Algorithms and Data Structures

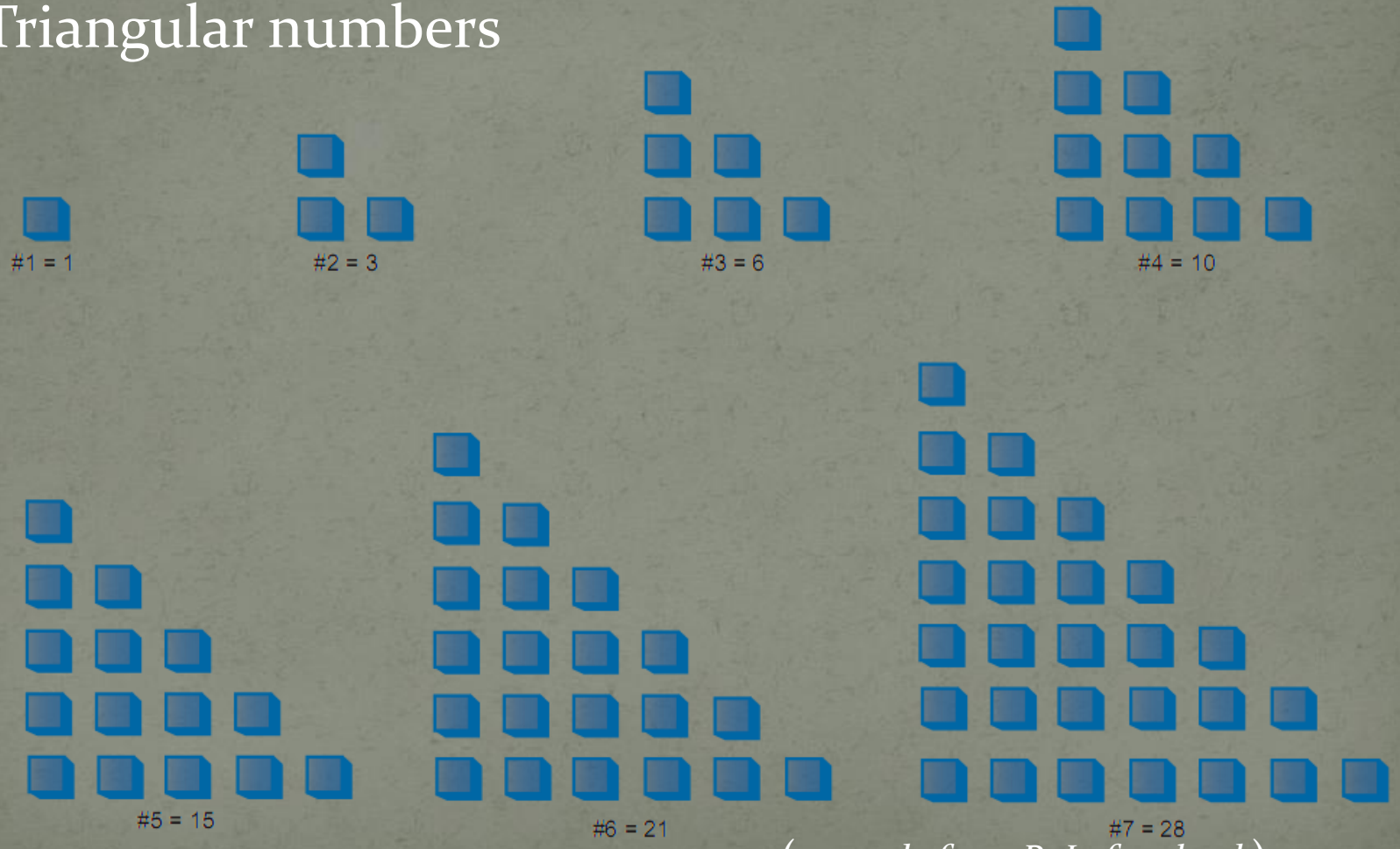
Andrzej Pisarski

Plan of the lecture

- Recursion
  - Weak and strong points
  - Anagrams
  - Binary Search
  - Sorting by Merging
  - Elimination of recursion

