Merge Sort

cse2011

section 11.1 of textbook

Sorting Problem

- **Input**:
  1. A sequence of ***n*** object
     - Stored in a link list or an array.
  2. A comparator that defines a total order on the objects.
- **Output**:
  - An ordered representation of these objects.

Goals

- Divide-and-conquer approach
- Solving recurrences
- One more sorting algorithm

Merge Sort: Main Idea

Based on **divide-and-conquer** strategy

- **Divide** the list into **two** smaller lists of about equal sizes.
- Sort each smaller list ***recursively***. *(base of recurrence ?)*
- ***Merge*** the two **sorted lists** to get one sorted list.

Questions:

- How do we divide the list? How much **time** needed?
- How do we merge the two sorted lists? How much **time** needed?

Dividing

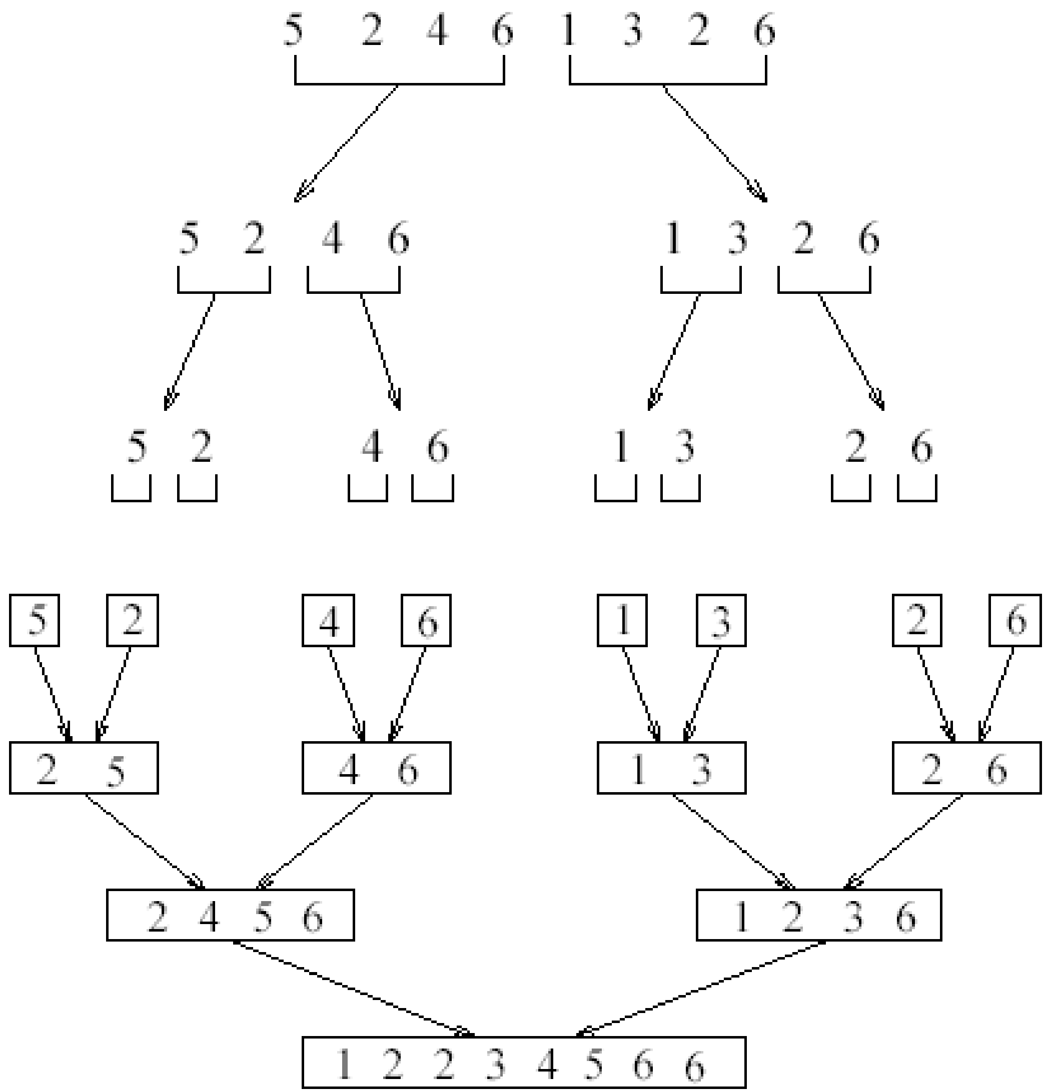
- If the input list is an **array** A[0..N-1]: dividing takes O(1) time:
  - Represent a sub-array by two integers *left* and *right*.
  - To divide A[*left* .. *right*], compute *center*=(*left*+*right*)/2 and obtain A[*left* .. *center*] and A[*center*+1 .. *right*]

