EE 441 – CH8

SORTING

Instructor: UĞUR HALICI

SORTING

We need to do sorting for the following reasons :

a) By keeping a data file sorted, we can do binary search on it.

b) Doing certain operations, like matching data in two different files, become much faster.

Given a file having n elements (records), there are various sorting methods, having different best and worst case behaviours.

	Best	**Worst**
Bubble sort	$\Omega(n^2)$	$O(n^2)$
Insertion sort	$\Omega(n^2)$	$O(n^2)$
Quick sort	$\Omega(n\log n)$	$O(n^2)$
Merge sort	$\Omega(n\log n)$	$O(n\log n)$

1. SELECTION SORT

- n data items are stored in Array A
- Make n-1 passes over the list
- On pass 0:
  - select the smallest element in the list
  - exchange it with A[0]
  - Front of the list A[0] is ordered
  - Tail of the list A[1] to A[n-1] is unordered
- On pass 1:
  - select the smallest element in the unordered tail list
  - exchange it with A[1]
- Continue until n-1 passes

```
template <class T>
void SelectionSort(T A[ ], int n)
{// index of smallest item in each pass
   int smallIndex;
   int i, j;
// sort A[0]..A[n-2], and A[n-1] is in place
   for (i = 0; i < n-1; i++)
   {
      // start the scan at index i; set smallIndex to i
      smallIndex = i;
      // j scans the sublist A[i+1]..A[n-1]
      for (j = i+1; j < n; j++)
         // update smallIndex if smaller element is found
         if (A[j] < A[smallIndex])  smallIndex = j;
      // when finished,  place smallest item in A[i]
      Swap(A[i], A[smallIndex]);
    }
}
template <class T>
void Swap (T &x, T &y)
{   T temp;
      temp = x; x = y; y = temp;}
```

A	0	1	2	3	4
init	5	4	3	2	1
i=0	1	4	3	2	5
i=1	1	2	3	4	5
i=2	1	2	3	4	5
i=3	1	2	3	4	5
i=4	1	2	3	4	5

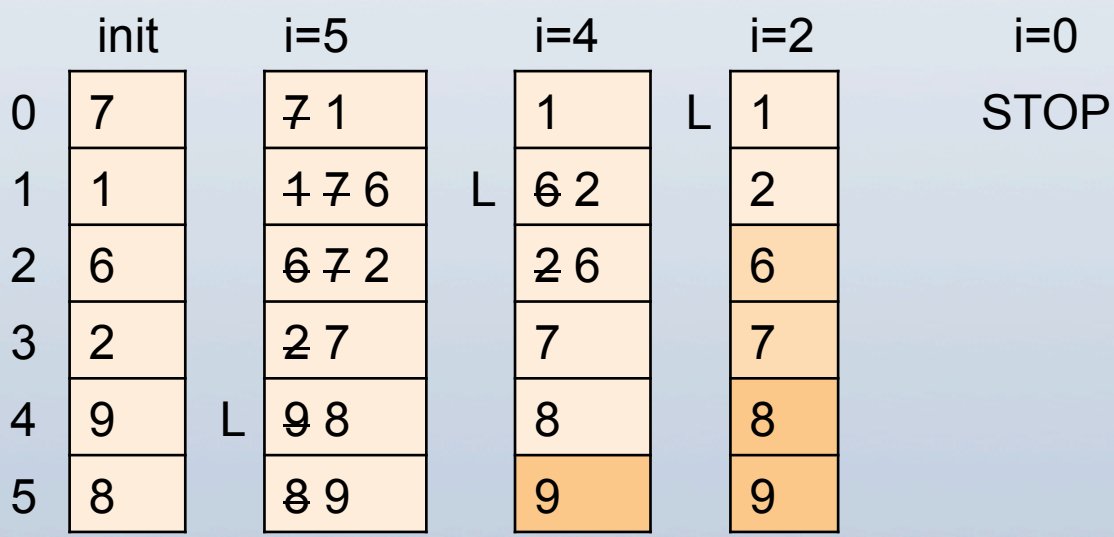
A	0	1	2	3	4
init	12	7	5	2	6
i=0	2	7	5	12	6
i=1	2	5	7	12	6
i=2	2	5	6	12	7
i=3	2	5	6	7	12
i=4	2	5	6	7	12

$O(n^2)$, $\Omega(n2)$, so $\theta(n^2)$

2) BUBBLE SORT

- Array A with n elements
- n-1 passes
- Each pass,
- we compare the adjacent elements
  If A[i-1]>A[j], the order is not correct and we exchange the elements
  At the end of each pass: the largest element goes bottom of the current list, and smaller elements bubbles up towars top
- After pass 0 is complete: the tail of the list A[n-1] is ordered and the front of the list remains unordered
- We keep track of the index of the last exchanged element

```
template <class T>
void BubbleSort(T A[], int n)
{  int i,j;
   int lastExchangeIndex;
   i = n-1; // i is the index of last element in the sublist
   // continue the process until no exchanges are made
while (i > 0)
   { lastExchangeIndex = 0;
      // scan the sublist A[0] to A[i]
      for (j = 0; j < i; j++)
         // exchange a pair and update lastExchangeIndex
         if (A[j] > A[j+1])  { Swap(A[j],A[j+1]); lastExchangeIndex = j; }
      // set i to index of the last exchange. continue sorting the sublist A[0] to A[i]
      i = lastExchangeIndex;
   }
}
```



