Divide and Conquer

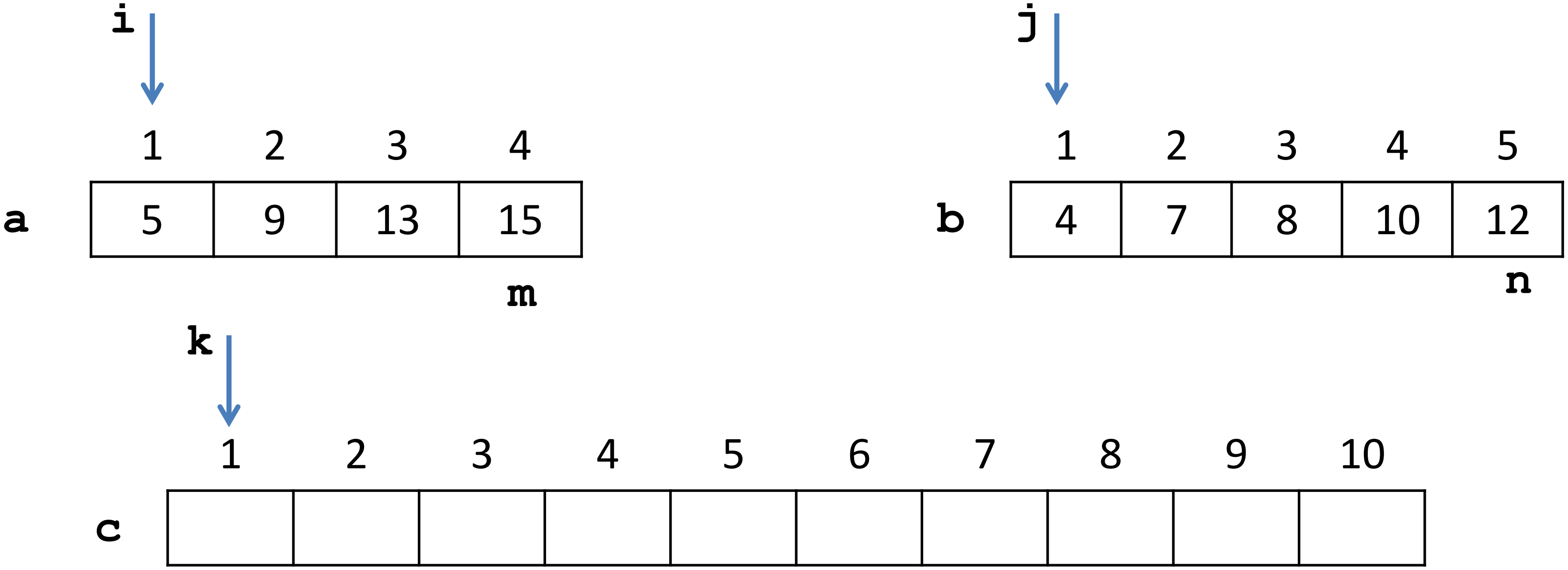
Merge Sort

Divide and Conquer

- Divide and Conquer strategy computes the solution of the problem by computing the solution to the sub-problems and unifying the solutions of the sub-problems as the solution of the given problem.
- Solution of the sub-problem is computed in a similar way till the base case is reached.
- Merge Sort is an example

Merging

- Combining two sorted lists into one sorted list is called as merging.
- Initialize an index variable to the index of the first element of each of the list.
- Initialize an index variable to the index of the first element of the initially empty merged list.

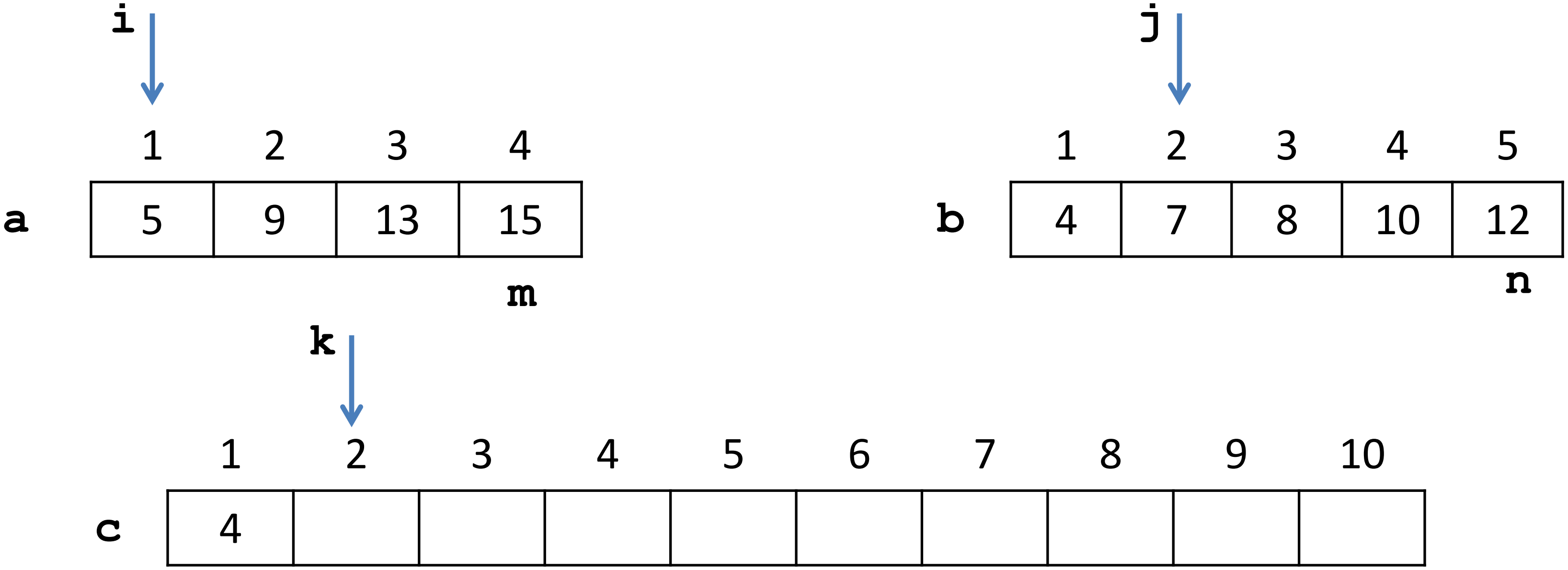


	1	2	3	4
a	5	9	13	15

	1	2	3	4	5
b	4	7	8	10	12

	1	2	3	4	5	6	7	8	9	10
c										

1. **Compare a[i] and b[j] and the smallest is copied into the current index location k of the merged list.**
2. **Increment the source and destination indices of the copy operation**
3. **c[k]:=a[i];i:=i+1 or c[k]:=b[j];j:=j+1;**
4. **k:=k+1;**

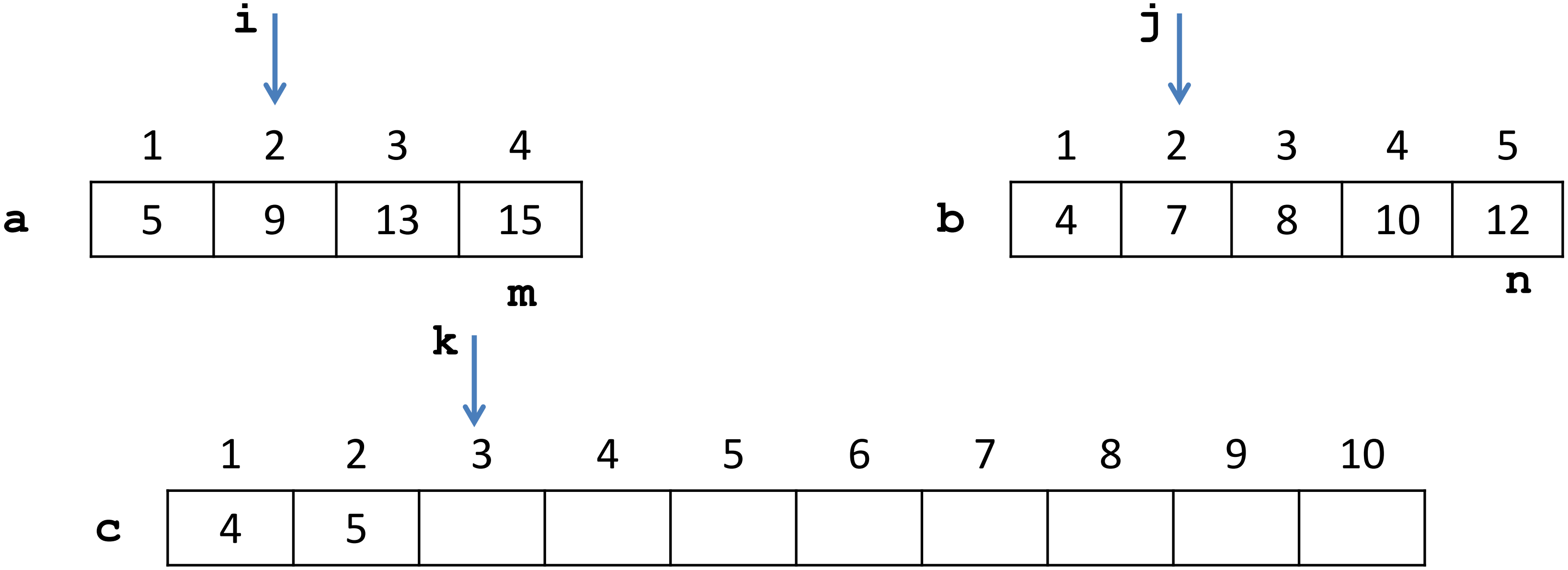


	1	2	3	4
a	5	9	13	15

	1	2	3	4	5
b	4	7	8	10	12

	1	2	3	4	5	6	7	8	9	10
c	4									

1. **Compare a[i] and b[j] and the smallest is copied into the current index location k of the merged list.**
2. **Increment the source and destination indices of the copy operation**
3. **c[k]:=a[i];i:=i+1 or c[k]:=b[j];j:=j=1;**
4. **k:=k+1;**



i

	1	2	3	4
a	5	9	13	15

m

j

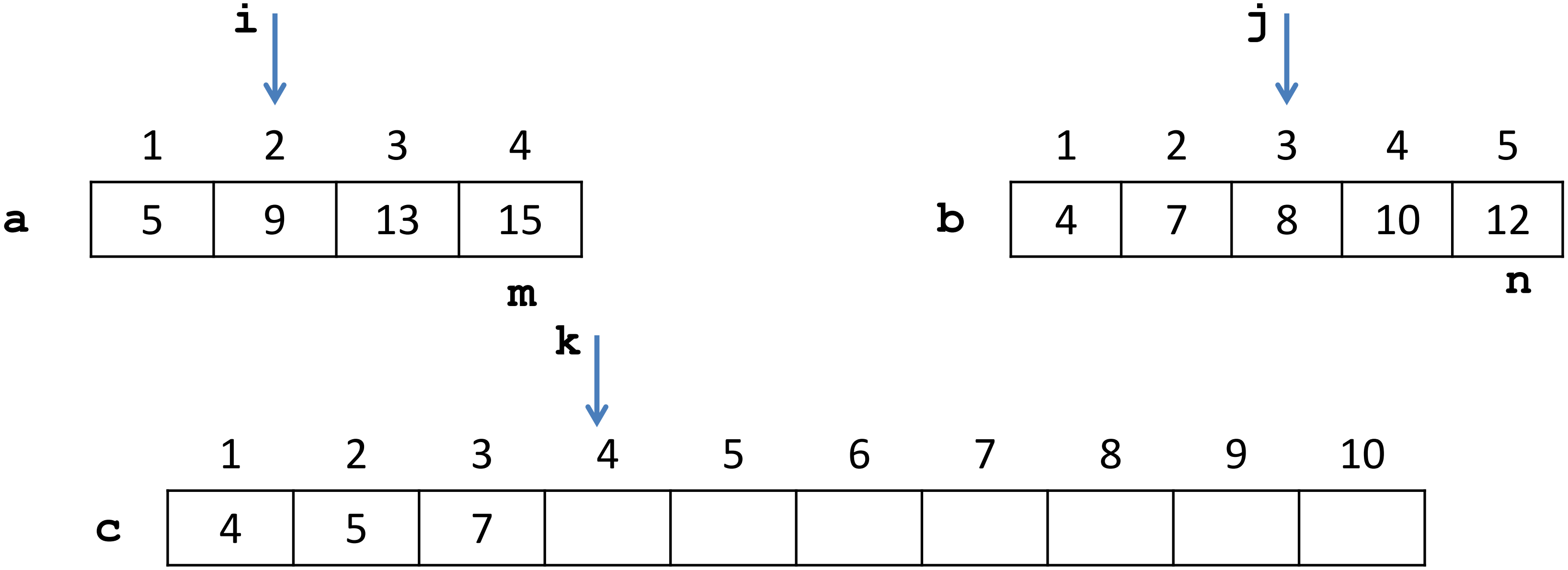
	1	2	3	4	5
b	4	7	8	10	12

n

k

	1	2	3	4	5	6	7	8	9	10
c	4	5								

1. **Compare a[i] and b[j] and the smallest is copied into the current index location k of the merged list.**
2. **Increment the source and destination indices of the copy operation**
3. **c[k]:=a[i];i:=i+1 or c[k]:=b[j];j:=j=1;**
4. **k:=k+1;**

