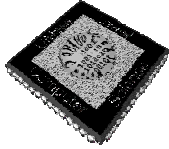
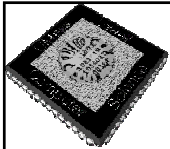


# Divide-and-conquer approach

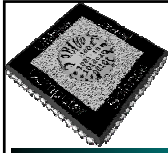


Text  
Chapters 2



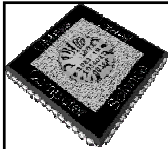
## Outline

- What's divide-and-conquer
- How to analyze a divide-and-conquer algorithm
- Examples: merge sort, binary search



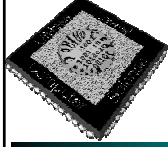
## What is divide and conquer

- A technique for designing algorithms that decompose instance into smaller sub-instance of the same problem
  - Solving the sub-instances independently
  - Combining the sub-solutions to obtain the solution of the original instance



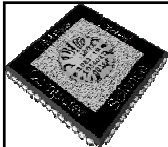
## Divide-and-conquer

- basic steps:
  - **divide** the problem into sub-problems similar to original problem but smaller in size
  - **conquer** the sub-problems recursively
  - **combine** solutions to create solution to original problem



## Merge Sort Algorithm

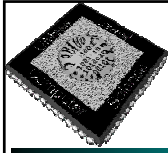
- **Divide:** divide  $n$ -element sequence into two sub-sequences of  $n/2$  elements each
- **Conquer:** sort two sub-sequences recursively using merge sort
- **Combine:** Merge the two sorted sub-sequences to produce the sorted answer



## Merge Sort Algorithm

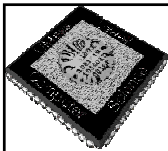
Merge-Sort  $A[1..n]$

1. if  $n = 1$ , done.
2. Recursively sort  $A[1..\lceil n/2 \rceil]$  and  $A[\lceil n/2 \rceil + 1..n]$
3. Merge the two sorted lists

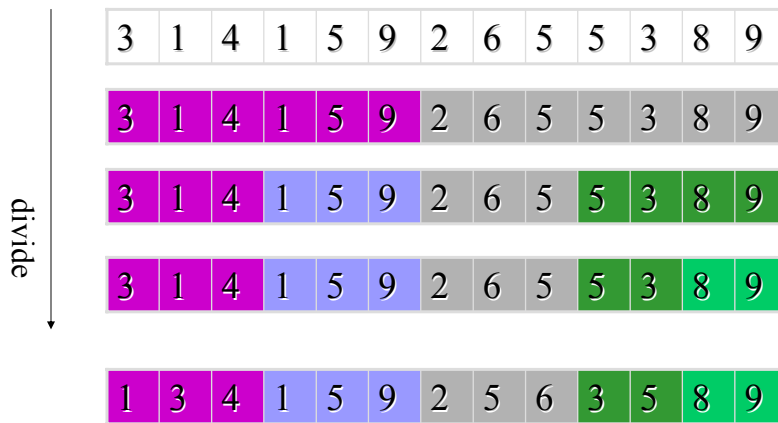


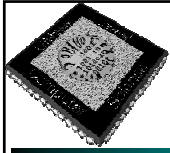
## Merging Two Sorted Lists

- choose the smaller element of the two lists
- remove it from list and put it into a list
- repeat previous steps

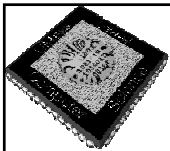
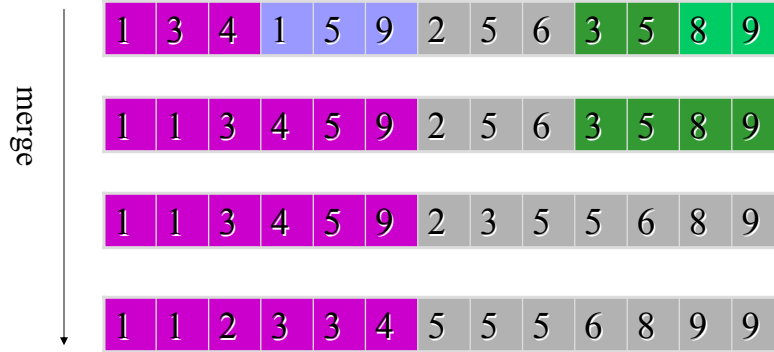