- If the input list is a **linked list**, dividing takes **Θ(N)** time:
  - Scan the linked list, stop at the $\lfloor N/2 \rfloor^{th}$ entry and cut the link.

Merge Sort: Algorithm

- Divide-and-conquer strategy
  - **recursively** sort the first half and the second half
  - **merge** the two sorted halves together

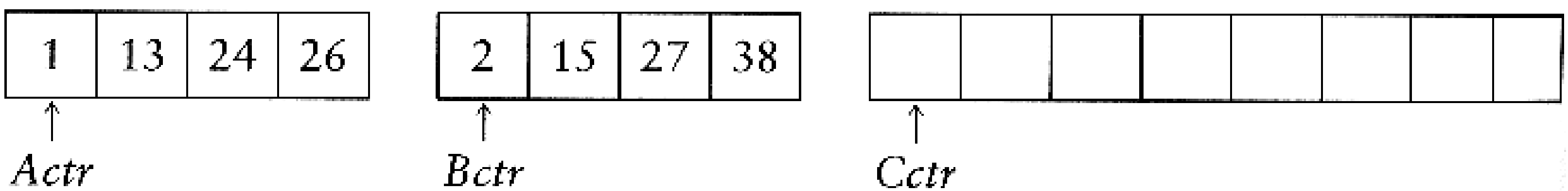
```
void mergeSort(int A[], int left, int right)
    if(left<right){
          int center = (left+right)/2;
          mergeSort(A, left, center);
          mergeSort(A, center + 1, right);
          merge(A, left, center+1, right);
    }
}
```