3) QUICK SORT

Partition approach to sort a list

Main Idea:

- Determine a pivot value
- Split the list into two parts according to the pivot
- Items smaller than the pivot goes into sublist left : sL
- Items larger than the pivot goes into right sublist: sR
- For sL and sR repeat the same procedure

	0	1	2	3	**4**	5	6	7	8	9
A	**55**	15	30	60	80	65	40	35	45	70

Scanning phase:

Low index: low=0

High index: high=9

The pivot: A[low]=55

we scan the entire list A[low+1] to A[9]

2 sublists : sL ≤pivot, sR >pivot

- Two index variables: iL and iR
- iL: responsible for locating the elements in sublist sL

  Initially: iL is set as low+1=1
- iR: responsible for locating the elements in sublist sR

  Initially: iR is set as high=9
- Each pass identifies the elements in each of the sublists

- Each pass identifies the elements in each of the sublists

iL searches the left list. It moves right as far as A[iL] ≤ pivot

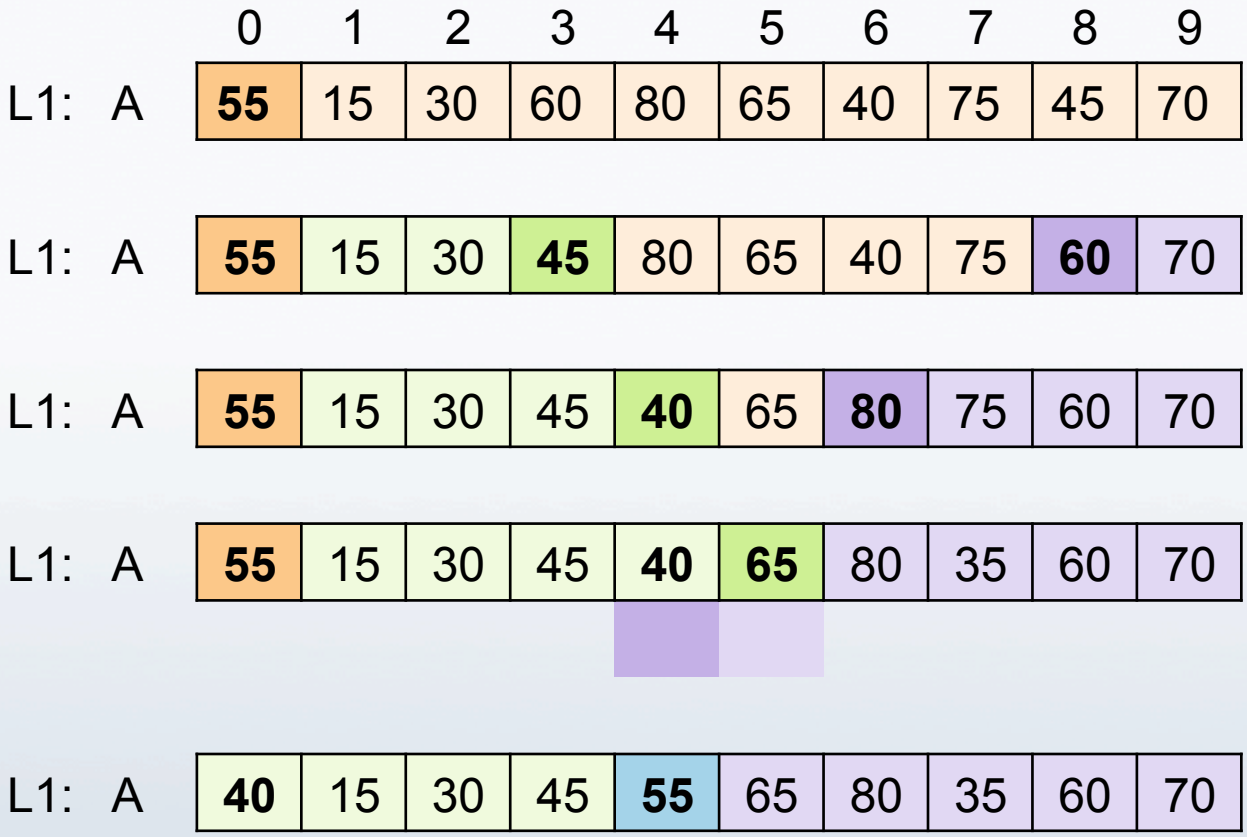
iR searches the right list. It moves left as far as A[iR] > pivot

	0	1	2	3	4	5	6	7	8	9
A	**55**	15	30	**60**	80	65	40	75	**45**	70

iL=3; A[3]=60>55
iR=8; A[8]=45<55

- Those two elements are in wrong sublists so exchange them: Swap(A[iL],A[iR])