## Merge sort: top down

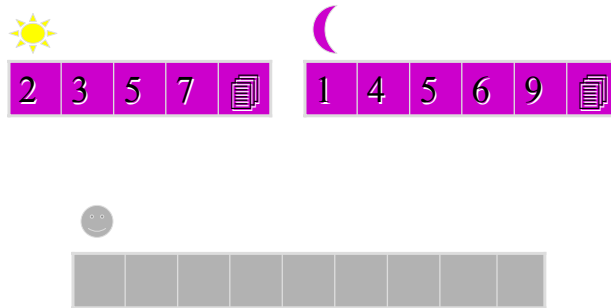


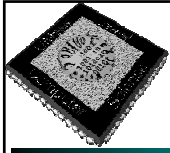


## Merge sort: top down



## Merge two arrays





## Merge two arrays



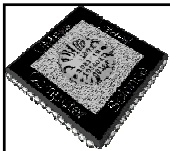
2 3 5 7



1 4 5 6 9



1



## Merge two arrays



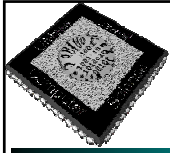
2 3 5 7



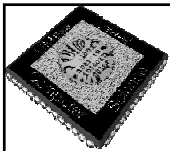
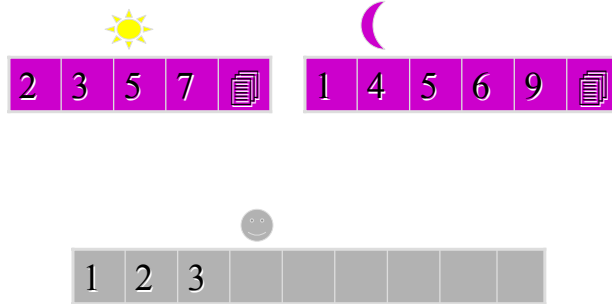
1 4 5 6 9



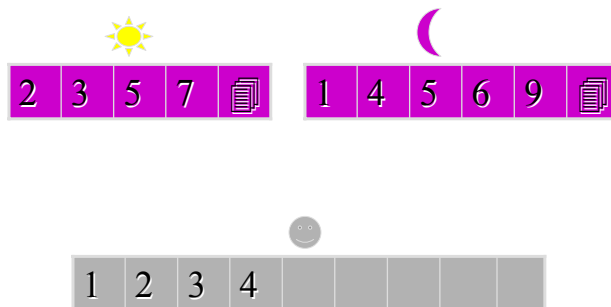
1 2

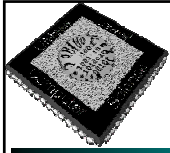


## Merge two arrays

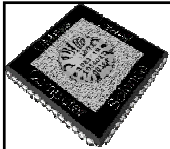
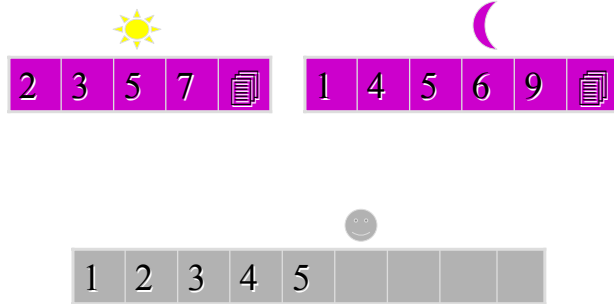


## Merge two arrays

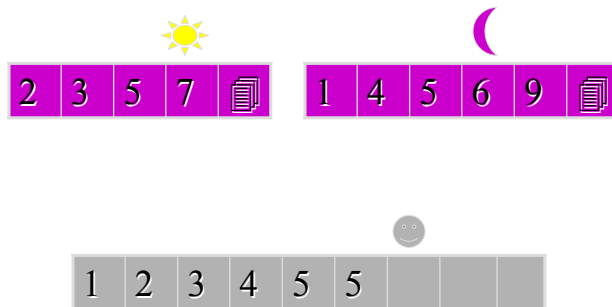


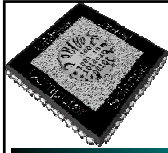


## Merge two arrays

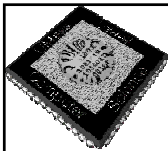
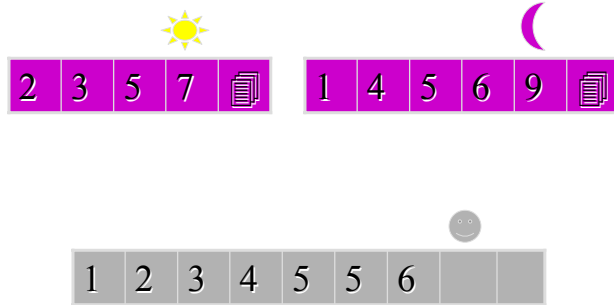