see applet http://www.cosc.canterbury.ac.nz/people/mukundan/dsal/MSort.html

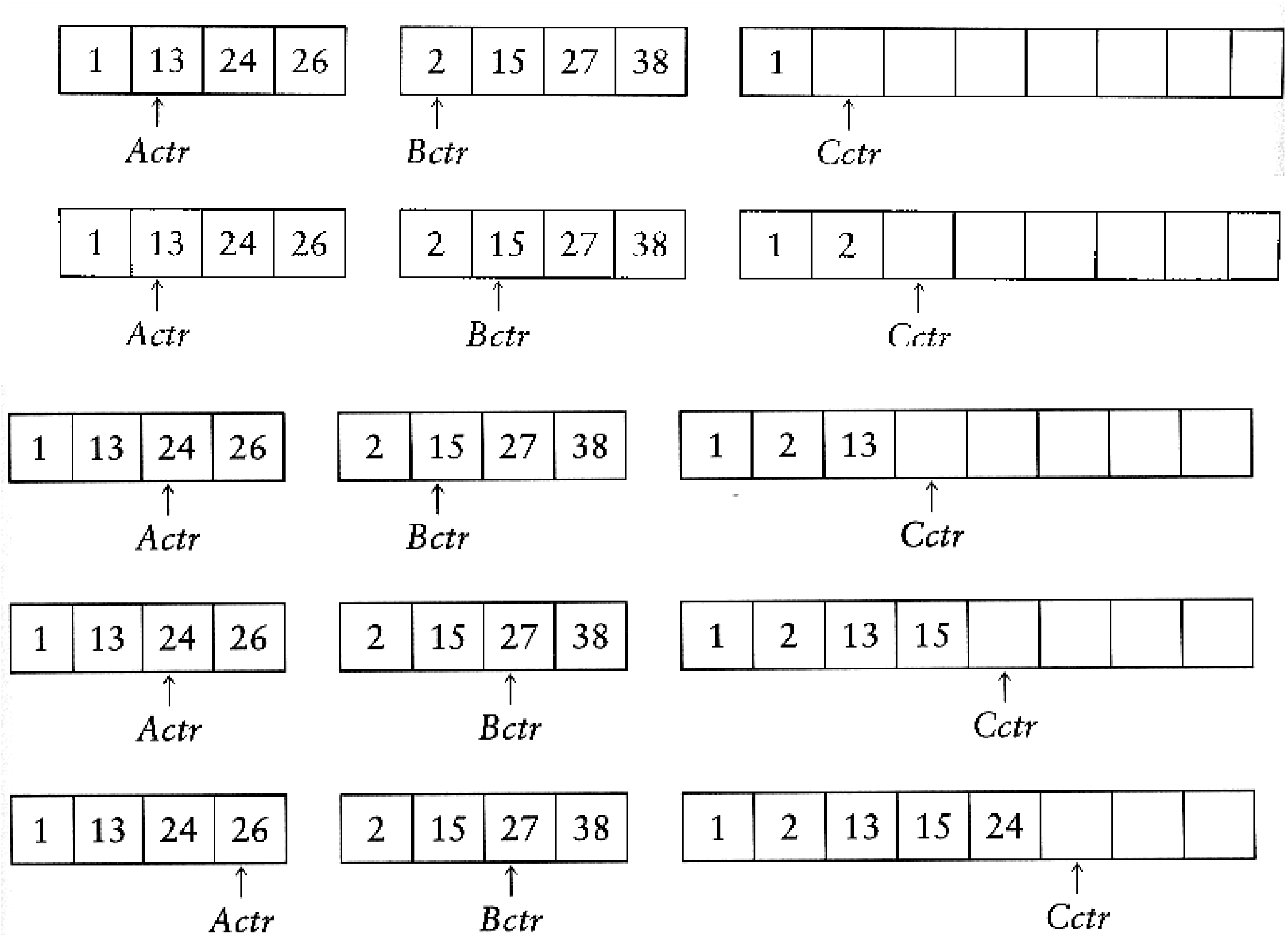
Merging

- **Input**: two sorted array A and B
- **Output**: an output sorted array C
- Three counters: *Actr*, *Bctr*, and *Cctr*
  - initially set to the beginning of their respective arrays