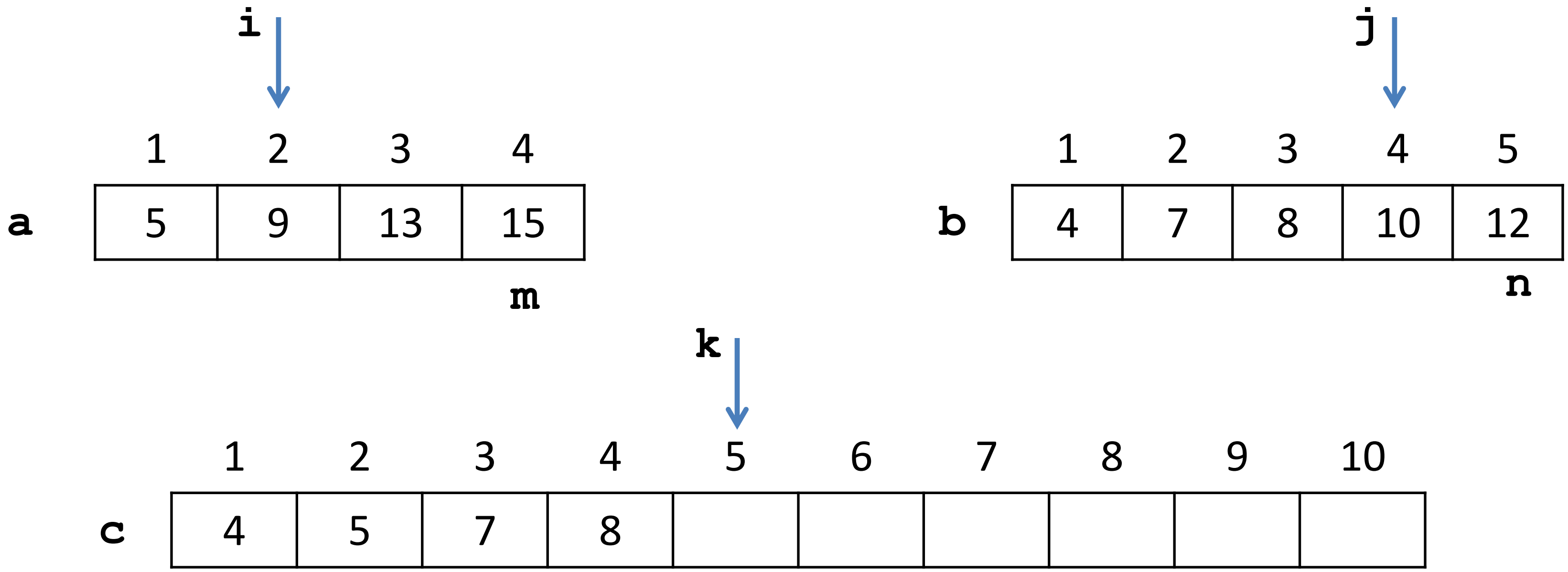


	1	2	3	4
a	5	9	13	15

	1	2	3	4	5
b	4	7	8	10	12

	1	2	3	4	5	6	7	8	9	10
c	4	5	7							

1. **Compare a[i] and b[j] and the smallest is copied into the current index location k of the merged list.**
2. **Increment the source and destination indices of the copy operation**
3. **c[k]:=a[i];i:=i+1 or c[k]:=b[j];j:=j=1;**
4. **k:=k+1;**

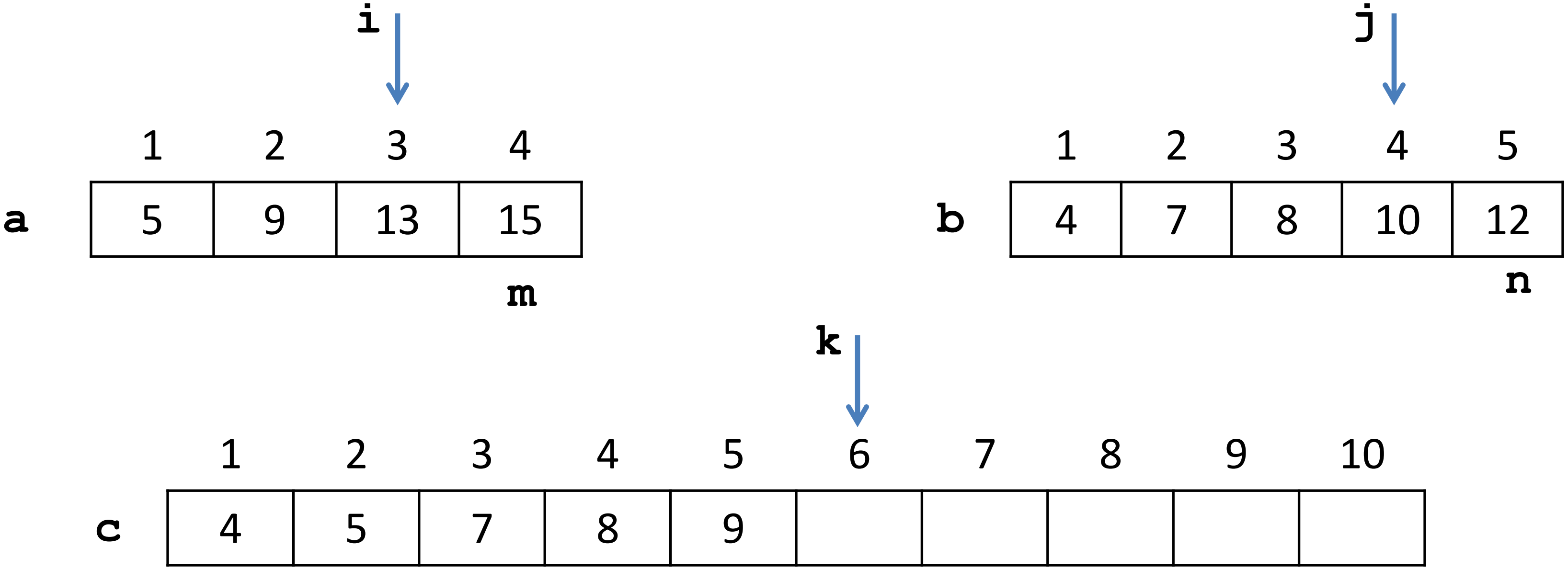


	1	2	3	4
a	5	9	13	15

	1	2	3	4	5
b	4	7	8	10	12

	1	2	3	4	5	6	7	8	9	10
c	4	5	7	8						

1. Compare a[i] and b[j] and the smallest is copied into the current index location k of the merged list.
2. Increment the source and destination indices of the copy operation
3. c[k]:=a[i];i:=i+1 or c[k]:=b[j];j:=j=1;
4. k:=k+1;

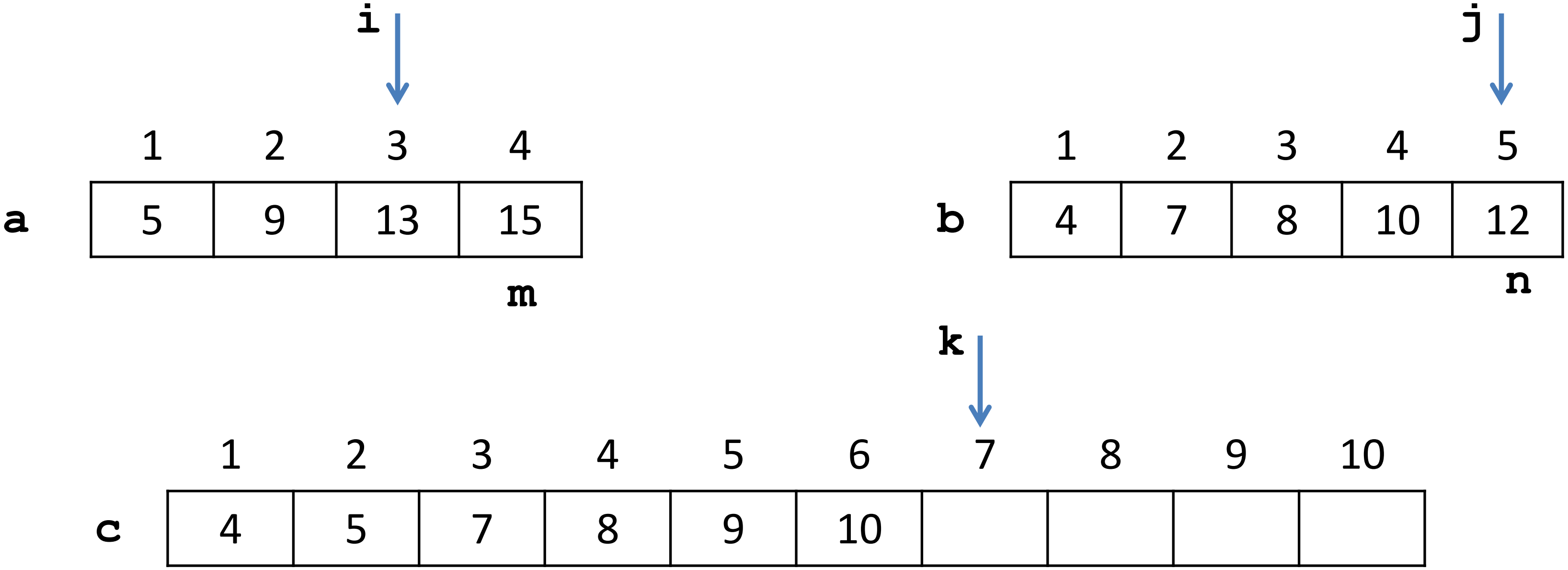


a	1	2	3	4
	5	9	13	15

b	1	2	3	4	5
	4	7	8	10	12

c	1	2	3	4	5	6	7	8	9	10
	4	5	7	8	9					

1. **Compare a[i] and b[j] and the smallest is copied into the current index location k of the merged list.**
2. **Increment the source and destination indices of the copy operation**
3. **c[k]:=a[i];i:=i+1 or c[k]:=b[j];j:=j=1;**
4. **k:=k+1;**

