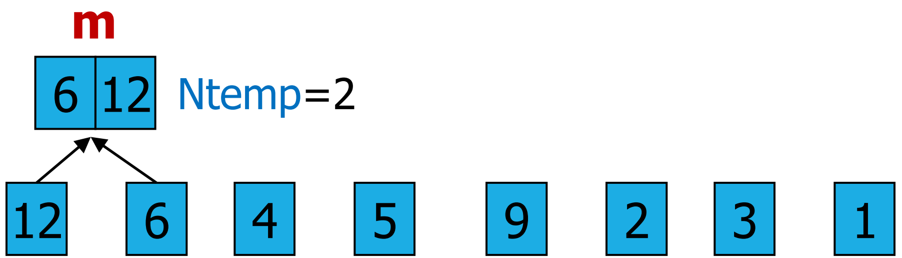
N=8

12	6	4	5	9	2	3	1

Example

12	6	4	5	9	2	3	1

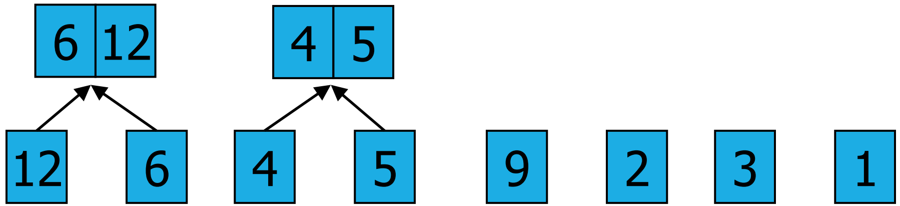
$N=8$



Example

12	6	4	5	9	2	3	1

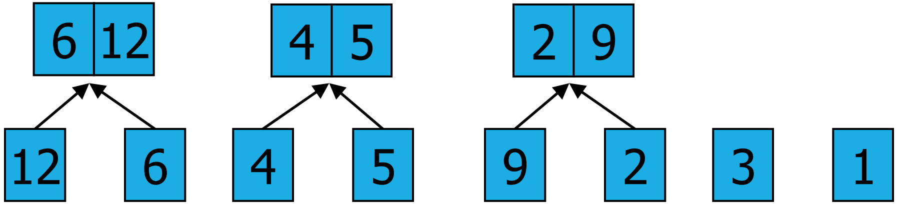
N=8



Example

12	6	4	5	9	2	3	1

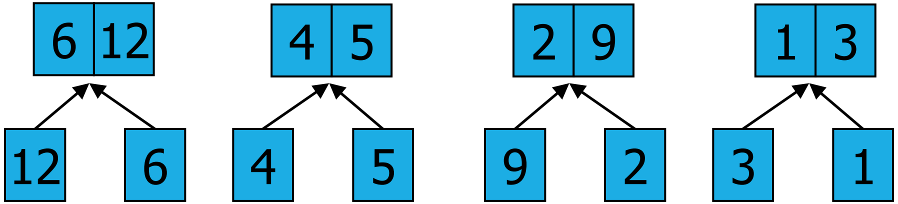
N=8



Example

12	6	4	5	9	2	3	1

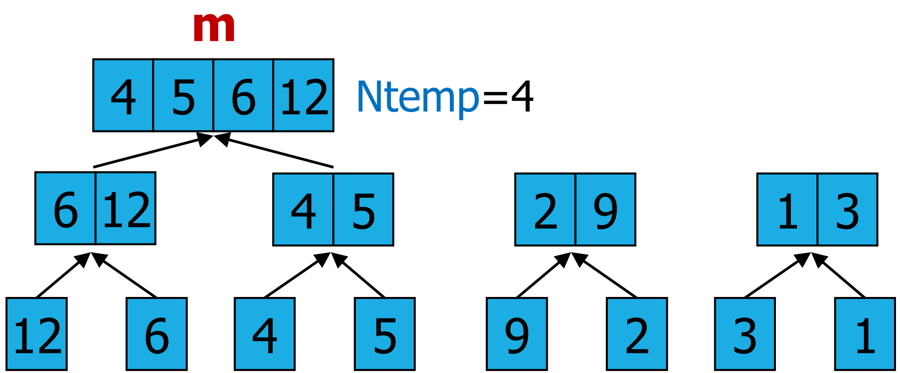
N=8



Example

12	6	4	5	9	2	3	1

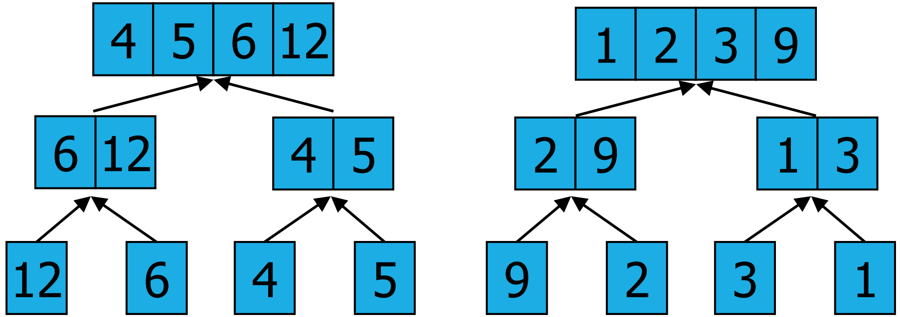
N=8



Example

12	6	4	5	9	2	3	1

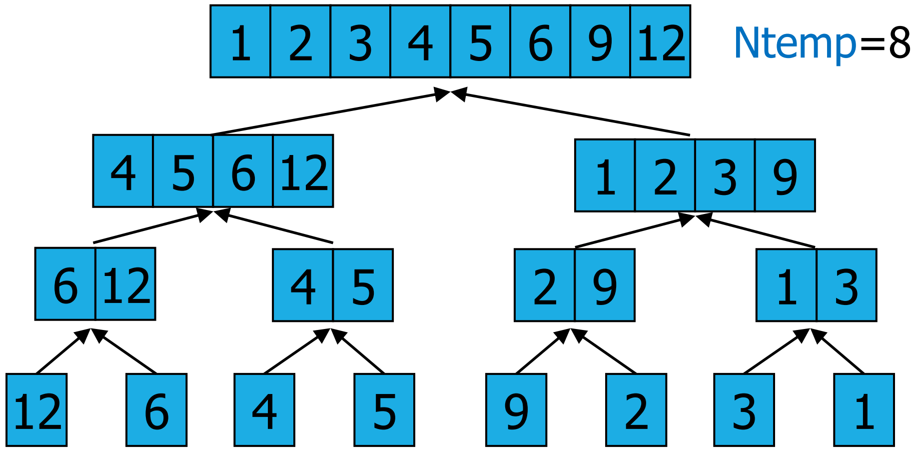