Recursion

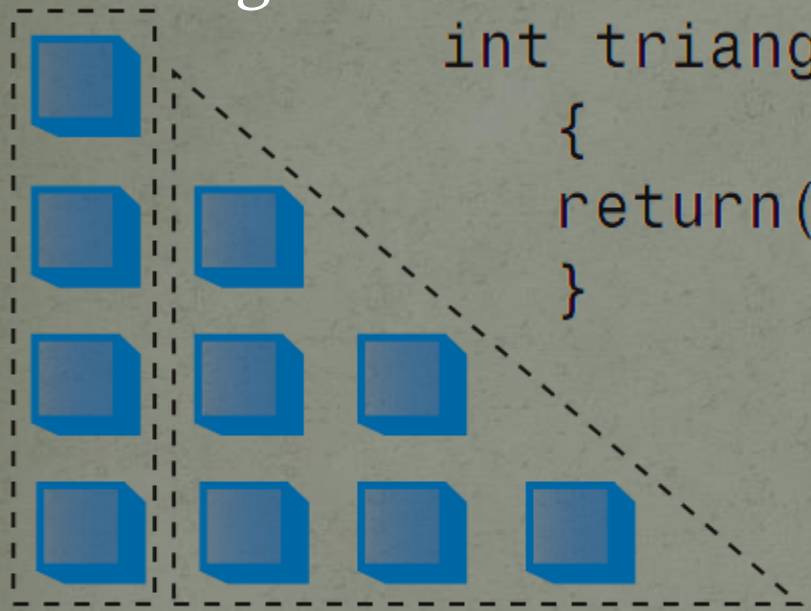
- Triangular numbers



(*example from R. Lafore book*)

Recursion

- Triangular numbers



```
int triangle(int n)
    {
    return( n + sumAllColumns(n-1) );
    }
```

6 in the remaining columns

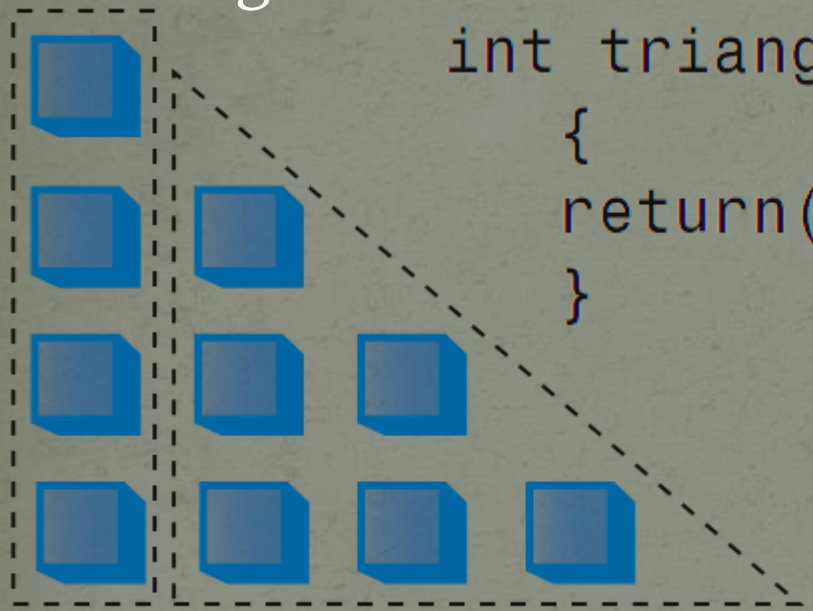
4 in the first column

Total: 10

(*example from R. Lafore book*)

Recursion

- Triangular numbers



```
int triangle(int n)
    {
    return( n + triangle(n-1) );
    }
```