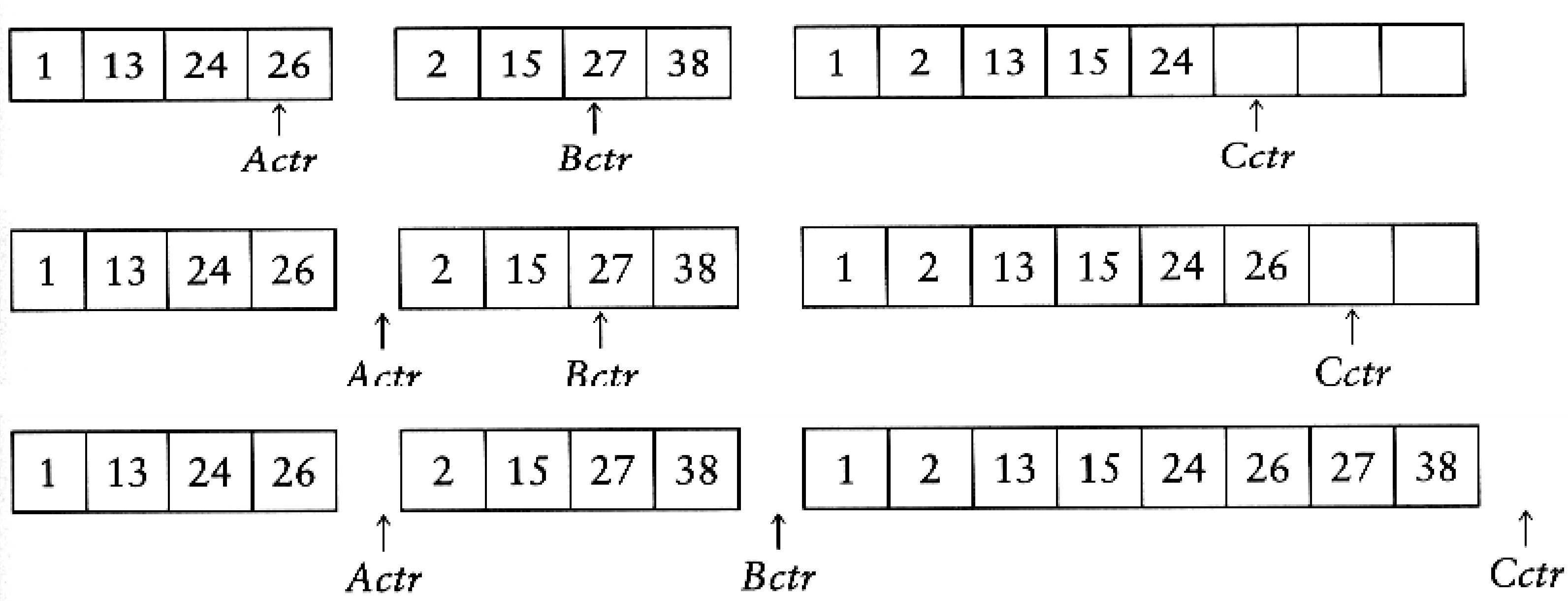


  - The smaller of A[*Actr*] and B[*Bctr*] is copied to the next entry in C, and the appropriate counters are increased.
  - As soon as each of the list becomes empty, the remainder of the other list is copied to C.

Merge: Example



Example: Merge (2)



ArrayList

- **`ArrayList`**
  - A class in the standard Java libraries that can hold any type of object
  - An object that can grow and shrink while your program is running (unlike arrays, which have a fixed length once they have been created)
- In general, an **`ArrayList`** serves the same purpose as an array, except that an **`ArrayList`** can change length while the program is running

Merge: Java Code – Lists Implementations

```
void merge(ArrayList<E> L1, ArrayList<E> L2,  Comparator<E> c, ArrayList<E> rList){
    while (!L1.isEmpty() && !L2.isEmpty()) {
            if(c.compare(L1.get(0), L2.get(0)) <= 0){
                        rList.add(L1.get(0));
                        L1.remove(0);
            } else {
                        rList.add(L2.get(0));
                        L2.remove(0);
            }
    }
    while(!L1.isEmpty()){
            rList.add(L1.get(0));
            L1.remove(0);
    }
    while(!L2.isEmpty()){
            rList.add(L2.get(0));
            L2.remove(0);
    }
}
```