N=8



Example

12	6	4	5	9	2	3	1

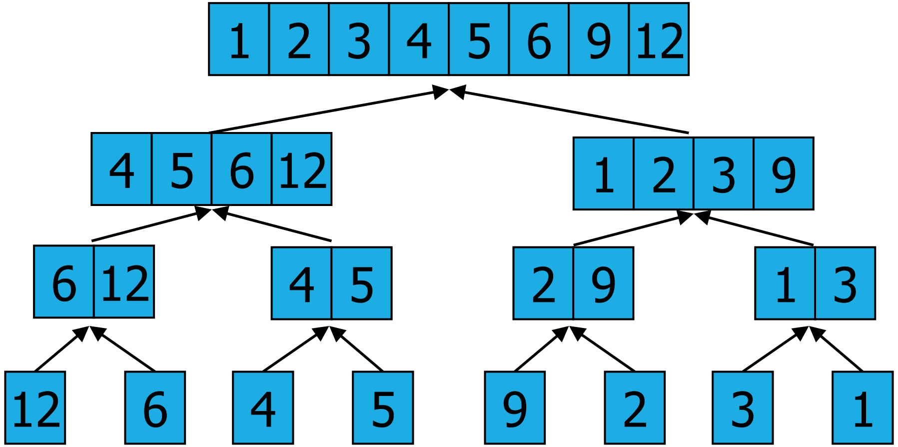
N=8



Ntemp=8

- **Outer loop: m** is initially **1**, it doubles at each step until it becomes **N**
- **Inner loop:** run **Merge** on each pair of sorted and adjacent subarrays of size **m**, obtaining as a result a sorted subarray twice as big (size **2m**)

- Identification of sorted and adjacent subarrays:



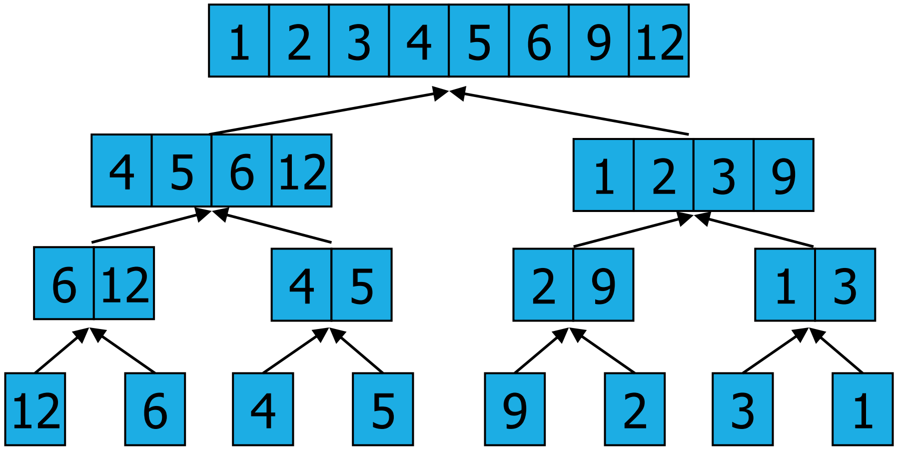
- Identification of sorted and adjacent subarrays:

m=8

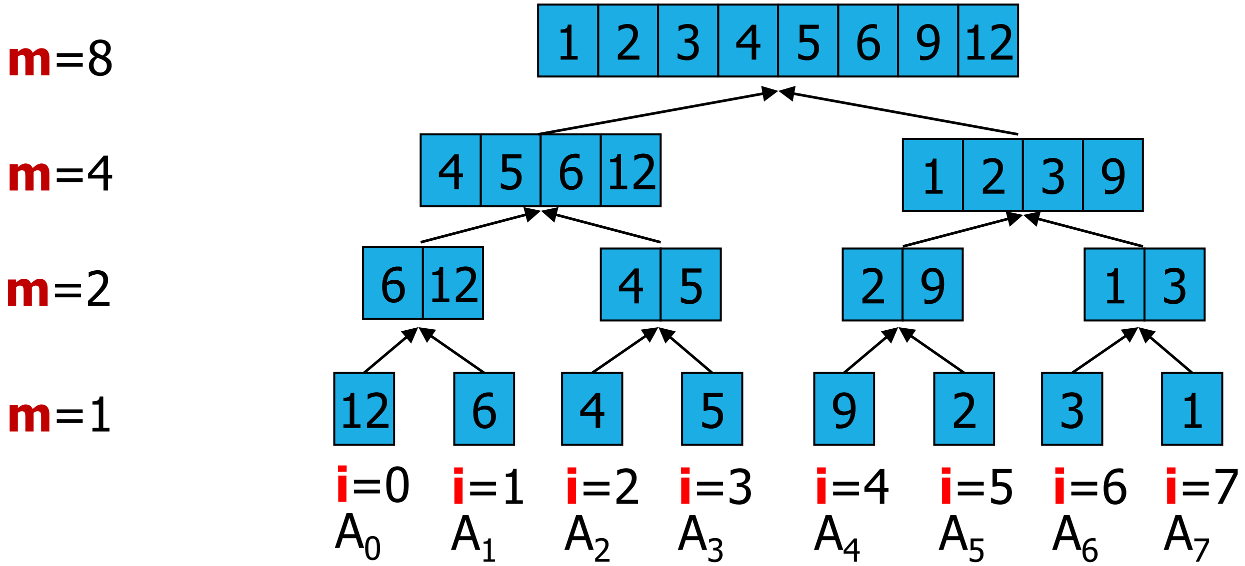
m=4

m=2

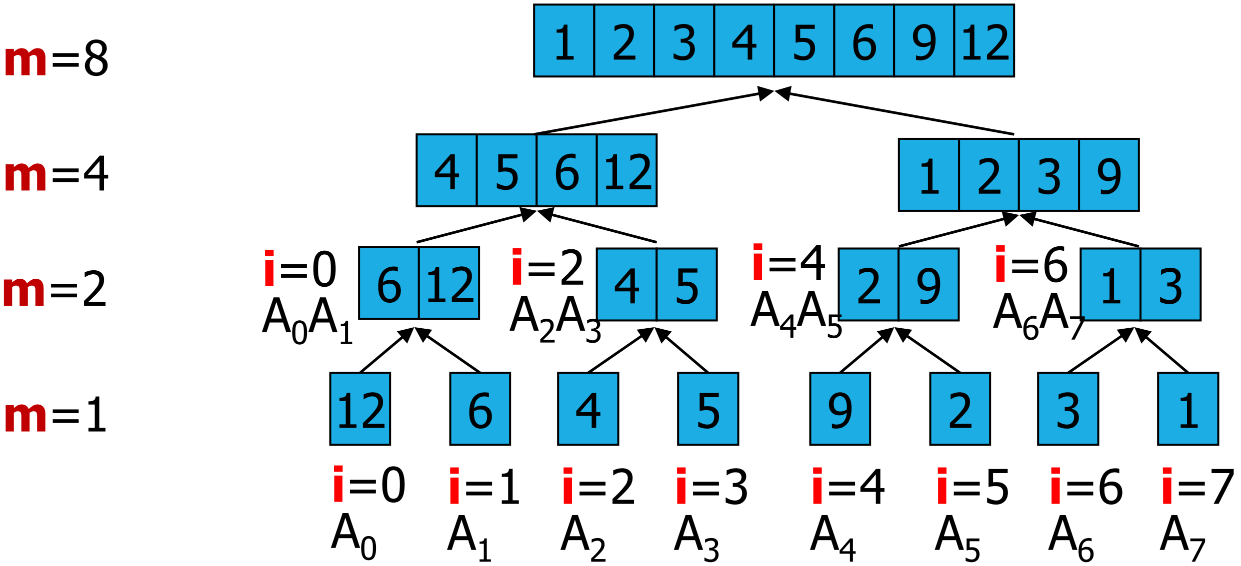
m=1



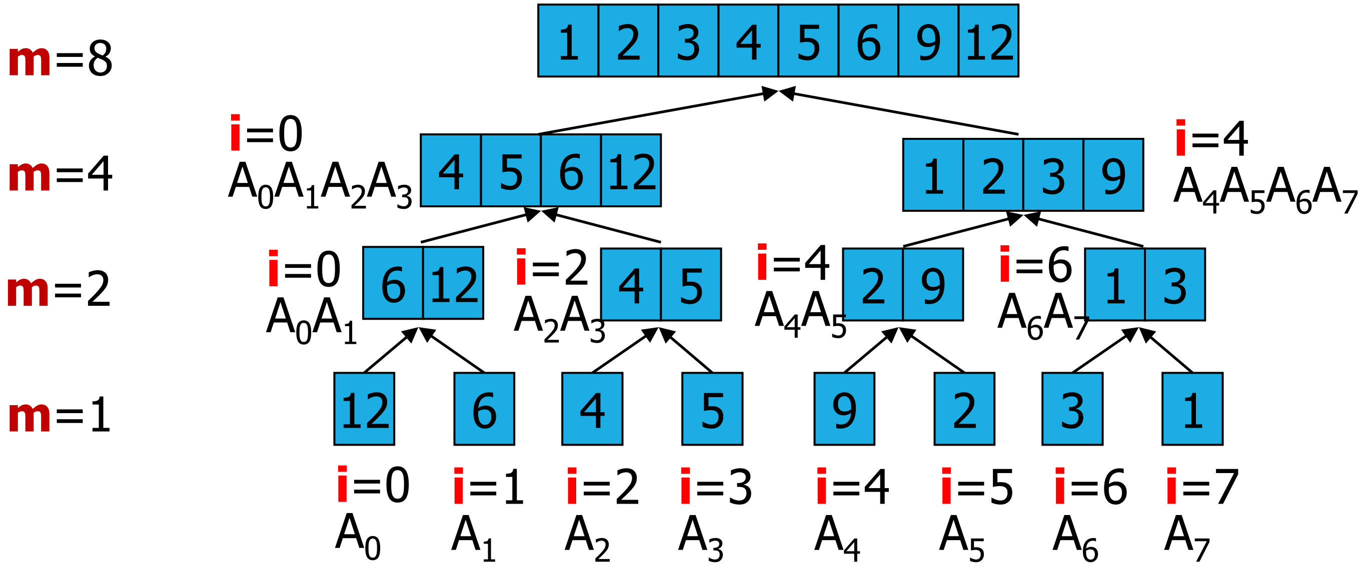