6 in the remaining columns

4 in the first column

Total: 10

(*example from R. Lafore book*)

Recursion

- Triangular numbers
- Base Case

  It is important that every recursive program has a base case (to prevent infinitive call functions)

```
int triangle(int n)
    {
    if(n==1)
       return 1;
    else
       return( n + triangle(n-1) );
    }
```

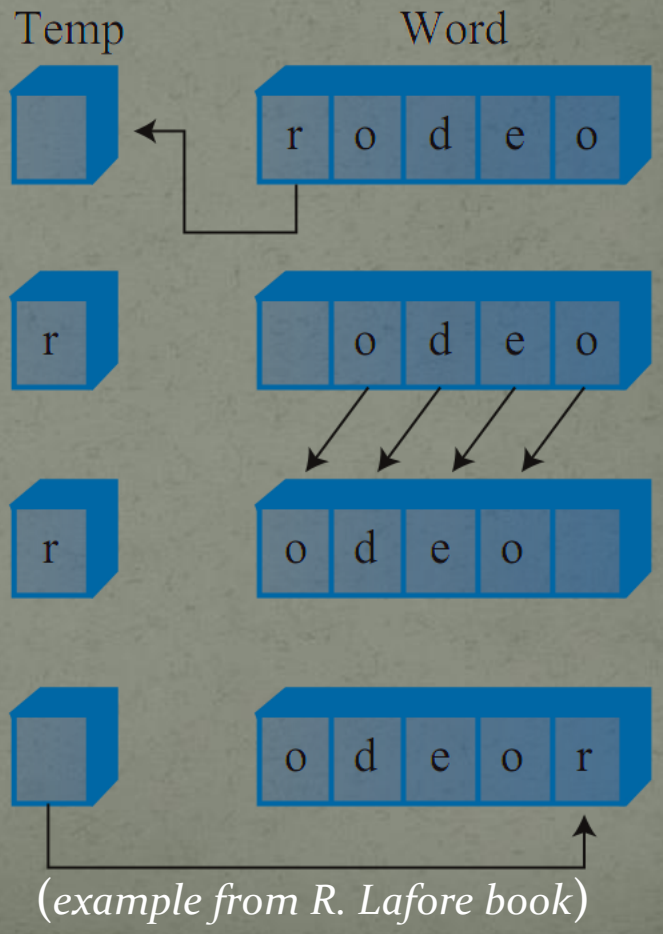
(*example from R. Lafore book*)

Recursion: weak and strong points

- weak points:
  - stack overflow (all data are hold in system's internal stack: arguments, return values, address to function)
  - not so fast (loop approach version can be more quickly)
- strong points:
  + simplifies a problem (can be translated to less complicated problem)

Recursion: anagrams

1. Anagram the rightmost n-1 letters
2. Rotate all n letters
3. Repeat these steps n times



(*example from R. Lafore book*)