## Merge two arrays

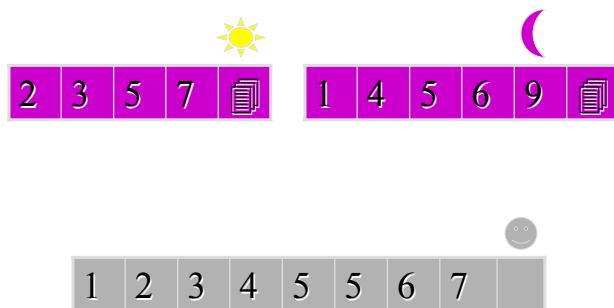


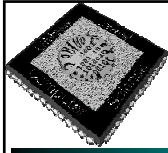


## Merge two arrays

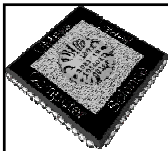
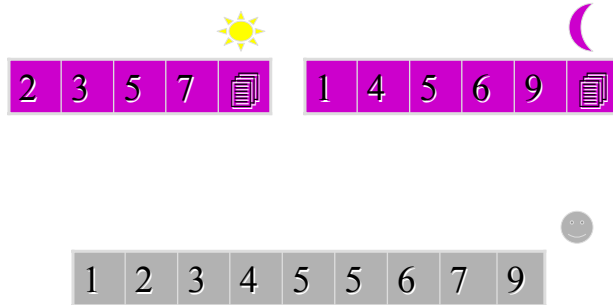


## Merge two arrays

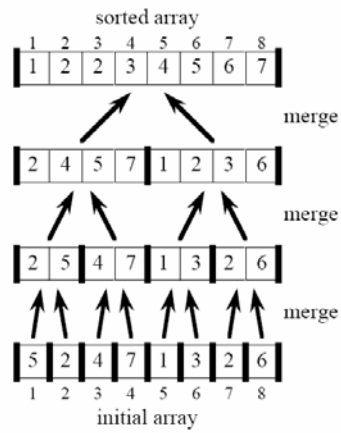


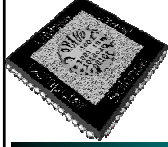


## Merge two arrays



## Merge Sort

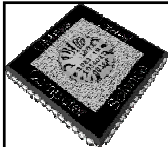




## Analyzing Merge Sort

$T(n)$  Merge-Sort  $A[1..n]$   
 $\Theta(1)$  ← 1. if  $n=1$ , done.  
 $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$  ← 2. Recursively sort  $A[1..\lceil n/2 \rceil]$   
 $\sim 2T(n/2)$  and  $A[\lfloor n/2 \rfloor + 1..n]$   
 $\Theta(n)$  ← 3. Merge the two sorted lists

Recurrence: 
$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ 2T(n/2) + \Theta(n), & \text{if } n > 1 \end{cases}$$



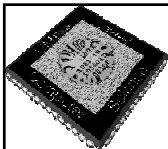
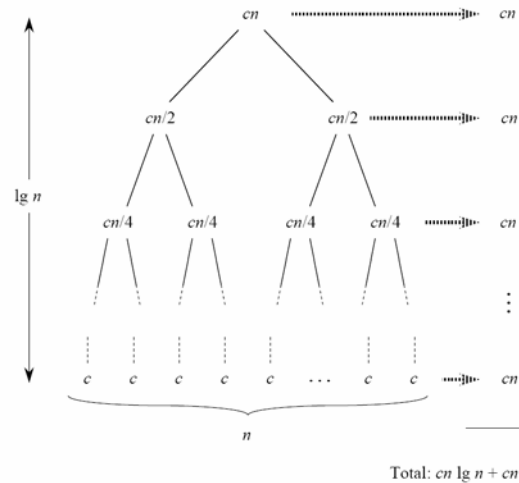
## Recursion Tree

$$T(n) = \begin{cases} c, & \text{if } n = 1 \\ 2T(n/2) + cn, & \text{if } n > 1 \end{cases}$$

$$T(n) = cn \lg n + cn = \Theta(n \lg n)$$



## Recursion Tree



## A general template

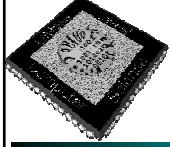
```
DC(x)
{
  if (x is sufficiently small or simple)
    adhoc(x); // use a basic sub-algorithm

  decompose x into small instances x[0],...,x[l-1]; // divide

  for (i=0; i<l; i++) //conquer
    s[i] = DC(x[i]);

  combine s[0],..., s[l-1] to obtain solution s for x; // combine
  return s;
}
```

- Three conditions to be considered
  - When to use the basic sub-algorithm
  - Efficient decomposition and recombination
  - The sub-instances must be roughly the same size

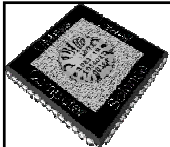


## Sequential Search from a sorted sequence

- ❑ T[] is a sequence in nondecreasing order
- ❑ Find an element in T[]

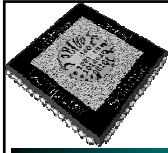
```
sequentialSearch(T[], x)
{
    for (i=0; i<n; i++) {
        if (T[i] == x)
            return i;
    }
}
```

Cost: best, worst, average?



## Binary Search

- ❑ **Divide**: check middle element
- ❑ **Conquer**: recursively search 1 subarray
- ❑ **Combine**: trivial



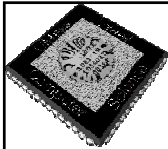
## Binary Search: Example

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15



## Cost of binary search

$$T(n) = 1 T(n/2) + \Theta(1)$$

number of sub-problem      size of sub-problem      work dividing and combining

$$T(n) = \Theta(\lg n)$$