- Identification of sorted and adjacent subarrays:



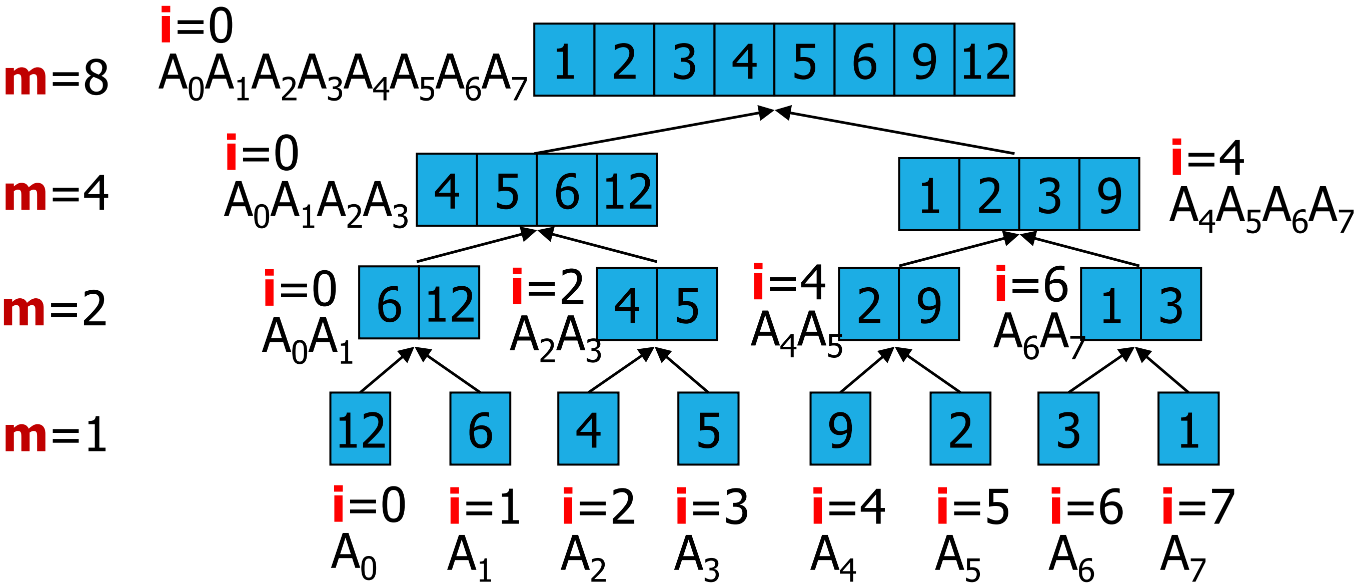
- Identification of sorted and adjacent subarrays:



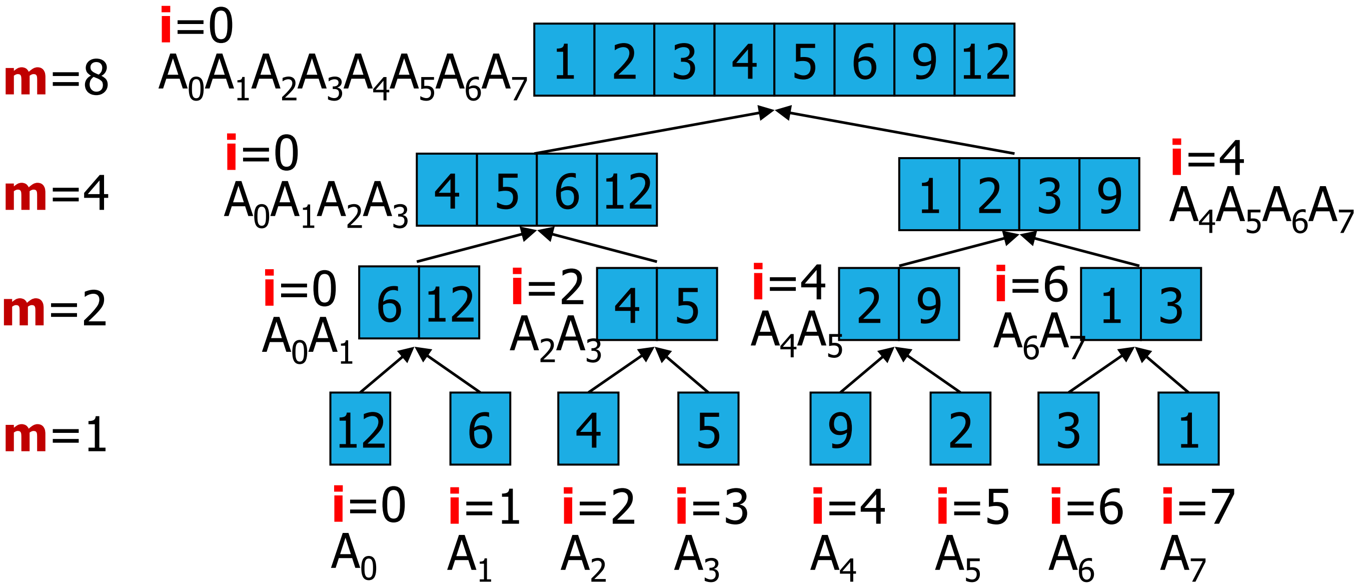
- Identification of sorted and adjacent subarrays:



- Identification of sorted and adjacent subarrays:



- Identification of sorted and adjacent subarrays:



In the inner loop:

q = i + m -1

Left subarray

$A_i \ldots A_q$

Right subarray

$A_{q+1} \ldots A_{q+m}$

temporary array

boundaries

size of sorted subarray doubles

```
void BottomUpMergeSort(int A[], int B[], int N){
  int i, q, m, l=0, r=N-1;
  for (m = 1; m <= r - l; m = m + m)
    for (i = l; i <= r - m; i += m + m) {
      q = i+m-1;
      Merge(A, B, i, q, r);
    }
}
```

merging sorted and adjacent pairs of subarrays $A_i \ldots A_q$, $A_{q+1} \ldots A_{q+m}$

identification of the starting index for the next pair of sorted and adjacent subarrays of size m

2-way Merge

- **Assumption:** size of array **A** is a power of 2 ($N = 2^k$)
- Merging **2 sorted subarrays** of **A** (***2-way***) of size **m** to get a sorted subarray of size **2m**
- Possible to generalize to **k** arrays (**k-way** Merge)
- Index **q** to split in half subarrays in **A**, the result being a **left** and a **right** subarray $q = i + m - 1$
- **Left subarray** with index **i** in the **range** $l \leq i \leq q$
- **Right subarray** with index **j** in the range $q+1 \leq j \leq r$
- Temporary array **B** of size **N** with index **k** in the range $l \leq k \leq r$ to store result of merging process. Array **B** is passed as a parameter.