Recursion: anagrams

1. Anagram the rightmost n-1 letters
2. Rotate all n letters
3. Repeat these steps n times

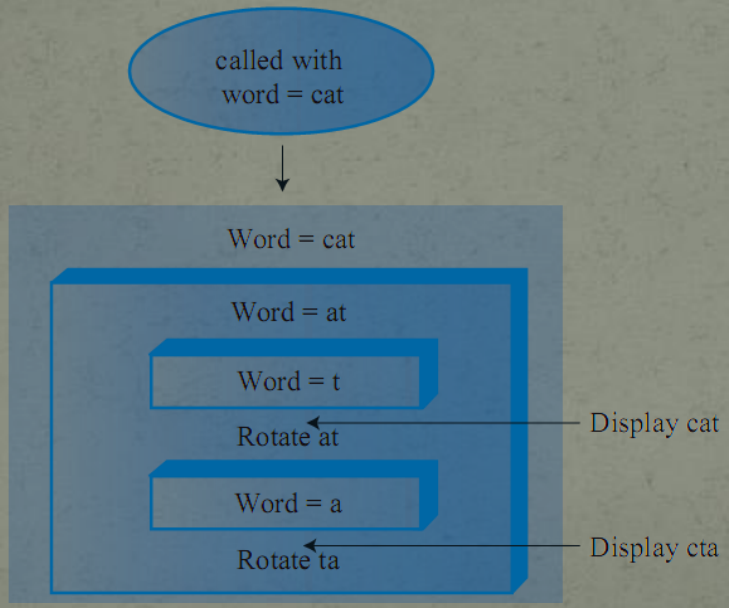
ANAGRAMMING THE WORD CAT

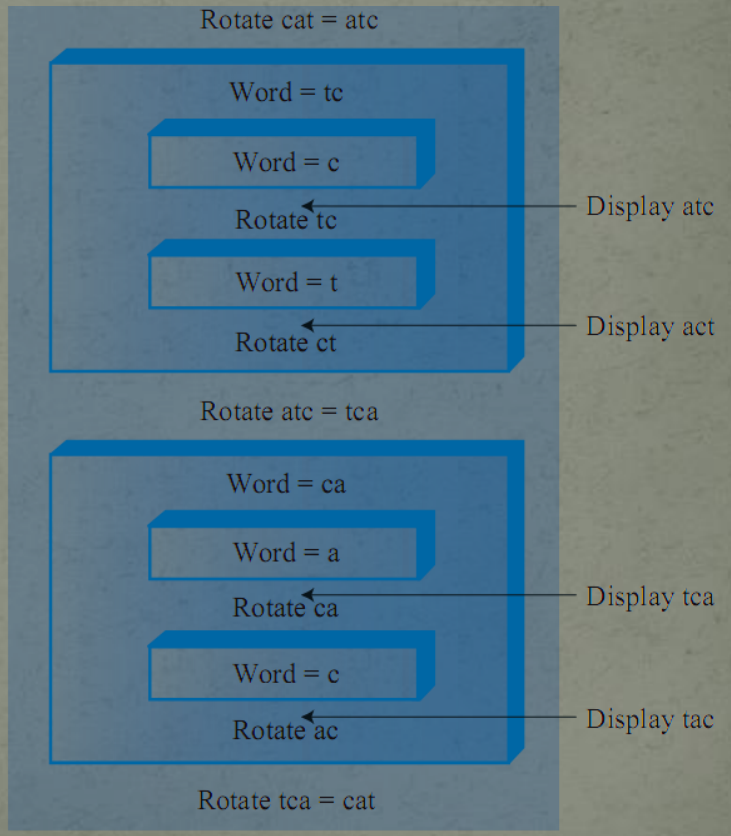
Word	Display Word?	First Letter	Remaining Letters	Action
cat	Yes	c	at	Rotate at
cta	Yes	c	ta	Rotate ta
cat	No	c	at	Rotate cat
atc	Yes	a	tc	Rotate tc
act	Yes	a	ct	Rotate ct
atc	No	a	tc	Rotate atc
tca	Yes	t	ca	Rotate ca
tac	Yes	t	ac	Rotate ac
tca	No	t	ca	Rotate tca
cat	No	c	at	Done

(*example from R. Lafore book*)

Recursion: anagrams

1. Anagram the rightmost n-1 letters
2. Rotate all n letters
3. Repeat these steps n times





(*example from R. Lafore book*)