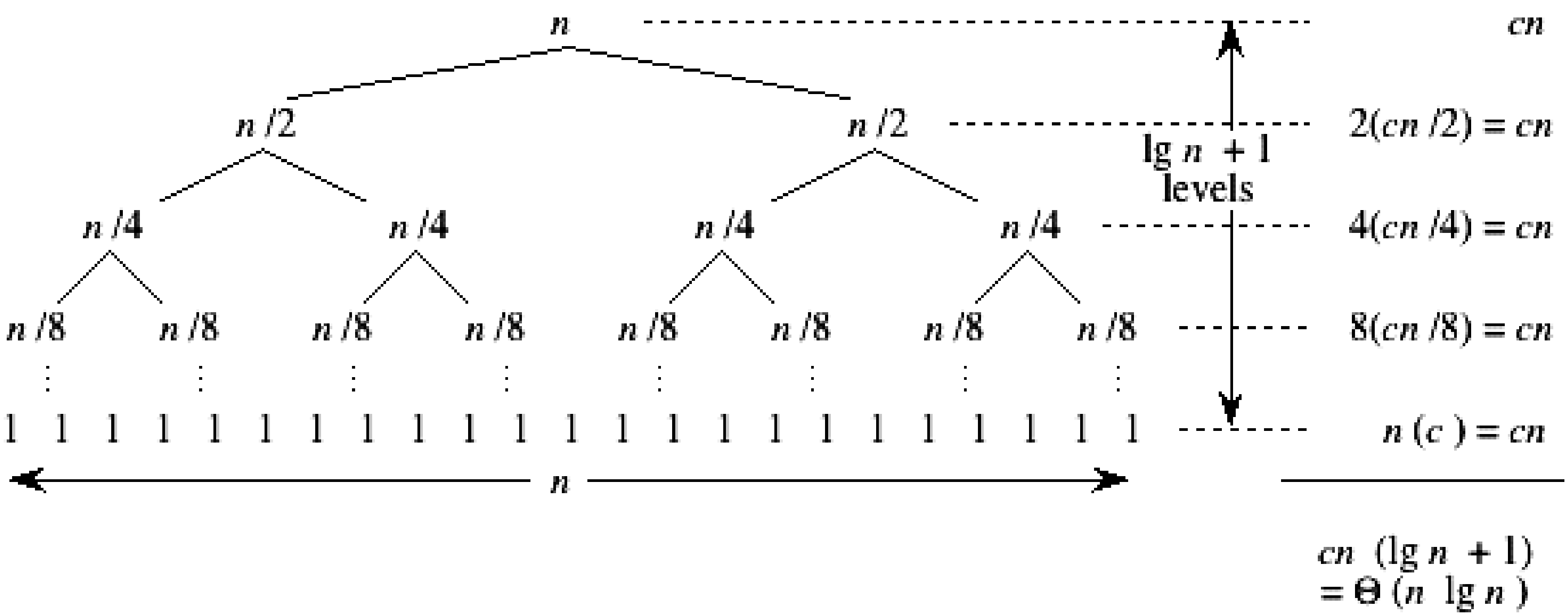
Merge: Analysis

- Running time analysis:
  - Merge takes $O(m_1 + m_2)$, where m_1 and m_2 are the sizes of the two sub-arrays.

- Space requirement:
  - merging two sorted lists requires **linear extra memory**
  - additional work to copy to the temporary array and back

Analysis of Merge Sort – <u>First Approach</u>

- Merge sort tree has **O(log *n*)** levels.
- The time spent at any node excludes the recursive call
- Merge sort tree has ***2^i*** nodes at depth ***i***.
- The number of elements at each node in depth ***i*** is **$n/2^i$**.



Analysis of Merge Sort - Second Approach

recurrence relation

`void mergeSort(int A[], int left, int right)`	$T(N)$
`   if(left<right){`	$\theta(1)$
`      int center = (left+right)/2;`	$\theta(1)$
`      mergeSort(A, left, center);`	$T(N/2)$
`      mergeSort(A, center + 1, right);`	$T(N/2)$
`      merge(A, left, center+1, right);`	$\theta(N)$
`   }`	
`}`	

Analysis of Merge Sort - Second Approach

- Let T(N) denote the worst-case running time of ***mergesort*** to sort N numbers.
- Assume that N is a power of 2.

- Divide step: O(1) time
- Conquer step: 2 x T(N/2) time
- Combine step: O(N) time

- Recurrence equation:

  T(1) = 1

  T(N) = 2T(N/2) + N

Solving the Recurrence

Since $N=2^k$, we have $k=\log_2 n$

$$
\begin{aligned}
T(N) &= 2T(\frac{N}{2}) + N \\
&= 2(2T(\frac{N}{4}) + \frac{N}{2}) + N \\
&= 4T(\frac{N}{4}) + 2N \\
&= 4(2T(\frac{N}{8}) + \frac{N}{4}) + 2N \\
&= 8T(\frac{N}{8}) + 3N = \cdots \\
&= 2^k T(\frac{N}{2^k}) + kN
\end{aligned}
$$

$$
\begin{aligned}
T(N) &= 2^k T(\frac{N}{2^k}) + kN \\
&= N + N\log N \\
&= O(N\log N)
\end{aligned}
$$