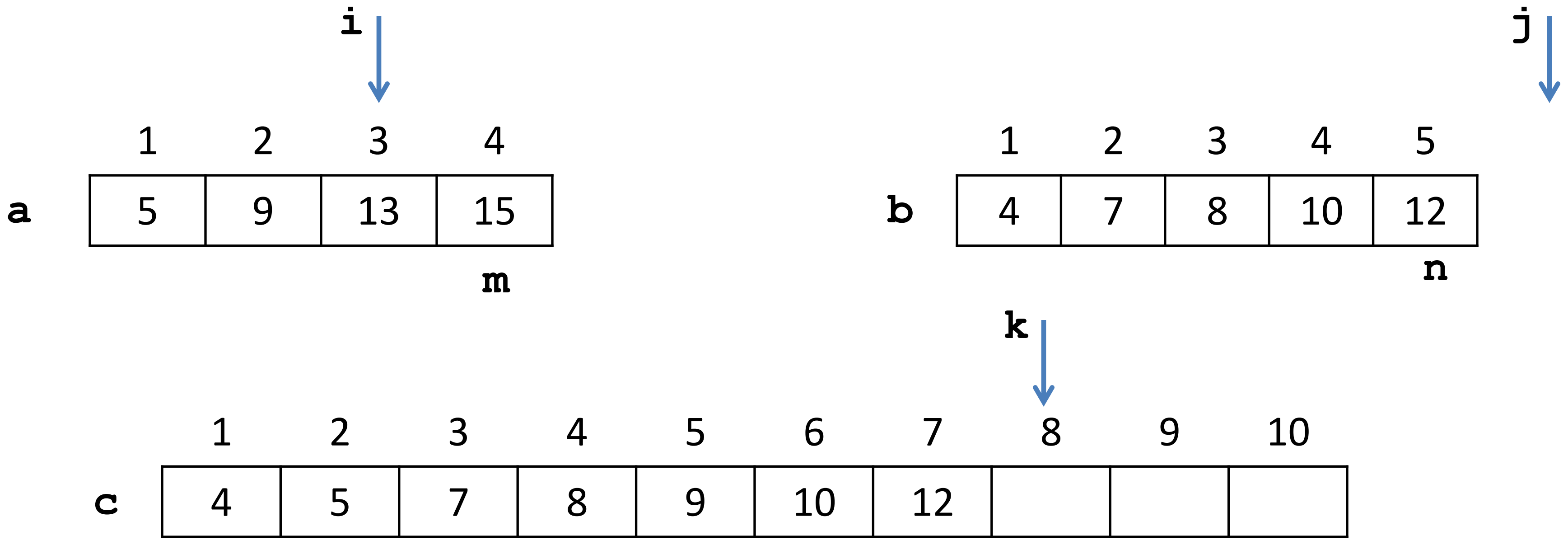


	1	2	3	4
a	5	9	13	15

	1	2	3	4	5
b	4	7	8	10	12

	1	2	3	4	5	6	7	8	9	10
c	4	5	7	8	9	10				

1. **Compare a[i] and b[j] and the smallest is copied into the current index location k of the merged list.**
2. **Increment the source and destination indices of the copy operation**
3. **c[k]:=a[i];i:=i+1 or c[k]:=b[j];j:=j=1;**
4. **k:=k+1;**

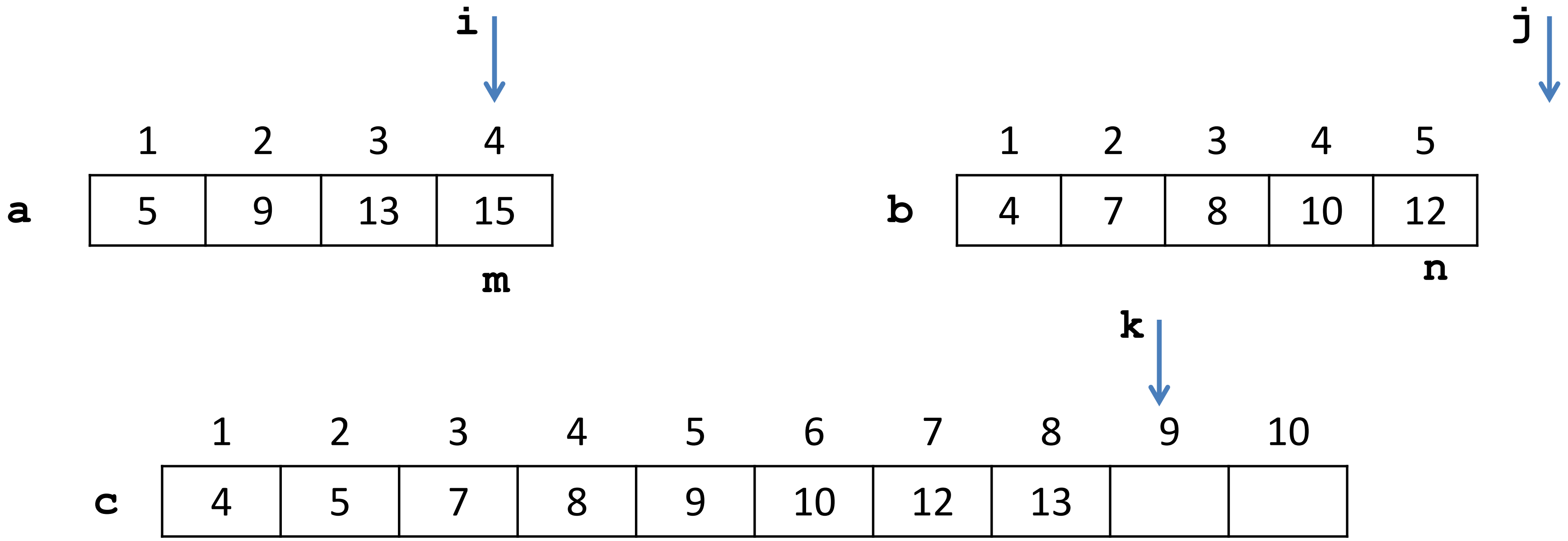


	1	2	3	4
a	5	9	13	15

	1	2	3	4	5
b	4	7	8	10	12

	1	2	3	4	5	6	7	8	9	10
c	4	5	7	8	9	10	12			

1. When we reach end of one of the lists, elements are copied from the other list, till the end of the source list is reached.

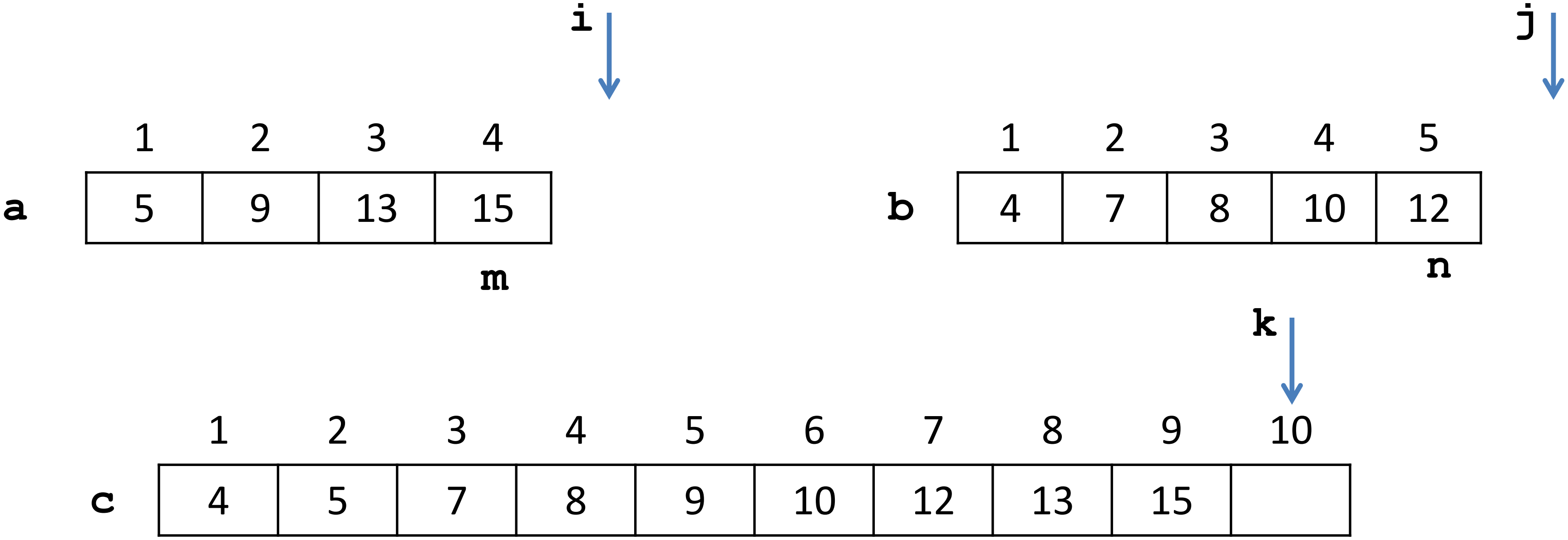


	1	2	3	4
a	5	9	13	15

	1	2	3	4	5
b	4	7	8	10	12

	1	2	3	4	5	6	7	8	9	10
c	4	5	7	8	9	10	12	13		

1. **When we reach end of one of the lists, elements are copied from the other list, till the end of the source list is reached.**
2. **Copy the elements from i to m or j to n**



	1	2	3	4
a	5	9	13	15

	1	2	3	4	5
b	4	7	8	10	12

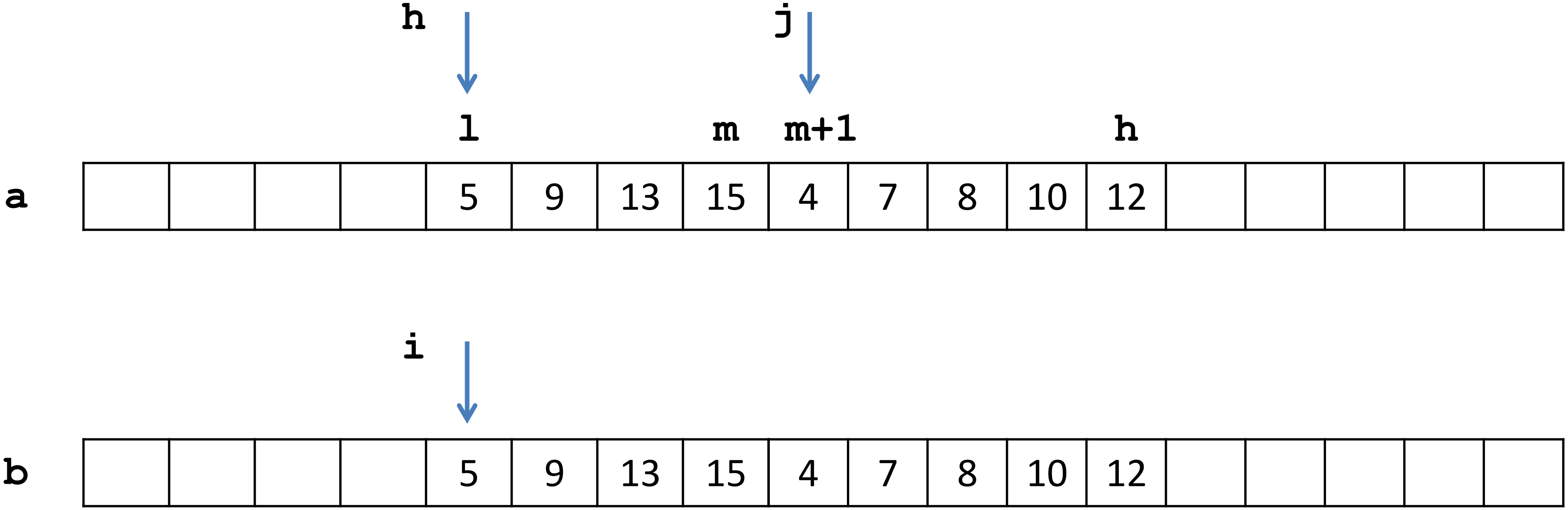
	1	2	3	4	5	6	7	8	9	10
c	4	5	7	8	9	10	12	13	15	