Recursion: binary search

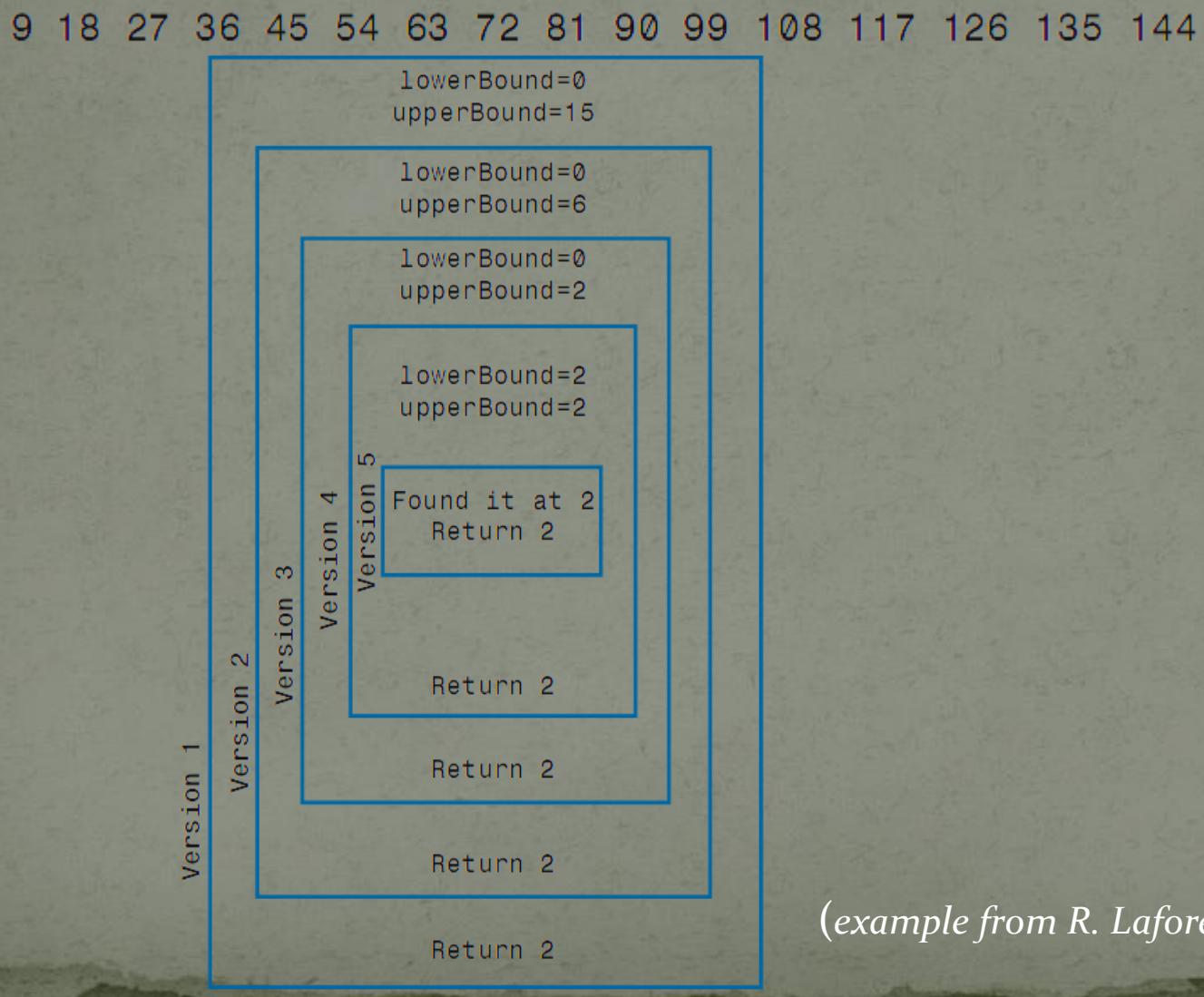
```
 9 18 27 36 45 54 63 72 81 90 99 108 117 126 135 144
```

```
int recFind(double searchKey, int lowerBound, int upperBound)
   {
    int curIn;

    curIn = (lowerBound + upperBound ) / 2;
    if(v[curIn]==searchKey)
       return curIn;                    //found it
    else if(lowerBound > upperBound)
       return nElems;                   //can't find it
    else                                //divide range
       {
       if(v[curIn] < searchKey)         //it's in upper half
          return recFind(searchKey, curIn+1, upperBound);
       else                             //it's in lower half
          return recFind(searchKey, lowerBound, curIn-1);
       }  //end else divide range
    }  //end recFind()
```