Approach:

- Walk through **left** and **subarrays** with indices **i** and **j** and through array **B** with index **k**
  - **If left subarray empty:**
    - copy in **B** remaining items from **right subarray**
  - **Else if right subarray empty:**
    - copy in **B** remaining items from **left subarray**
  - **Else**
    - compare current item **A**[**i**] of **left subarray** to current item **A**[**j**] in **right subarray**
      - **if A**[**i**] ≤ **A**[**j**]: copy **A**[**i**] in **B** and increment **i**, **j** unchanged
      - **Else:** copy **A**[**j**] in **B** and increment **j**, **i** unchanged.

2-way Merge: Example

m=4, q=3

$$A \quad \underbrace{\begin{array}{|c|c|c|c|}\hline 4 & 5 & 6 & 12 \\ \hline\end{array}}_{\textstyle m}\begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 9 \\ \hline\end{array}$$

	0	1	2	3	4	5	6	7
A	4	5	6	12	1	2	3	9

⇧ **i**=0 ⇧ **j**=4

2-way Merge: Example

$m=4,\ q=3$

$r=N-1$

A	4	5	6	12	1	2	3	9
	0	1	2	3	4	5	6	7

⇧ $i=0$ ⇧ $j=4$

$i \le q\ \&\&\ j \le r\ \&\&\ A[0] > A[4]$

2-way Merge: Example

m=4, q=3

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

⇧ **i**=0 ⇧ **j**=4

i ≤ q && **j** ≤ r && **A**[0] > **A**[4]

B

1
0

⇧ k=0

2-way Merge: Example

m=4, q=3

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

i=0 j=4

$i \leq q$ && $j \leq r$ && A[0] > A[4]

B

1
0

k=0

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

i=0 j=5

$i \leq q$ && $j \leq r$ && A[0] > A[5]

B

1	2
0	1

k=1

2-way Merge: Example

m=4, q=3

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

⇧ **i**=0 ⇧ **j**=4

i ≤ q && **j** ≤ r && **A**[0] > **A**[4]

B

1
0

⇧ k=0

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

⇧ **i**=0 ⇧ **j**=5

i ≤ q && **j** ≤ r && **A**[0] > **A**[5]

B

1	2
0	1

⇧ k=1

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

⇧ **i**=0 ⇧ **j**=6

i ≤ q && **j** ≤ r && **A**[0] > **A**[6]

B

1	2	3
0	1	2

⇧ k=2

2-way Merge: Example

m=4, q=3

A	4	5	6	12	1	2	3	9
	0	1	2	3	4	5	6	7

⇧ **i**=0 ⇧ **j**=4

i ≤ q && **j** ≤ r && **A**[0] > **A**[4]

B	1
	0

⇧ k=0

A	4	5	6	12	1	2	3	9
	0	1	2	3	4	5	6	7

⇧ **i**=0 ⇧ **j**=5

i ≤ q && **j** ≤ r && **A**[0] > **A**[5]

B	1	2
	0	1

⇧ k=1

A	4	5	6	12	1	2	3	9
	0	1	2	3	4	5	6	7

⇧ **i**=0 ⇧ **j**=6

i ≤ q && **j** ≤ r && **A**[0] > **A**[6]

B	1	2	3
	0	1	2

⇧ k=2

A	4	5	6	12	1	2	3	9
	0	1	2	3	4	5	6	7

⇧ **i**=0 ⇧ **j**=7

i ≤ q && **j** ≤ r && **A**[0] ≤ **A**[7]

B	1	2	3	4
	0	1	2	3

⇧ k=3

2-way Merge: Example

m=4, q=3

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

i=1 **j**=7

2-way Merge: Example

m=4, q=3

A

0	1	2	3	4	5	6	7
4	5	6	12	1	2	3	9

i=1   j=7

i ≤ q && j ≤ r && A[1] ≤ A[7]

2-way Merge: Example

m=4, q=3

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

⇧ **i**=1 ⇧ **j**=7

i ≤ q && **j** ≤ r && **A**[1] ≤ **A**[7]