1. When we reach end of one of the lists, elements are copied from the other list, till the end of the source list is reached.
2. Copy the elements from i to m or j to n

```
i:=1;j:=1;k:=1;
while(i<=m and j<=n) do
{
   if(a[i]<=b[j])then
   {
      c[k]:=a[i];i:=i+1;
   }
   else
   {
      c[k]:=b[j];j:=j+1;
   }
   k:=k+1;
}
for p:= i to m  do
{
   c[k]:=a[p];k:=k+1;
}
for p:= j to n  do
{
   c[k]:=b[p];j:=j+1;
}
```

Merging in Merge Sort

- Merging in Merge Sort has a small variation.
- The two lists to be merged are logical partitions in the same array.
- A temporary array is used to store the merged list of the two logical partitions.
- The merged list from the temporary array is copied back.



h j

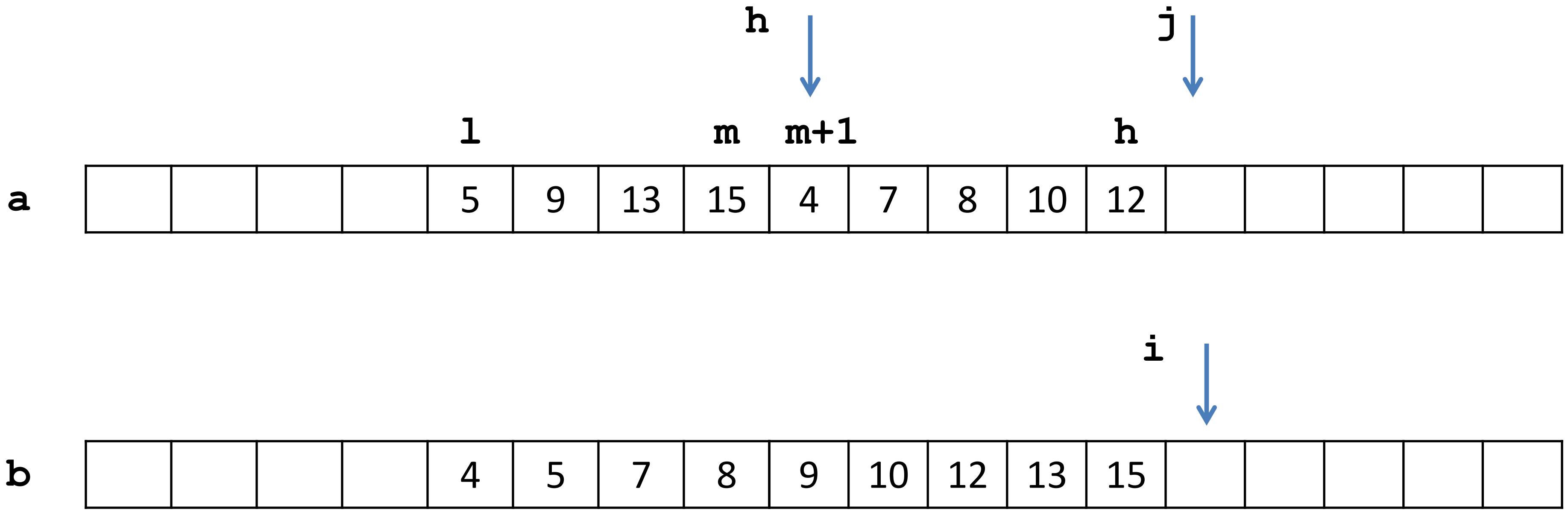
l m m+1 h

a					5	9	13	15	4	7	8	10	12					

i

b					5	9	13	15	4	7	8	10	12					

low(l)
mid(m)
high(h)



a					5	9	13	15	4	7	8	10	12					
b					4	5	7	8	9	10	12	13	15					

					l			**m**	**m+1**				**h**					
a					4	5	7	8	9	10	12	13	15					
b					4	5	7	8	9	10	12	13	15					

```
Algorithm Merge(low,mid,high)
//a[low:mid] and a[mid+1:high] are two sorted logical
//partitions of global array a
//These two partitions are merged to array b[low:high]
//The elements are copied from b[low:high] to a[low:high]
{
    h:=low;i:=low;j:=mid+1;
    while(h<=mid and j<=high) do
    {
        if(a[h]<=a[j])then
        {
            b[i]:=a[h];h:=h+1;
        }
        else
        {
            b[i]:=a[j];j:=j+1;
        }
        i:=i+1;
    }
```

```
   for k:= h to mid  do
   {
      b[i]:=a[k];i:=i+1;
   }
   for k:= j to high  do
   {
      b[i]:=a[k];i:=i+1;
   }
   for k:= low to high  do
      a[k]:=b[k];
   }

}
```

Merge Sort

- Given a sequence of **`n`** elements **`a[1:n]`**, the objective is to sort them in non-decreasing order.
- The recursive formulation of Merge Sort is **`MergeSort(low,high)`**:

Merge Sort

- **`MergeSort(low,high)`**:
- A list with one element is Merge Sorted.(base case)

  `if (low<high)then:`

  – Divide the list into two logical partitions.

  – **`mid=(low+high)/2`**

  – Merge Sort each partition.

  – **`MergeSort(low,mid)`**

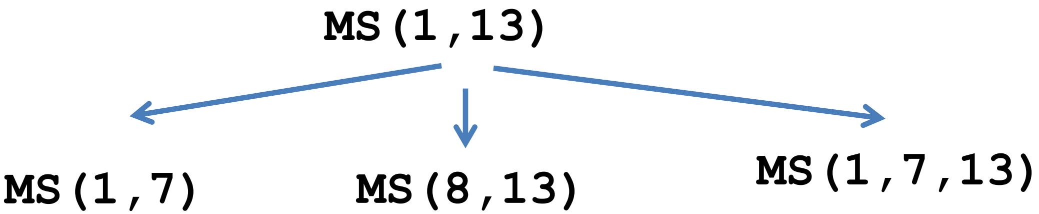
  – **`MergeSort(mid+1,high)`**

  – Merge two sorted partitions into one.

  – **`Merge(low,mid,high)`**

low ↓ ↓ high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	15	7	12	13	4	8	10	11	9	16	3	6	0
b													

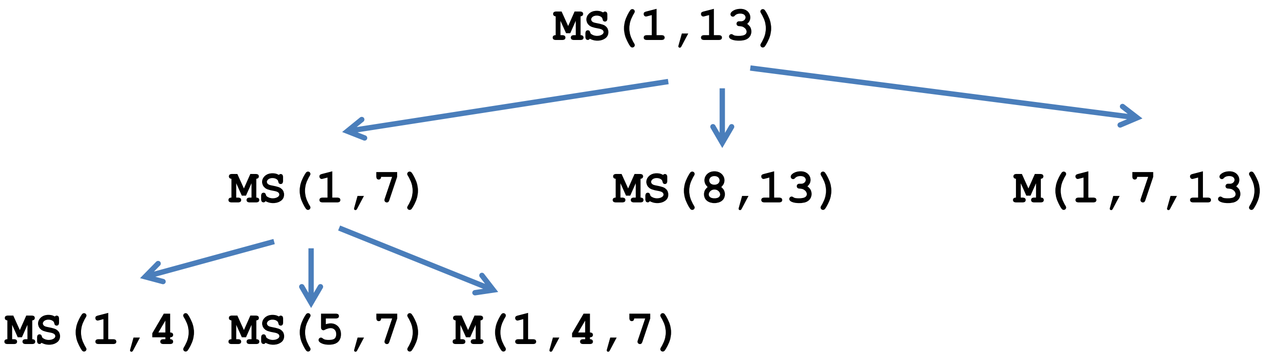


MS(1,13)

MS(1,7) **MS(8,13)** **MS(1,7,13)**

low ↓ ↓**high**

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	15	7	12	13	4	8	10	11	9	16	3	6	0
b													



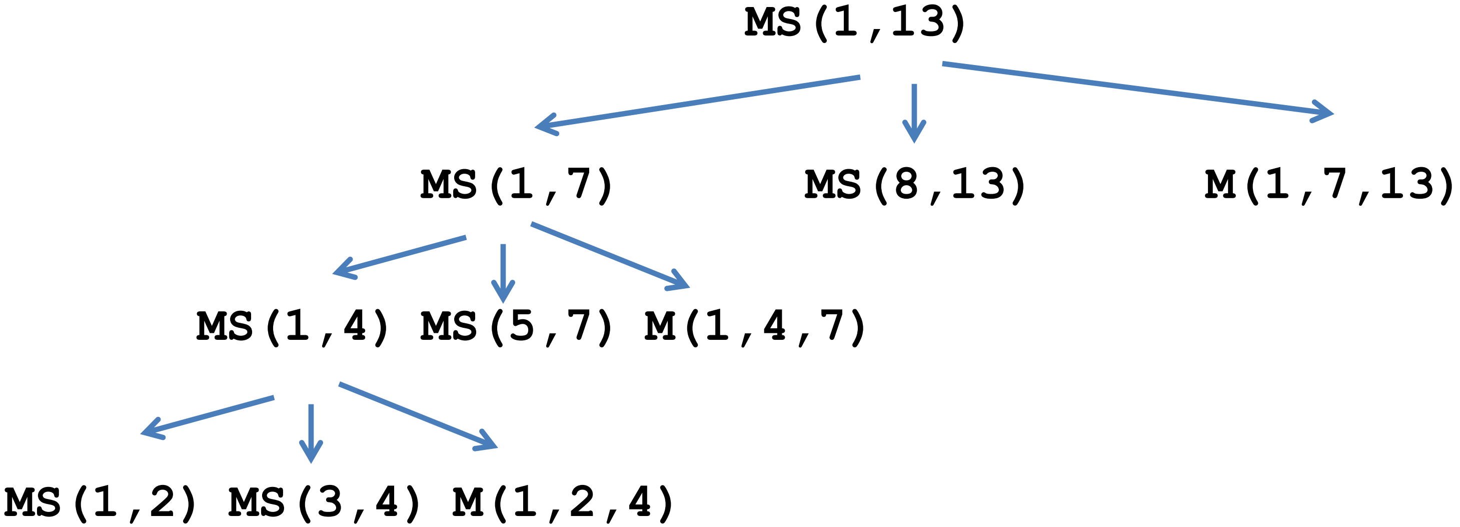
MS(1,13)

MS(1,7) MS(8,13) M(1,7,13)

MS(1,4) MS(5,7) M(1,4,7)

low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	15	7	12	13	4	8	10	11	9	16	3	6	0
b													



MS(1,13)