(*example from R. Lafore book*)

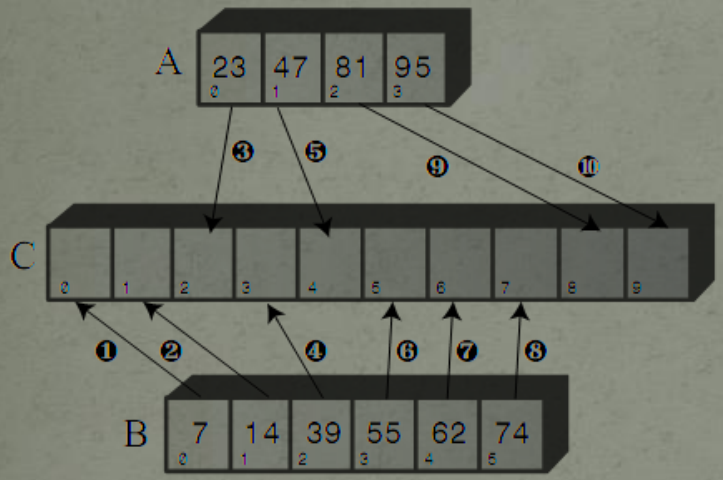
Recursion: binary search



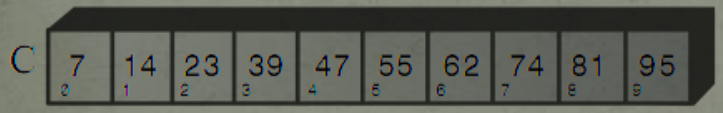
(*example from R. Lafore book*)

Sorting by Merging

- Merging two sorted arrays



a) Before Merge



b) After Merge

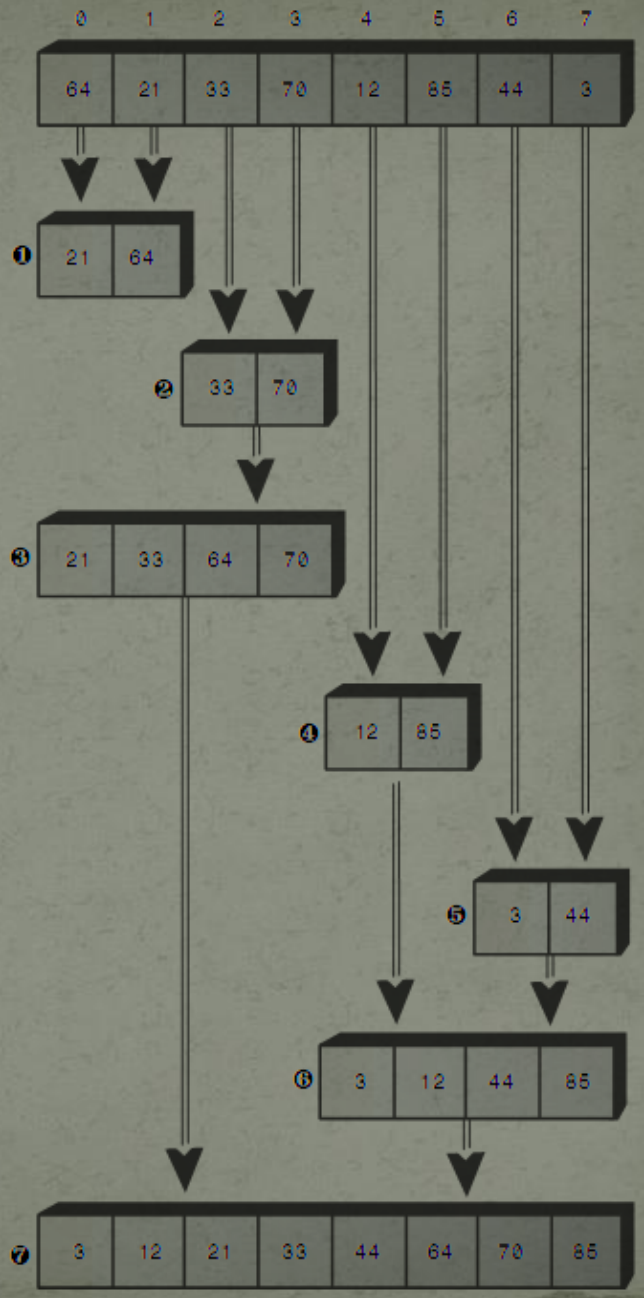
Step	*Comparison (If Any)*	*Copy*
1	Compare 23 and 7	Copy 7 from B to C
2	Compare 23 and 14	Copy 14 from B to C
3	Compare 23 and 39	Copy 23 from A to C
4	Compare 39 and 47	Copy 39 from B to C
5	Compare 55 and 47	Copy 47 from A to C
6	Compare 55 and 81	Copy 55 from B to C
7	Compare 62 and 81	Copy 62 from B to C
8	Compare 74 and 81	Copy 74 from B to C
9		Copy 81 from A to C
10		Copy 95 from A to C