B

1	2	3	4	5
0	1	2	3	4

⇧ k=4

2-way Merge: Example

m=4, q=3

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

⇧ **i**=1 ⇧ **j**=7

i ≤ q && **j** ≤ r && **A**[1] ≤ **A**[7]

B

1	2	3	4	5
0	1	2	3	4

⇧ k=4

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

⇧ **i**=2 ⇧ **j**=7

i ≤ q && **j** ≤ r && **A**[2] ≤ **A**[7]

B

1	2	3	4	5	6
0	1	2	3	4	5

⇧ k=5

2-way Merge: Example

m=4, q=3

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

⇧ **i**=1 ⇧ **j**=7

i ≤ q && **j** ≤ r && **A**[1] ≤ **A**[7]

B

1	2	3	4	5
0	1	2	3	4

⇧ k=4

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

⇧ **i**=2 ⇧ **j**=7

i ≤ q && **j** ≤ r && **A**[2] ≤ **A**[7]

B

1	2	3	4	5	6
0	1	2	3	4	5

⇧ k=5

A

4	5	6	12	1	2	3	9
0	1	2	3	4	5	6	7

⇧ **i**=3 ⇧ **j**=7

i ≤ q && **j** ≤ r && **A**[3] > **A**[7]

B

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

k=6 ⇧

2-way Merge: Example

m=4, q=3

A

0	1	2	3	4	5	6	7
4	5	6	12	1	2	3	9

i=1 j=7

i ≤ q && j ≤ r && A[1] ≤ A[7]

B

0	1	2	3	4
1	2	3	4	5

k=4

A

0	1	2	3	4	5	6	7
4	5	6	12	1	2	3	9

i=2 j=7

i ≤ q && j ≤ r && A[2] ≤ A[7]

B

0	1	2	3	4	5
1	2	3	4	5	6

k=5

A

0	1	2	3	4	5	6	7
4	5	6	12	1	2	3	9

i=3 j=7

i ≤ q && j ≤ r && A[3] > A[7]

B

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

k=6

A

0	1	2	3	4	5	6	7
4	5	6	12	1	2	3	9

i=3 j=8

i ≤ q && j > r

B

0	1	2	3	4	5	6	7
1	2	3	4	5	6	9	12

k=7

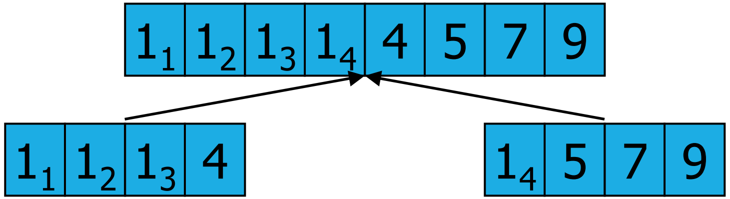
```
void Merge(int A[], int B[], int l, int q, int r) {
  int i, j, k;
  i = l;
  j = q+1;
  for (k = l; k <= r; k++)
    if (i > q)
      B[k] = A[j++];
    else if (j > r)
      B[k] = A[i++];
    else if ((A[i]< A[j]) || (A[i]== A[j]))
      B[k] = A[i++];
    else
      B[k] = A[j++];
  for ( k = l; k <= r; k++ )
    A[k] = B[k];
  return;
}
```

Left subarray empty

Right subarray empty

Merge sort Features

- Not in-place, a temporary array **B** of size **N** is required
- Stable: the **Merge function** copies from the **left subarray** in case of duplicate keys:



If we omit condition **A**[**i**] == **A**[**j**] in statement

```
if ((A[i]< A[j]) || (A[i]== A[j]))
```

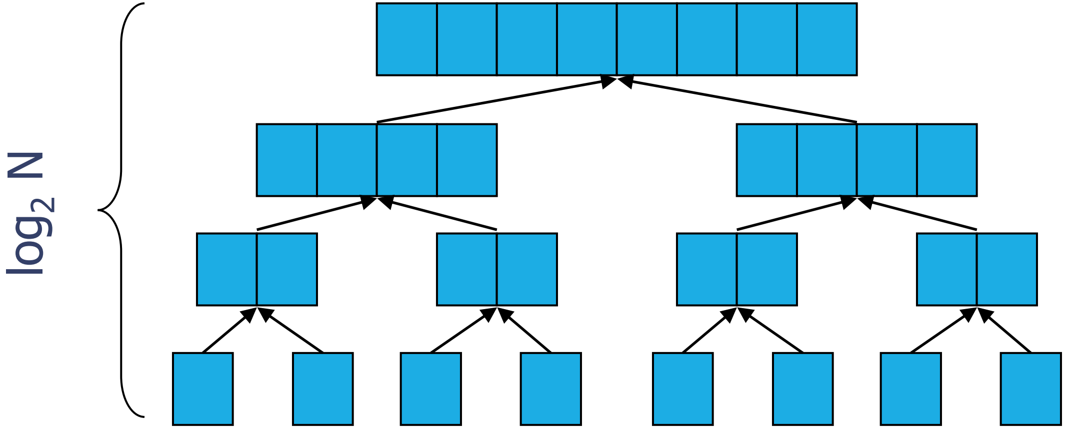
the algorithm becomes unstable!