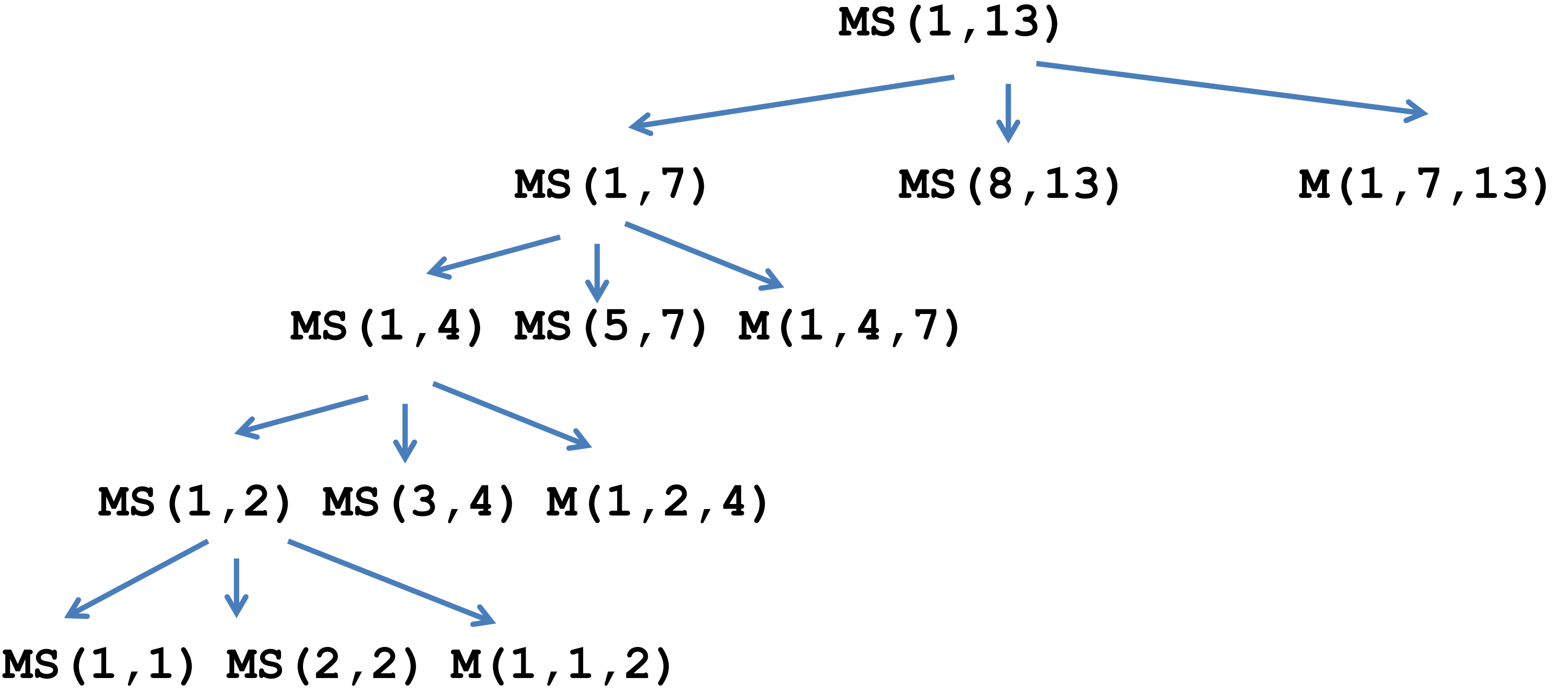
MS(1,7) MS(8,13) M(1,7,13)

MS(1,4) MS(5,7) M(1,4,7)

MS(1,2) MS(3,4) M(1,2,4)

low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	15	7	12	13	4	8	10	11	9	16	3	6	0
b													



MS(1,13)

MS(1,7) MS(8,13) M(1,7,13)

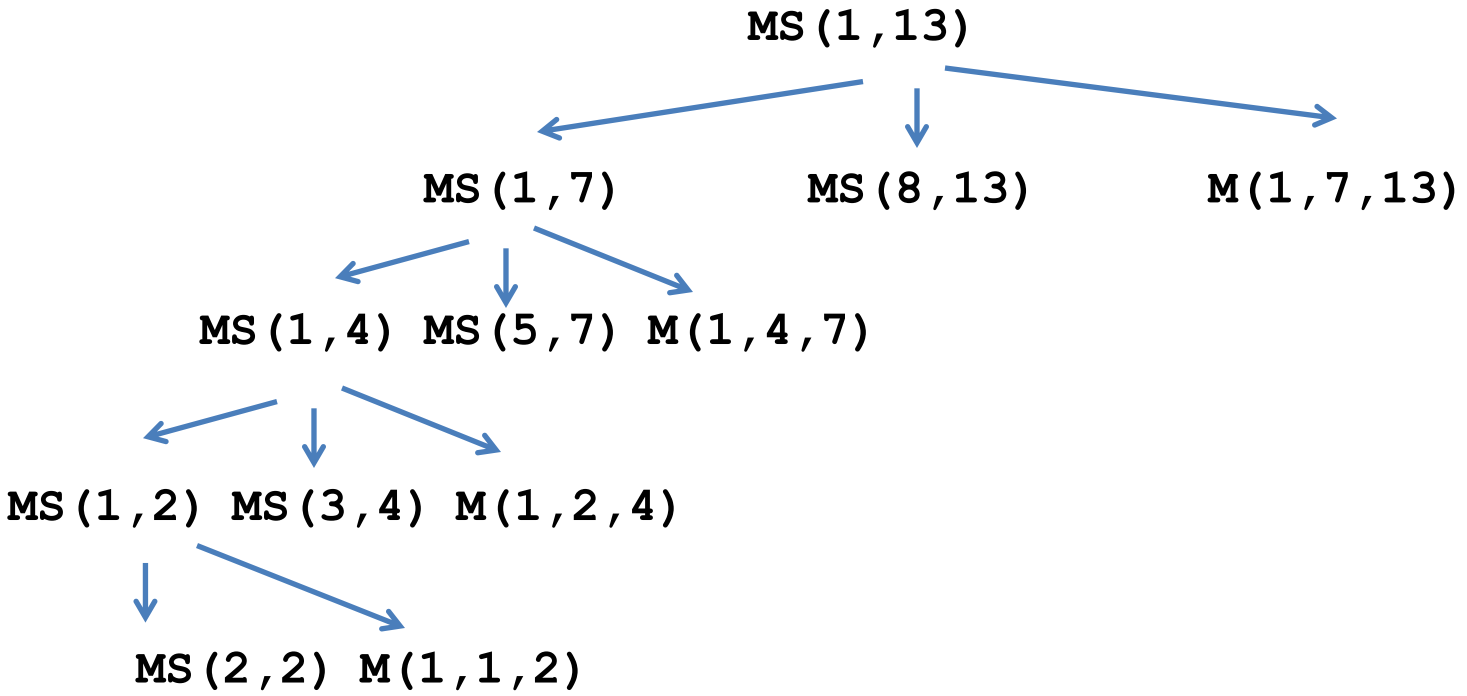
MS(1,4) MS(5,7) M(1,4,7)

MS(1,2) MS(3,4) M(1,2,4)

MS(1,1) MS(2,2) M(1,1,2)

low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	15	7	12	13	4	8	10	11	9	16	3	6	0
b													



MS(1,13)

MS(1,7) MS(8,13) M(1,7,13)

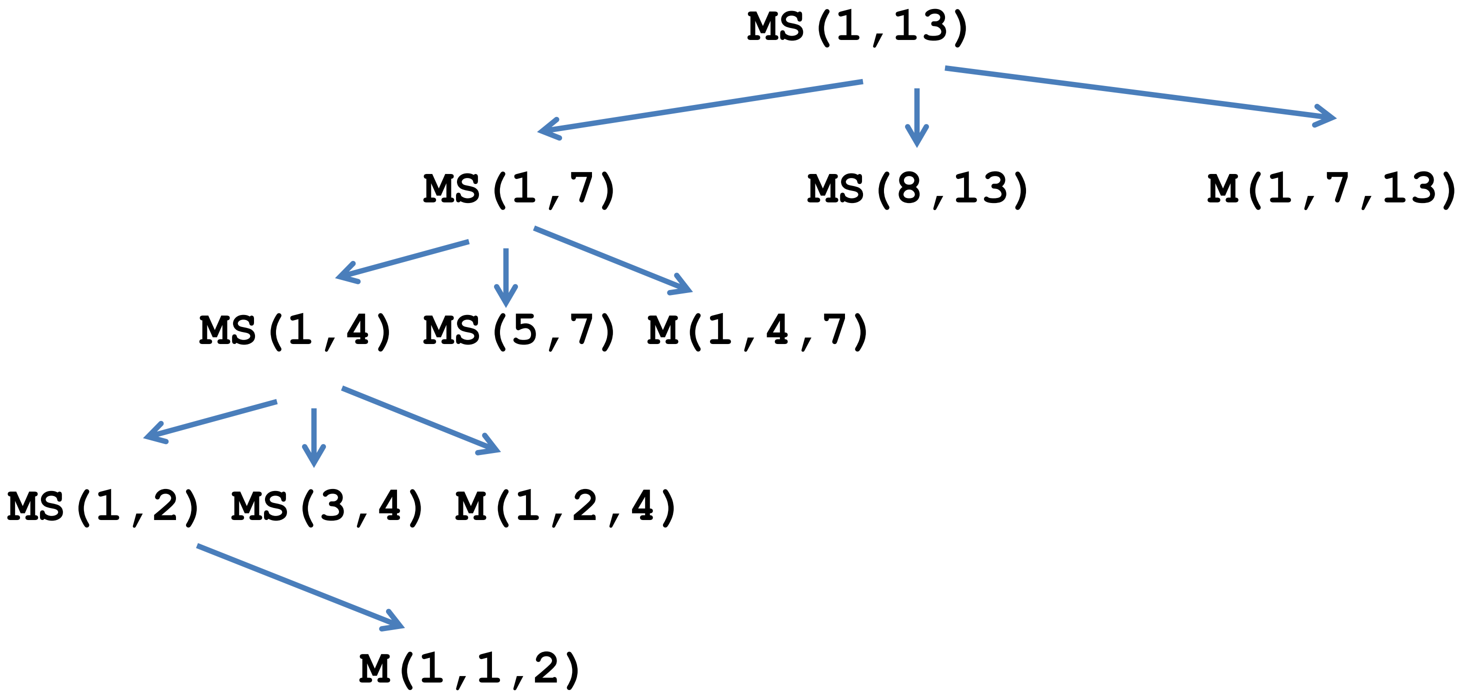
MS(1,4) MS(5,7) M(1,4,7)

MS(1,2) MS(3,4) M(1,2,4)

MS(2,2) M(1,1,2)

low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	15	7	12	13	4	8	10	11	9	16	3	6	0
b													


MS(1,13)

MS(1,7) MS(8,13) M(1,7,13)

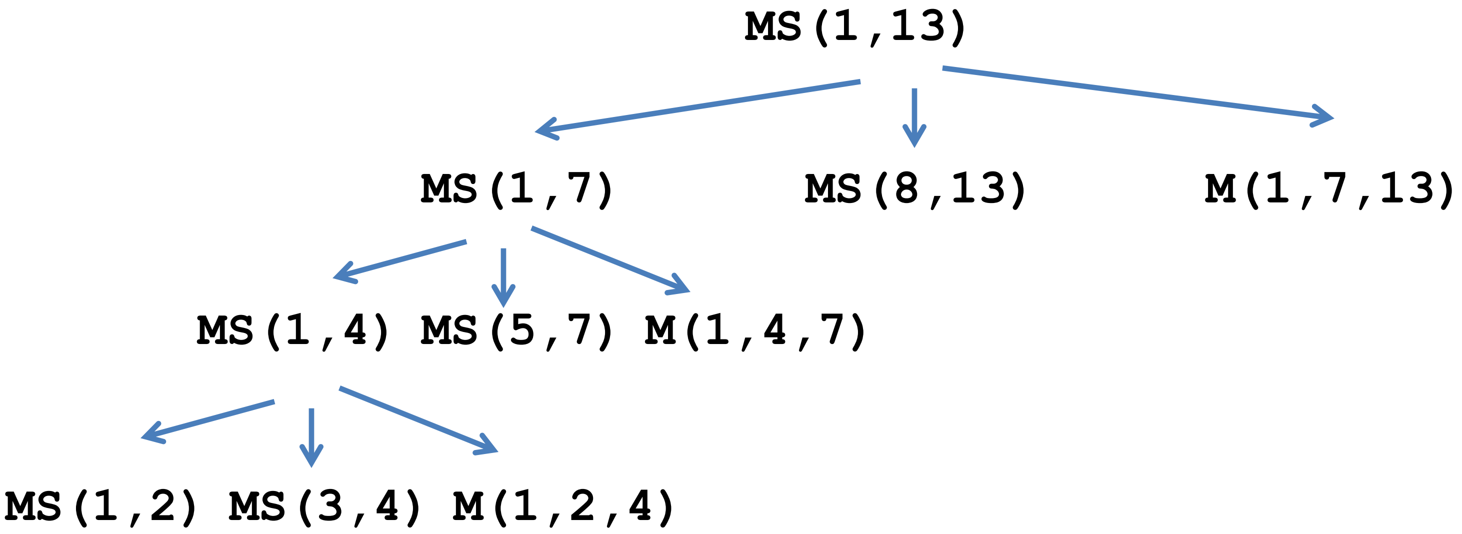
MS(1,4) MS(5,7) M(1,4,7)