(*example from R. Lafore book*)

Sorting by Merging

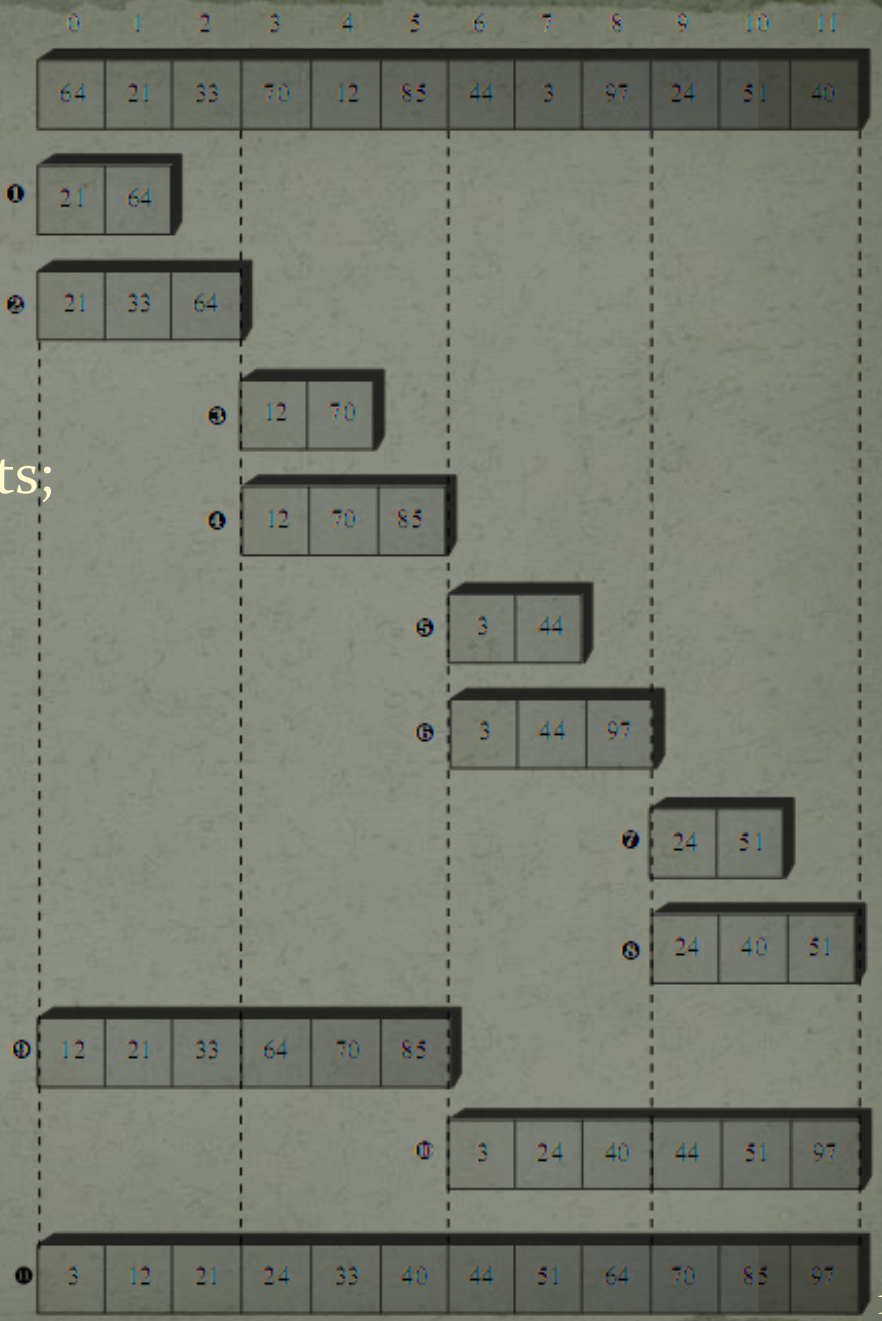
- Array size: power of 2.
  1. Divide an array to half (recursion)
     - Sort two items (size of subarray = 2)
  2. Merge the two halves (contains also sorting)