	0	1	2	3	4	5	6	7	8	9
A	**55**	15	30	**45**	80	65	40	75	**60**	70



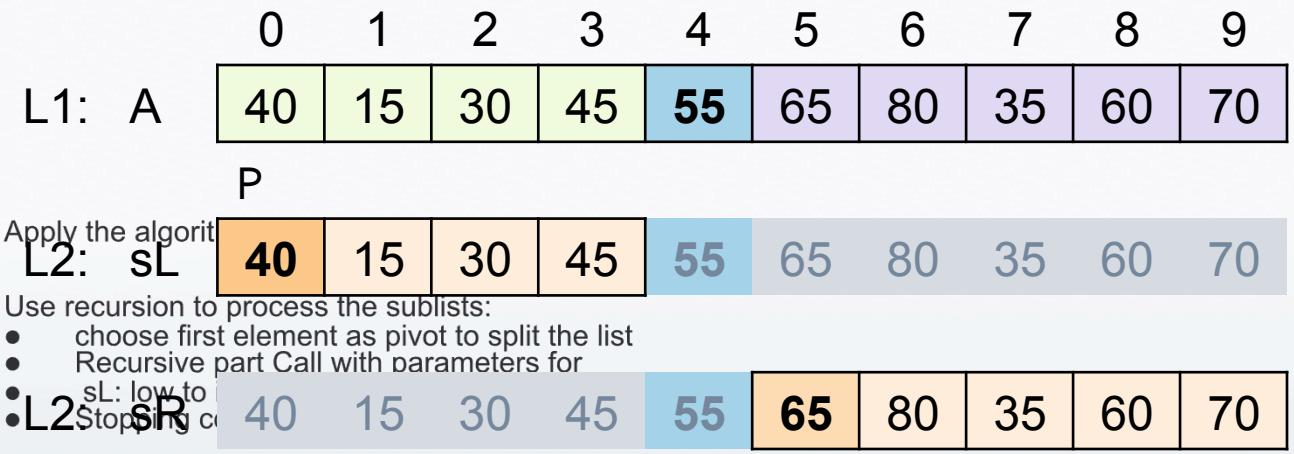
Continue until iL and iR pass each other

iR separates the list, Put Pivot back in its place:

Swap(A[low], A[iR])

then sL=A[0]-A[4], sH=A[6]-A[9]

- Recursive phase:



Apply the algorit

- Use recursion to process the sublists:
  - choose first element as pivot to split the list
  - Recursive part Call with parameters for
  - sL: low to
  - Stopping c

- 300,150,400,450,350
- ScanUp=400:A[2]
- ScanDown=150:A[1]
- Halt the process
- Exchange pivot and ScanDown
- 150,300,400,450,350
- Stop processing Sl because it is single element

- Recursive phase: Level 2, sL

A=sL: low=0, high=3, pivot=A[low]=40

- Recursive phase: Level3, sL

A=sL: low=0, high=1 pivot=A[low]=30

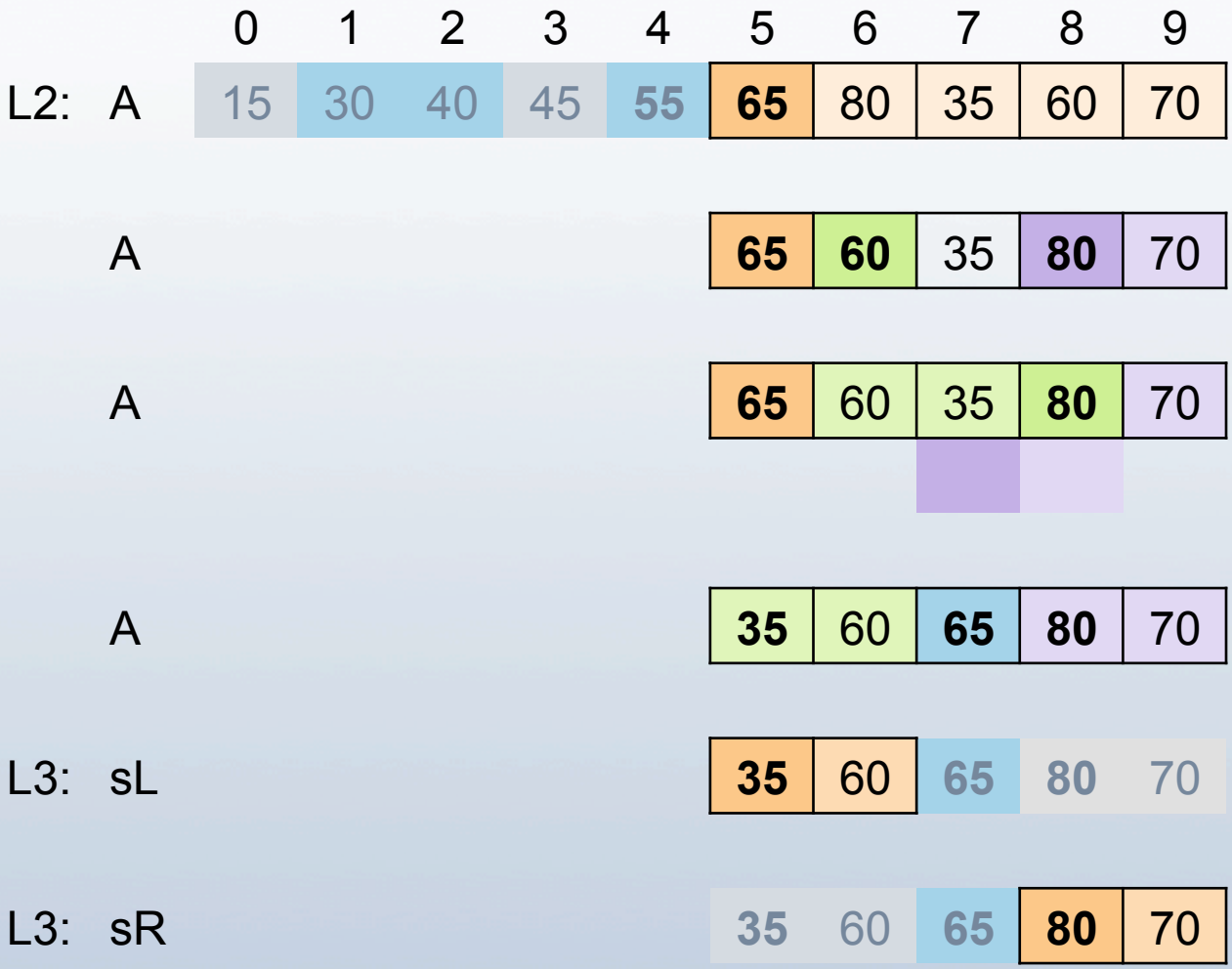
Recursive phase: Level4, sL: 1 element, return back