MS(1,2) MS(3,4) M(1,2,4)

M(1,1,2)

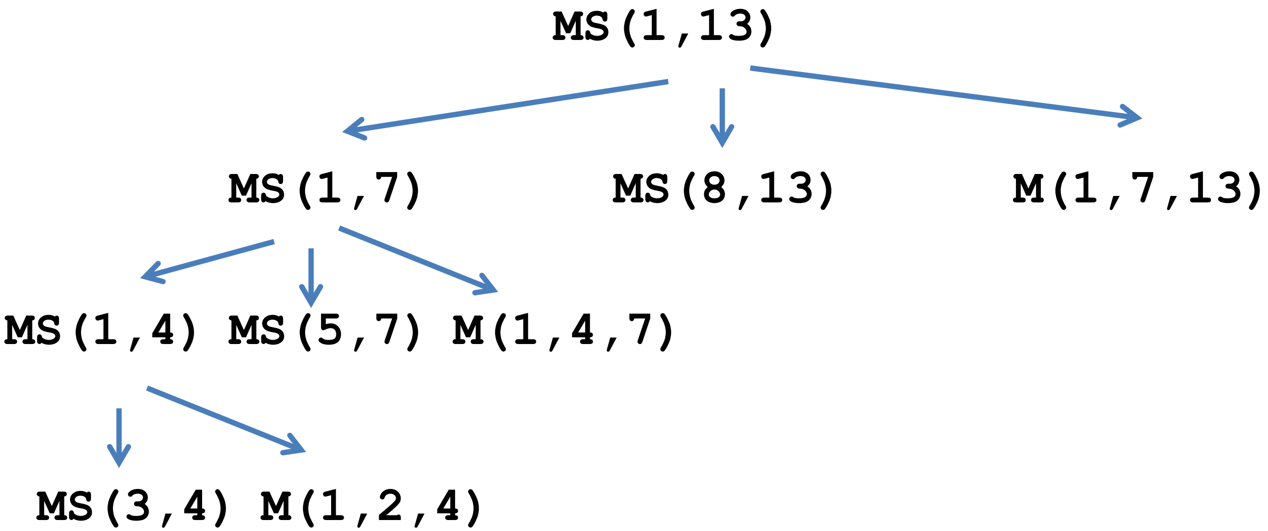
low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	15	7	12	13	4	8	10	11	9	16	3	6	0
b	7	15											



low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	7	15	12	13	4	8	10	11	9	16	3	6	0
b													



MS(1,13)

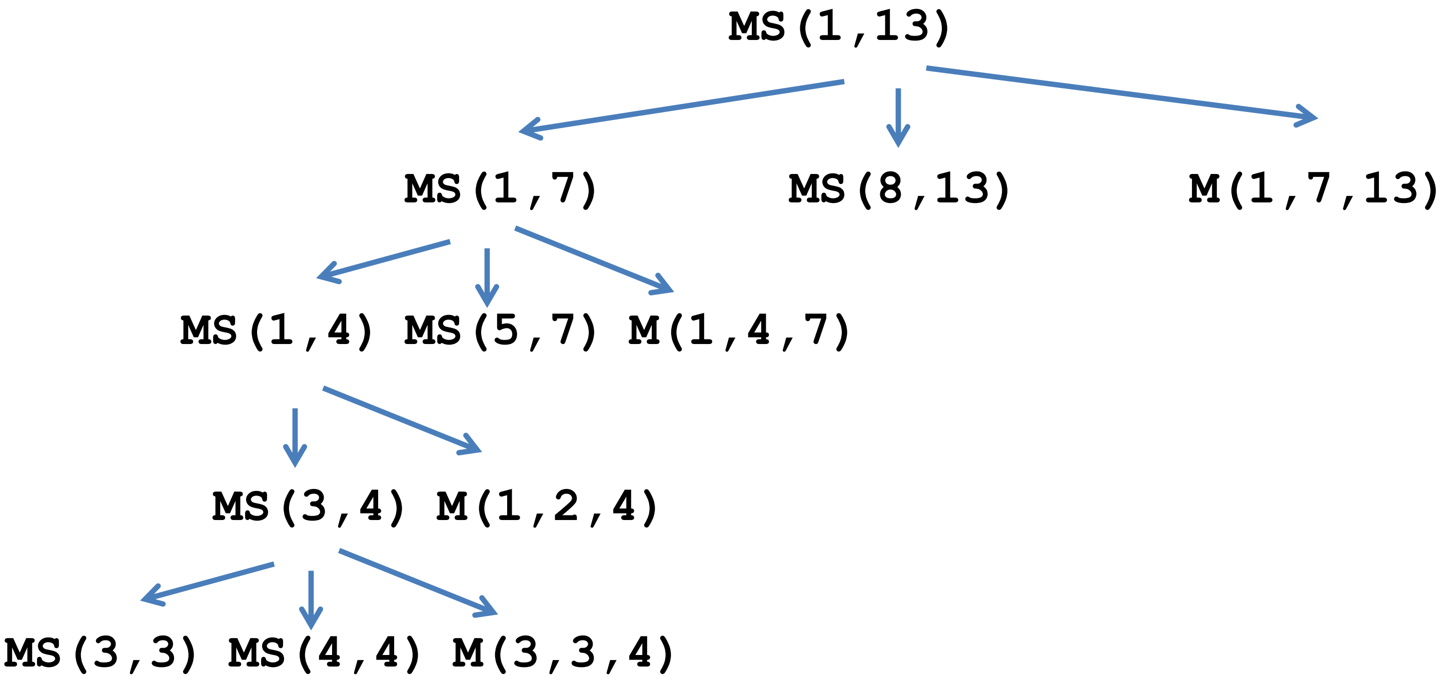
MS(1,7) MS(8,13) M(1,7,13)

MS(1,4) MS(5,7) M(1,4,7)

MS(3,4) M(1,2,4)

low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	7	15	12	13	4	8	10	11	9	16	3	6	0
b													



MS(1,13)

MS(1,7) **MS(8,13)** **M(1,7,13)**

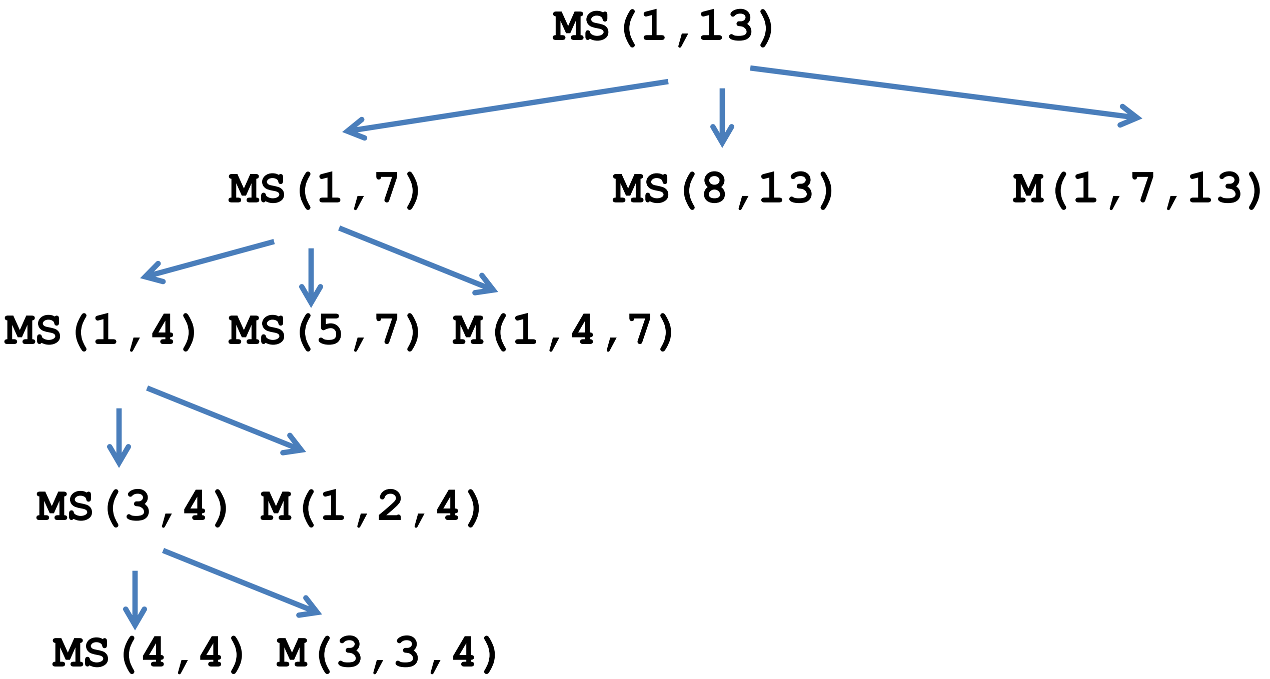
MS(1,4) **MS(5,7)** **M(1,4,7)**

MS(3,4) **M(1,2,4)**

MS(3,3) **MS(4,4)** **M(3,3,4)**

low ⇓ high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	7	15	12	13	4	8	10	11	9	16	3	6	0
b													



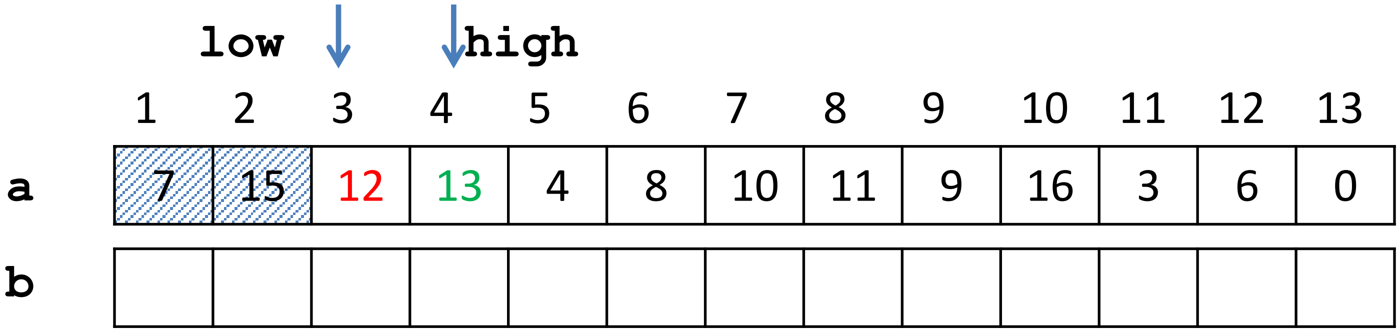
MS(1,13)