Complexity Analysis of Merge sort

Informal analysis under the assumption **N** = 2^k

- At each level **N** operations are globally executed by the calls to Merge
- Initially **sorted subarrays** have size **1**
- At each level the size of the **sorted subarrays doubles**, thus at the **i-th** step size is $\mathbf{2^i}$
- Termination occurs when 2^i = **N**, thus **i** = $\log_2$**N** levels are needed
- As the cost of each level is **N**, global cost is **N**$\log_2$**N**
- Complexity is linearithmic **T(n)** = O(NlogN)

Complexity Analysis of Merge sort



Levels: $\log_2 N$

Operations at each level: N

Total # of operations: $N \log_2 N$

Generalization to any N $(\neq 2^k)$

The **rightmost array** boundary during Merge is the smaller value between **r** and the value we would have if size were a power of 2 (i + m + m -1)

```
void BottomUpMergeSort(int A[], int B[], int N)
  {
  int i, q, m, l=0, r=N-1;
for (m = 1; m <= r – l; m = m + m)
     for (i = l; i <= r – m; i += m + m) {
        q = i+m-1;
        Merge(A, B, i, q, min(i+m+m-1,r));
     }
}
```

Example of generalization to any N

12	6	4	5	9	2	3

Example of generalization to any N

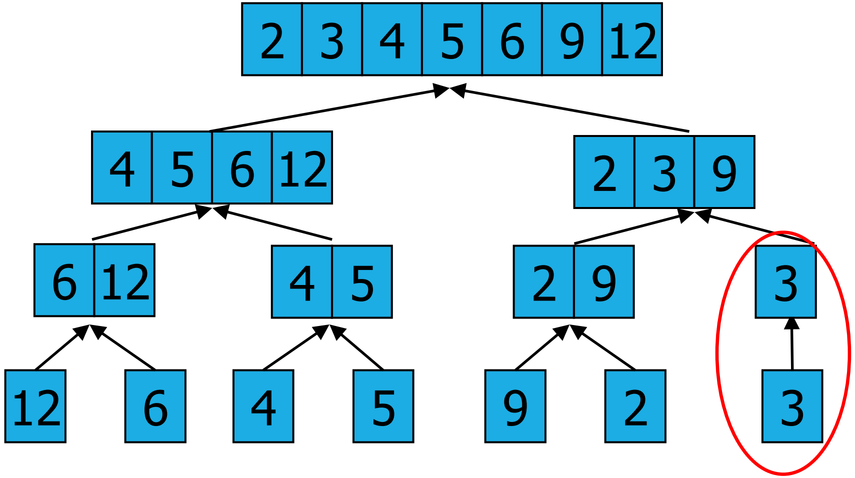
12	6	4	5	9	2	3

N=7

Example of generalization to any N

12	6	4	5	9	2	3

N=7



Example of generalization to any N

12	6	4	5	9	2	3

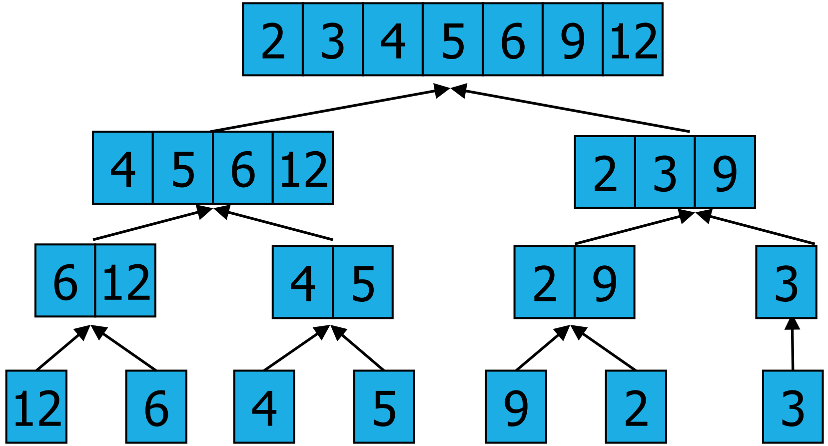
N=7

m=8

m=4

m=2

m=1



Example of generalization to any N

12	6	4	5	9	2	3

N=7