Recursive phase: Level4, sR: 0 element, return back

Recursive phase: Level3, sR: 1 element, return back

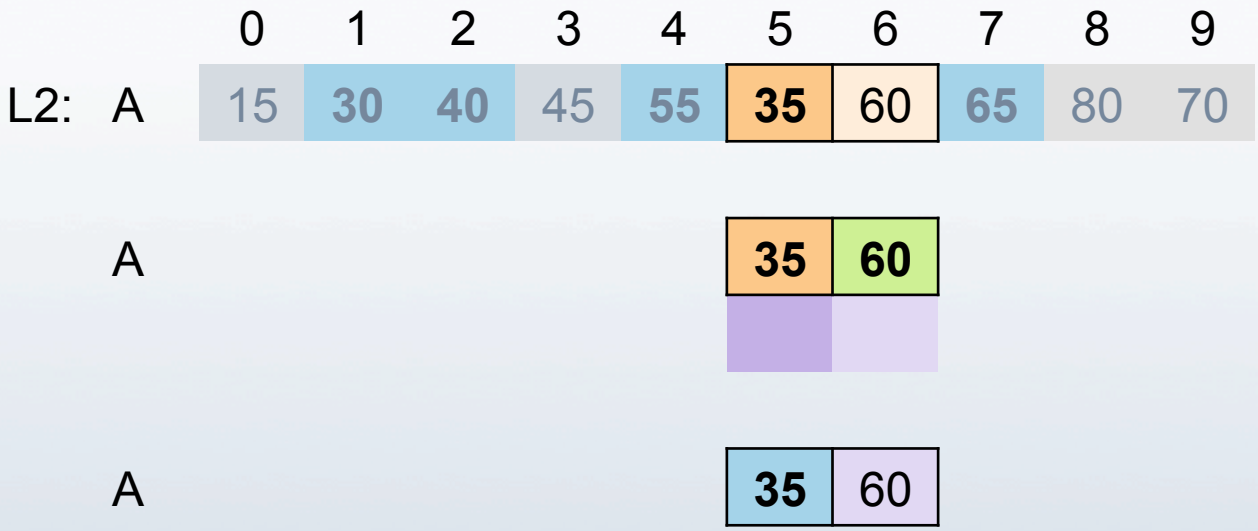
- Recursive phase: Level 2: sR

A=sR: low=5, high=9 pivot=A[low]=65



- Recursive phase: Level 3: sL

A=sL: low=5, high=6 pivot=A[low]=35

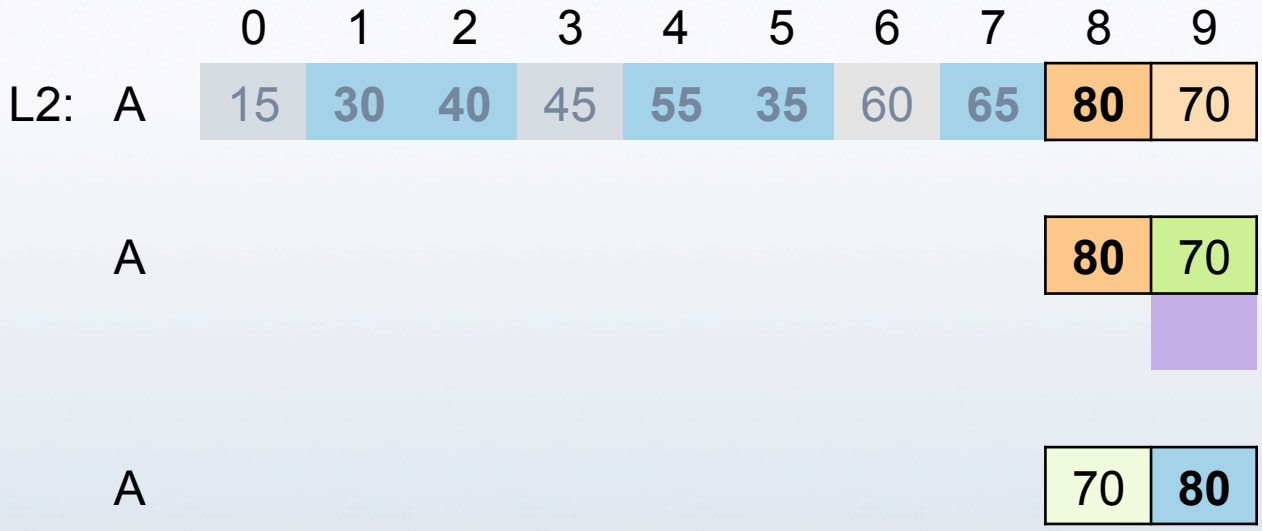


Recursive phase: Level4, sL: 0 element, return back

Recursive phase: Level4, sR: 1 element, return back

- Recursive phase: Level 2: sR

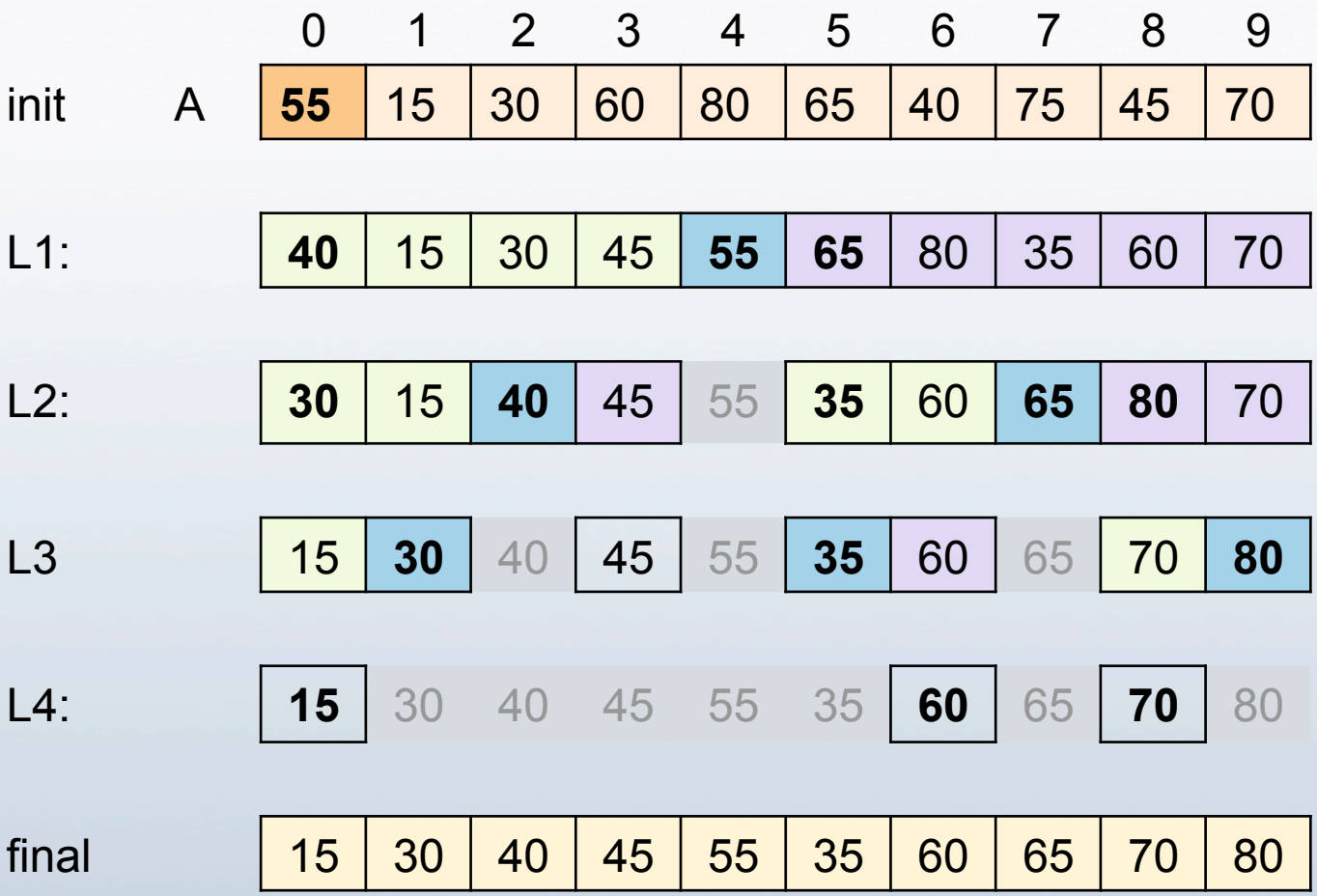
A=sR: low=8, high=9 pivot=A[low]=80



Recursive phase: Level3, sL: 1 element, return back

Recursive phase: Level3, sR: 0 element, return back

- summary: at the end of each level we will have sL and sR as:



```
Template <classT>
void qsort(T *A, int low, int high)
{//quick sort
      if (low<high) {
          int iL=low+1;
          int iR=high;
          T pivot=A[low];
          while (True)
         {     while (A[iL] <=elem) {iL++};
               while (A[iR] >elem) {iR--};
               if (iL<iR)
                swap(A,iL,iR);
               else break;
          }
         swap(A,low,iR);
         qsort(A,low,iR-1);
         qsort(A, iR+1,high);
     }
}
```

4. RADIX SORT

Let's have the following 4-bit binary numbers. Assume there is no sign bit.