MS(1,7) **MS(8,13)** **M(1,7,13)**

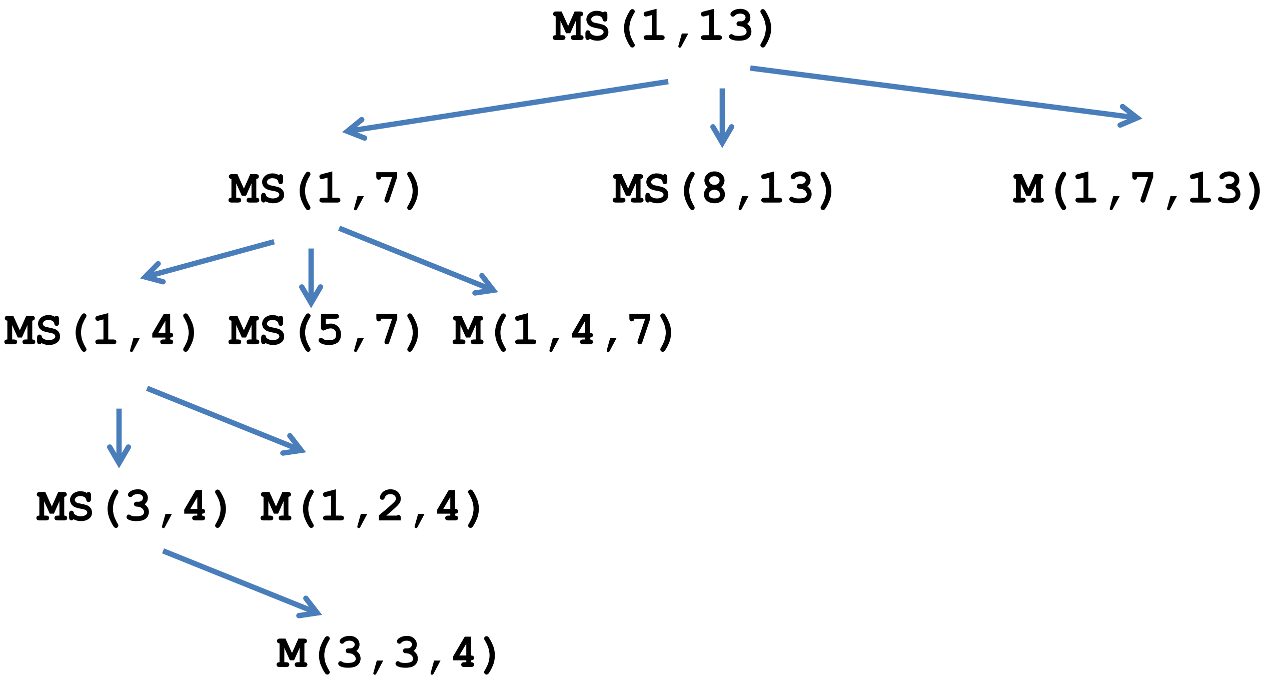
MS(1,4) **MS(5,7)** **M(1,4,7)**

MS(3,4) **M(1,2,4)**

MS(4,4) **M(3,3,4)**



	1	2	3	4	5	6	7	8	9	10	11	12	13
a	7	15	12	13	4	8	10	11	9	16	3	6	0
b													



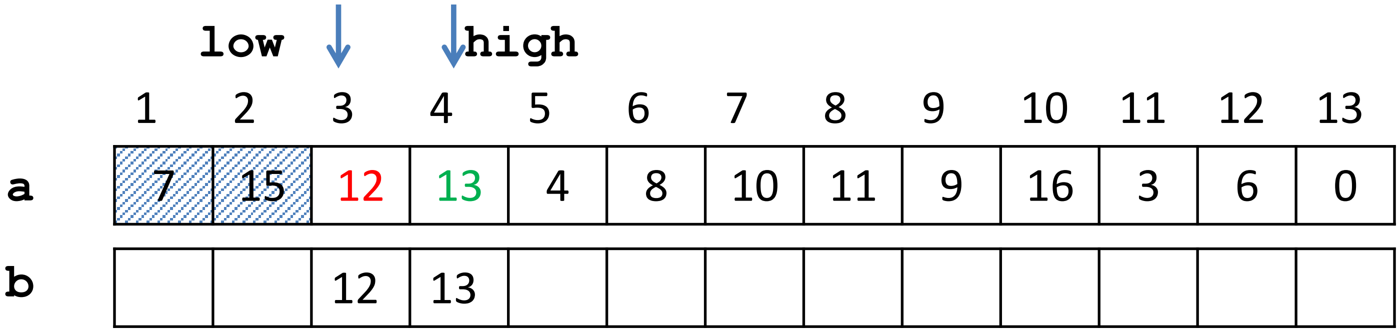
MS(1,13)

MS(1,7) MS(8,13) M(1,7,13)

MS(1,4) MS(5,7) M(1,4,7)

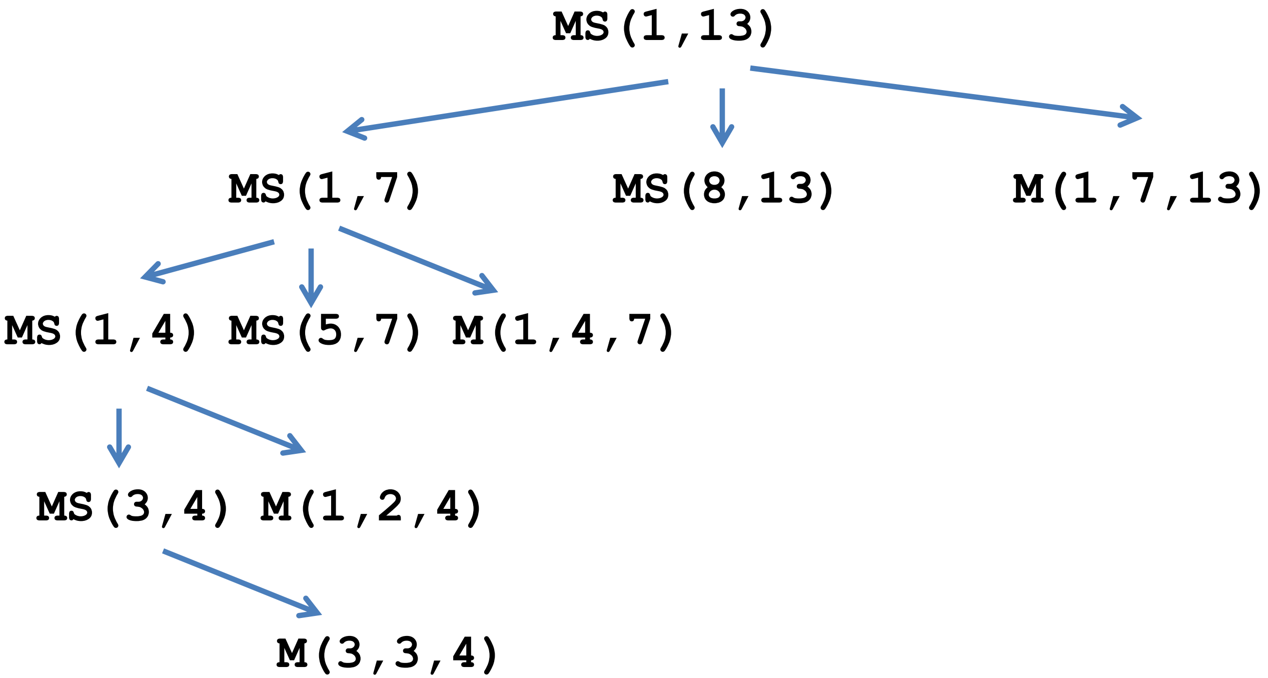
MS(3,4) M(1,2,4)

M(3,3,4)



low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	7	15	12	13	4	8	10	11	9	16	3	6	0
b			12	13									



MS(1,13)

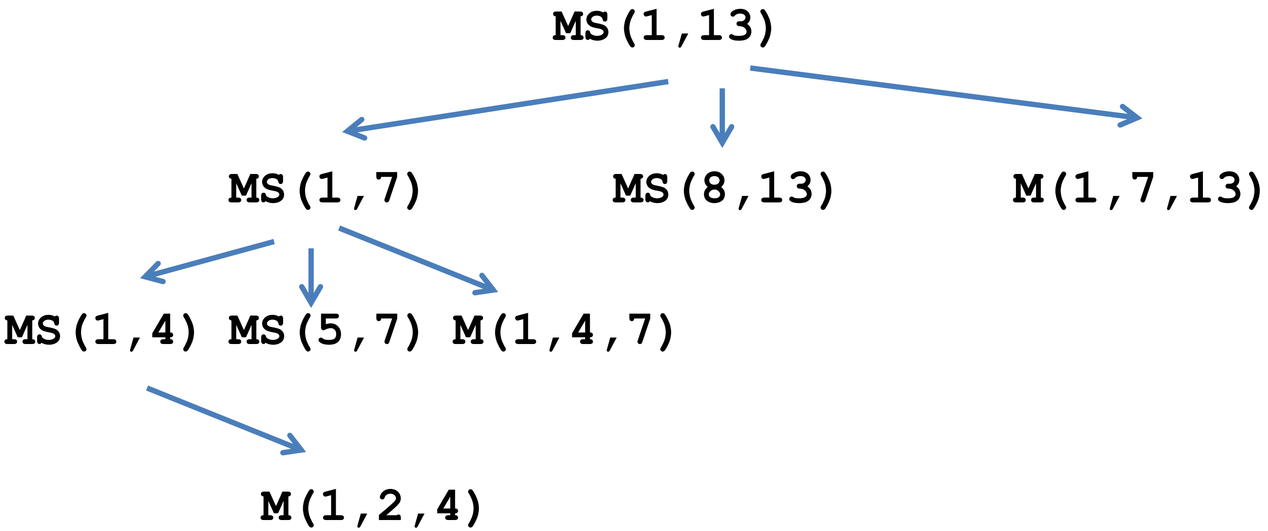
MS(1,7) MS(8,13) M(1,7,13)

MS(1,4) MS(5,7) M(1,4,7)

MS(3,4) M(1,2,4)

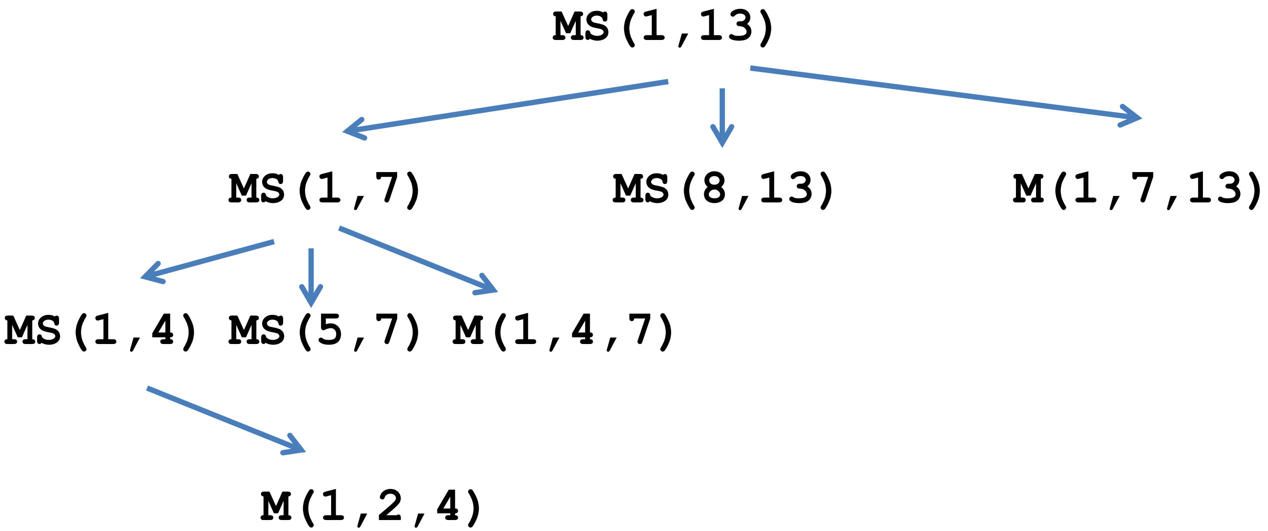
M(3,3,4)

			low ↓	↓high									
	1	2	3	4	5	6	7	8	9	10	11	12	13
a	▨ 7	▨ 15	12	13	4	8	10	11	9	16	3	6	0
b													



low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	7	15	12	13	4	8	10	11	9	16	3	6	0
b													