(*example from R. Lafore book*)



Sorting by Merging

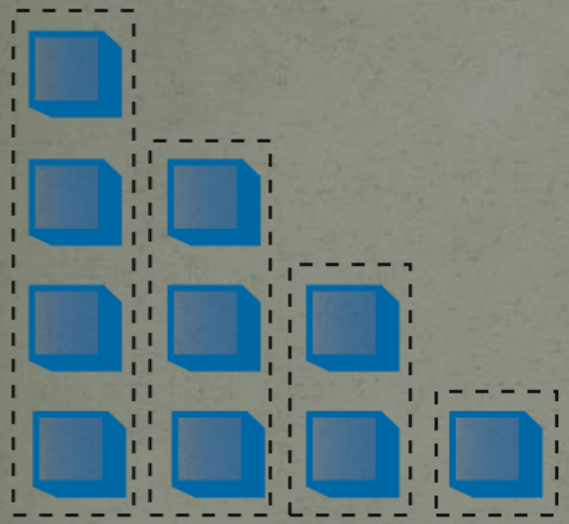
- Array size: not a power of 2.
  1. Divide an array to two (not necessarily equal parts; recursion)
     - Sort two items (size of subarray = 2)
  2. Merge the two parts (contains also sorting)



(*example from R. Lafore book*)

Elimination of recursion

- Triangular numbers (while loop)



```
int triangle(int n)
   {
   int total = 0;

   while(n > 0)              // until n is 1
      {
      total = total + n;     // add n (column height) to total
      --n;                   // decrement column height
      }
   return total;
   }
```

1 in this column
2 in this column
3 in this column
4 in this column

Total: 10

(*example from R. Lafore book*)

Elimination of recursion

- Binary Search (while loop)

```
int find(double searchKey)
   {
   int lowerBound = 0;
   int upperBound = nElems-1;
   int curIn;

   while(true)
      {
      curIn = (lowerBound + upperBound ) / 2;
      if(v[curIn]==searchKey)
         return curIn;              //found it
      else if(lowerBound > upperBound)
         return nElems;             //can't find it
      else                          //divide range
         {
         if(v[curIn] < searchKey)
            lowerBound = curIn + 1; //it's in upper half
         else
            upperBound = curIn - 1; //it's in lower half
         }  //end else divide range
      }  //end while
   }  //end find()
```

(*example from R. Lafore book*)