1	0	1	**0**	(10)
0	1	0	**1**	(5)
0	0	1	**1**	(3)
1	0	1	**1**	(11)
0	1	1	**0**	(6)
0	1	1	**1**	(7)

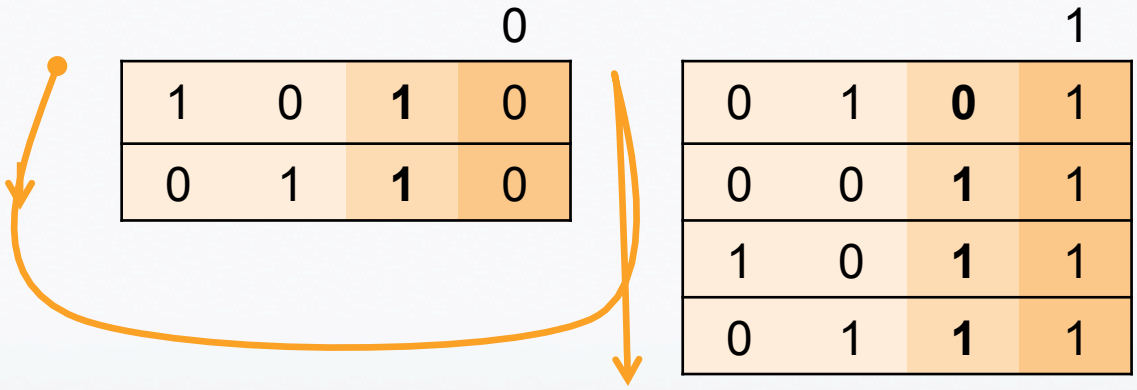
1) First begin with the LSB (least significant bit). Make two groups,one with all numbers that end in a "0" and the other with all numbers that end in a "1".