MS(1,13)

MS(1,7)

MS(8,13)

M(1,7,13)

MS(1,4)

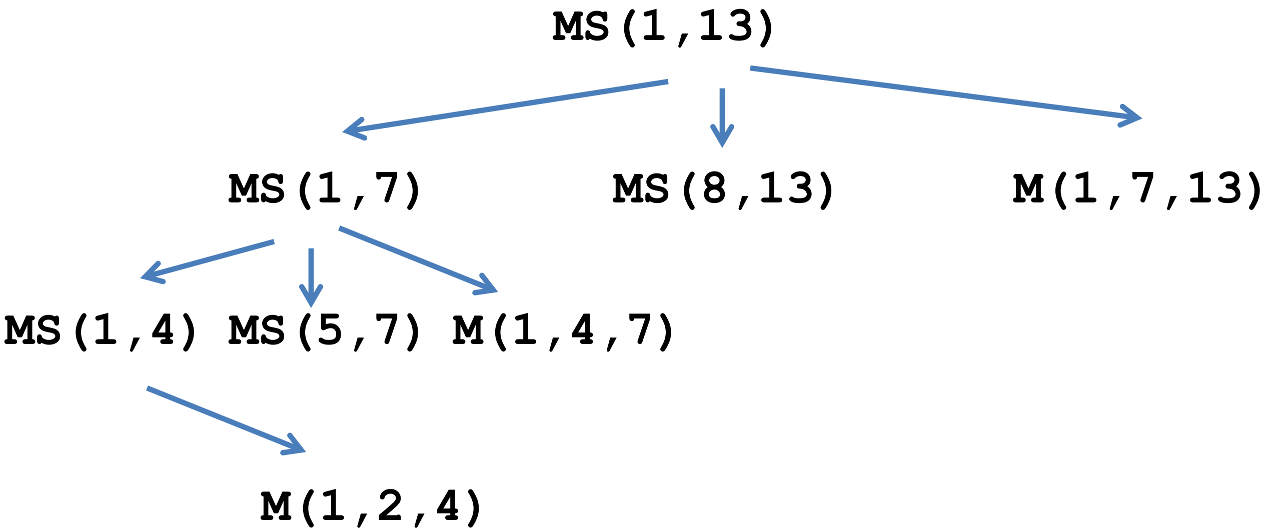
MS(5,7)

M(1,4,7)

M(1,2,4)

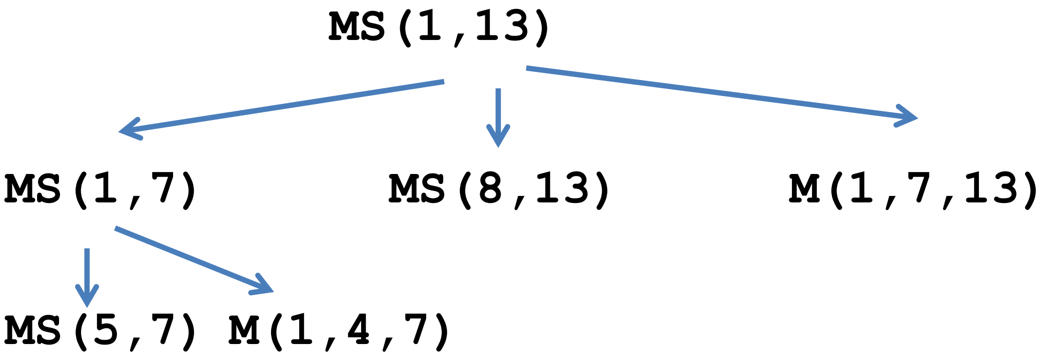
low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	7	15	12	13	4	8	10	11	9	16	3	6	0
b	7	12	13	15									



low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	7	12	13	15	4	8	10	11	9	16	3	6	0
b													



MS(1,13)

MS(1,7)

MS(8,13)

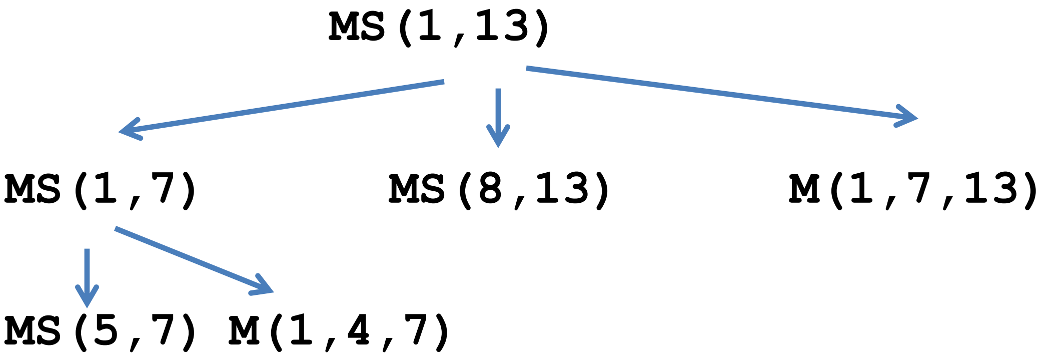
M(1,7,13)

MS(5,7)

M(1,4,7)

low ↓ ↓high

	1	2	3	4	5	6	7	8	9	10	11	12	13
a	7	12	13	15	4	8	10	11	9	16	3	6	0
b													



MS(1,13)

MS(1,7)

MS(8,13)

M(1,7,13)

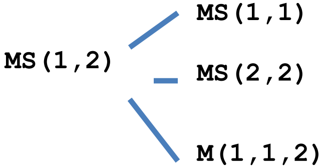
MS(5,7)

M(1,4,7)

	1	2	3	4	5	6	7	8	9	10	11	12	13
	15	7	12	13	4	8	10	11	9	16	3	6	0
MS(1,1)	15												

	1	2	3	4	5	6	7	8	9	10	11	12	13
	15	7	12	13	4	8	10	11	9	16	3	6	0
MS(1,1)	15												
MS(2,2)		7											

	1	2	3	4	5	6	7	8	9	10	11	12	13
	15	7	12	13	4	8	10	11	9	16	3	6	0
`MS(1,1)`	15												
`MS(2,2)`		7											
`M(1,1,2)`	7	15											



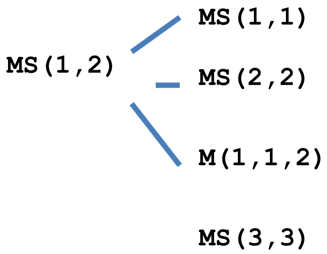
MS(1,2)

MS(1,1)

MS(2,2)

M(1,1,2)

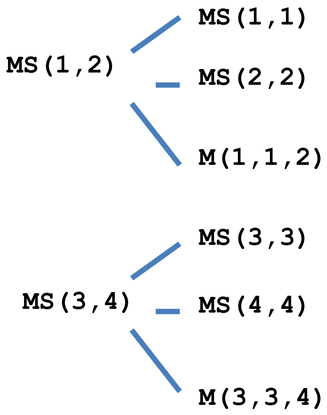
1	2	3	4	5	6	7	8	9	10	11	12	13
15	7	12	13	4	8	10	11	9	16	3	6	0
15												
	7											
7	15											



1	2	3	4	5	6	7	8	9	10	11	12	13
15	7	12	13	4	8	10	11	9	16	3	6	0
15												
	7											
7	15											
		12										

		1	2	3	4	5	6	7	8	9	10	11	12	13
		15	7	12	13	4	8	10	11	9	16	3	6	0
MS(1,2)	MS(1,1)	15												
	MS(2,2)		7											
	M(1,1,2)	7	15											
	MS(3,3)			12										
	MS(4,4)				13									

		1	2	3	4	5	6	7	8	9	10	11	12	13
		15	7	12	13	4	8	10	11	9	16	3	6	0
	MS(1,1)	15												
MS(1,2)	MS(2,2)		7											
	M(1,1,2)	7	15											
	MS(3,3)			12										
	MS(4,4)				13									
	M(3,3,4)			12	13									



MS(1,2)

MS(1,1)

MS(2,2)

M(1,1,2)

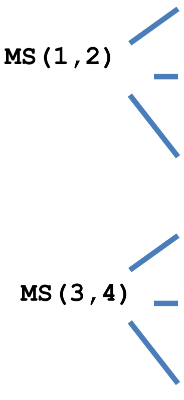
MS(3,4)

MS(3,3)

MS(4,4)

M(3,3,4)

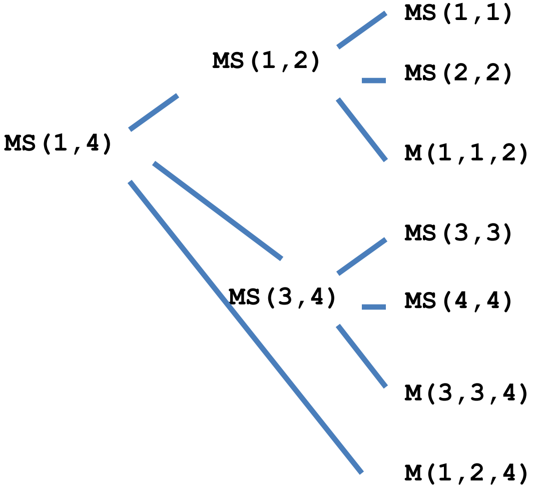
1	2	3	4	5	6	7	8	9	10	11	12	13
15	7	12	13	4	8	10	11	9	16	3	6	0
15												
	7											
7	15											
		12										
			13									
		12	13									



MS(1,2)

MS(3,4)

	1	2	3	4	5	6	7	8	9	10	11	12	13
	15	7	12	13	4	8	10	11	9	16	3	6	0
MS(1,1)	15												
MS(2,2)		7											
M(1,1,2)	7	15											
MS(3,3)			12										
MS(4,4)				13									
M(3,3,4)			12	13									
M(1,2,4)	7	12	13	15									



MS(1,4)

MS(1,2)

MS(1,1)

MS(2,2)

M(1,1,2)

MS(3,4)

MS(3,3)

MS(4,4)

M(3,3,4)

M(1,2,4)

1	2	3	4	5	6	7	8	9	10	11	12	13
15	7	12	13	4	8	10	11	9	16	3	6	0
15												
	7											
7	15											
		12										
			13									
		12	13									
7	12	13	15									

MergeSort

```
Algorithm MergeSort(low,high)
//a[low:high] is a global array to be sorted
//b[low:high] is a temporary array used for merging
{
    if(low<high)then
    {
        mid:=floor(low+high)/2);
        MergeSort(low,mid);
        MergeSort(mid+1,high);
        Merge(low,mid,high);
    }
}
```

MergeSort

- $T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$
- $a = 2, b = 2, k = 1$
- $\log_b a = \mathrm{k\ or\ a} = b^k$
- $T(n) = \theta(nlogn)$

MergeSort

- $T(n) = 2T\left(\frac{n}{2}\right) + n$
- $T(n) = 2(2T\left(\frac{n}{4}\right) + n/2) + n$
- $= 2^2T\left(\frac{n}{2^2}\right) + 2n$
- $= 2^2(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}) + 2n$
- $= 2^3T\left(\frac{n}{2^3}\right) + 3n$
- …
- $= \ 2^kT\left(\frac{\boldsymbol{n}}{\boldsymbol{2^k}}\right) + kn$ $\qquad \frac{\boldsymbol{n}}{\boldsymbol{2^k}} = 1, k = logn$
- $= nT(1) + n \ \log n$
- $= n + n\log n$
- $= \theta(n\log n)$