0

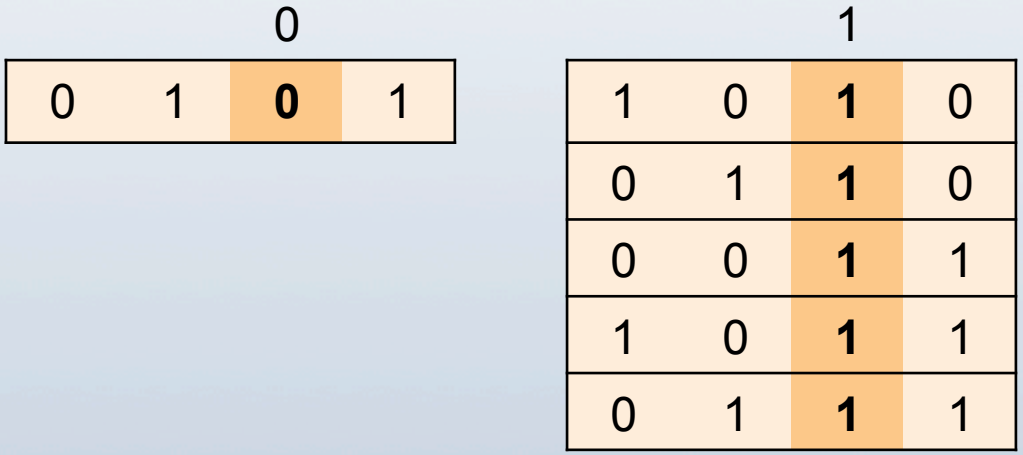
1	0	1	**0**
0	1	1	**0**

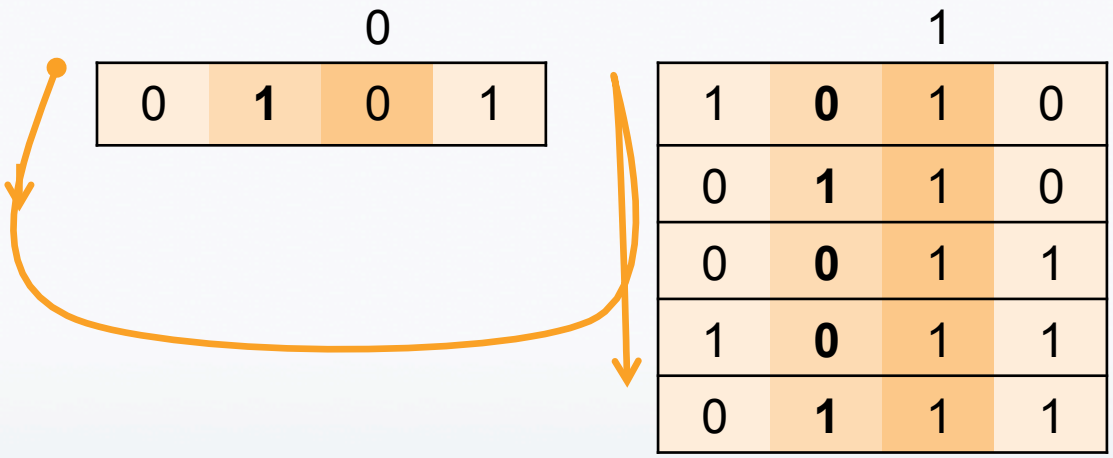
1

0	1	0	**1**
0	0	1	**1**
1	0	1	**1**
0	1	1	**1**

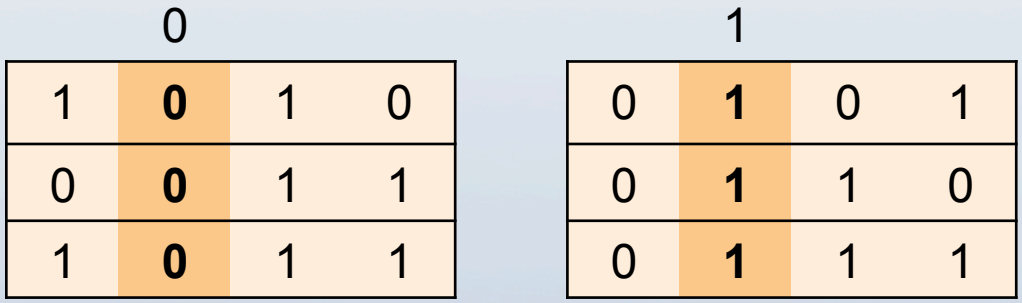


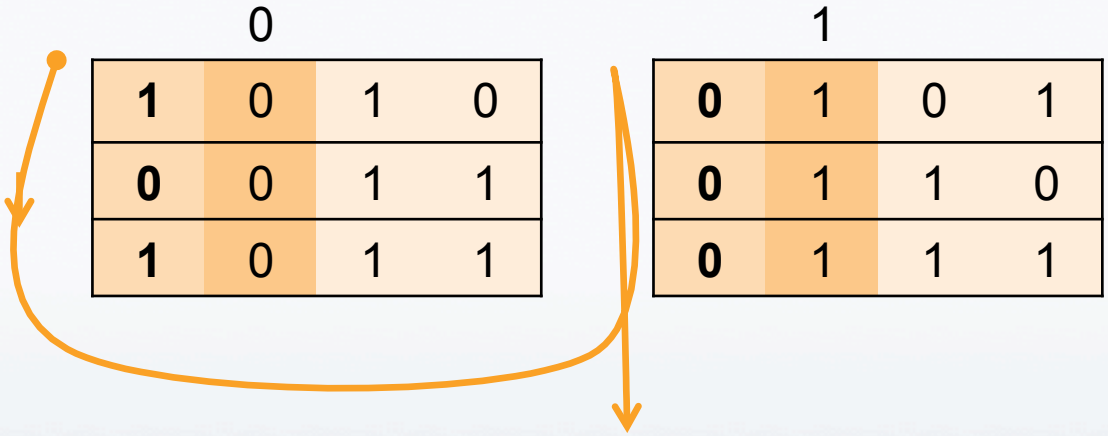
2) Now, go to the next less SB and by examining the previous groups in order, form two new groups:



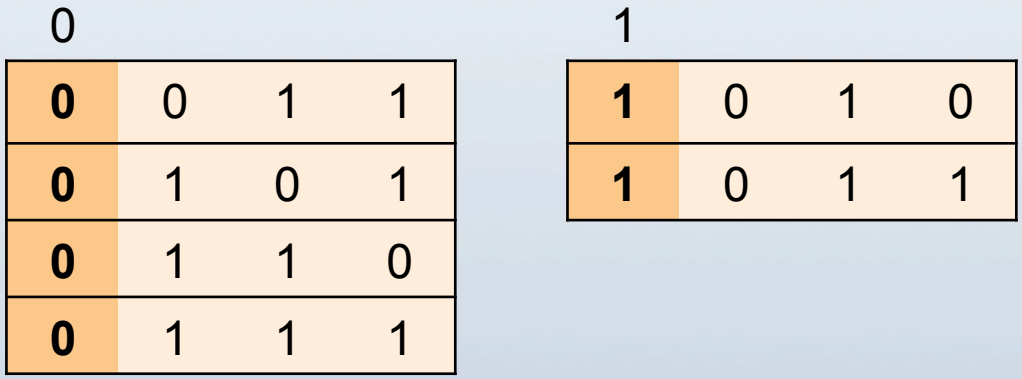


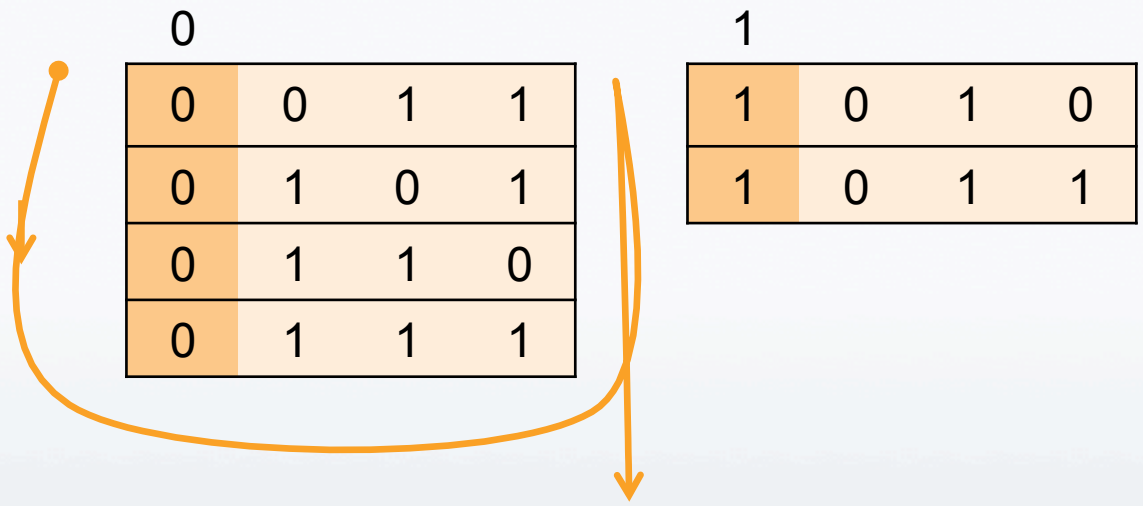
3) Repeat the operation for the third bit from the right:





4) Repeat it for the most significant bit:





Then the result is

0	0	1	1	(3)
0	1	0	1	(5)
0	1	1	0	(6)
0	1	1	1	(7)
1	0	1	0	(10)
1	0	1	1	(11)

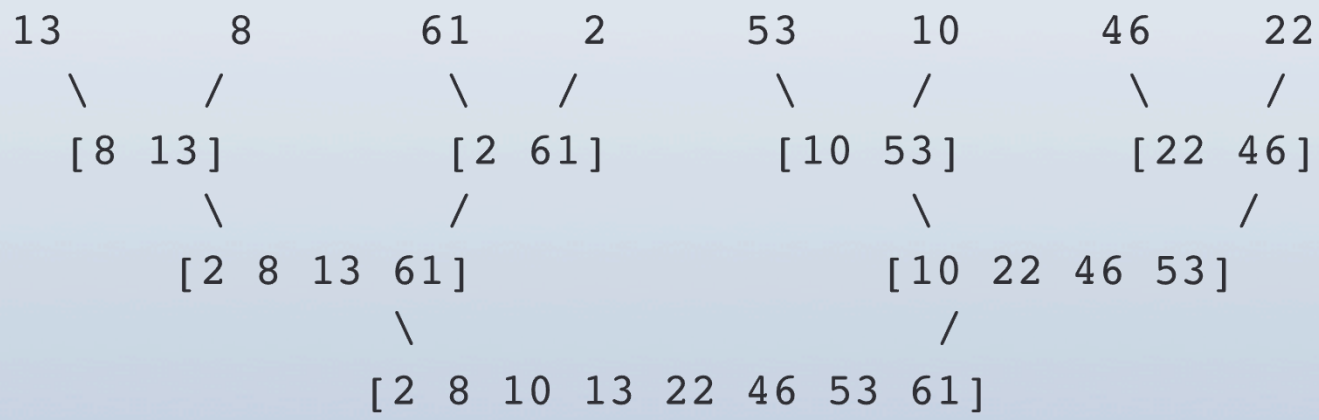
5. MERGE

In merge sort, two already sorted files are 'merged' to obtain a third file which is the sorted combination of the two sorted input files.

- We begin by assuming we have n sorted files with size=1
- Then,we merge these files of size=1 pairwise to obtain n/2 sorted files of size=2
- Then,we merge these n/2 files of size=2 pairwise to obtain n/4 sorted files of size=4,etc..

-Until we are left with one file with size=n.

Example :



To merge two sorted files (x[1].. x[m]) and (y[1]..y[n]), to get a third file (z[1]..z [m +n]) with key1<key2<...<keyn , which will be the sorted combination of them, the following procedure can be used :

```
int MERGE(m,n:integer; int x[ ], int
y[ ]; int &z[ ]);
int i,j,k,p
{i=1; /* i is a pointer to x */
   j=1; /* j is a pointer to y */
   k=1;
 /* k points to the next available
    location in z */
   while (i<=m)&& (j<=n)
    {
        if (x[i].key <= y[j].key)
        { /* take element from x */
          z[k]=x[i];
          i++; }
       else
         { /* take element from y */
           z[k]=y[j];
           j++; };
 k:=k+1
   /* added one more element into z */
   }; /* while */
if ( i>m )
   {
      for (p=j;p<=n; p++)
            z[k++]= y[p]
       /* remainig part of y into z */
   }
   else
{
       for (p=i; j<=m; j++)
            z[k++] = x[p]
        /* remaining part of x into z */
}
  return k-1;
}
```