Data Structures and Algorithms

Lecture 6

Insertion Sort And Merge Sort

Department of Computer Science & Technology
United International College

Outline

- Motivation
- Insertion Sort
- Merge Sort
- Divide and Conquer

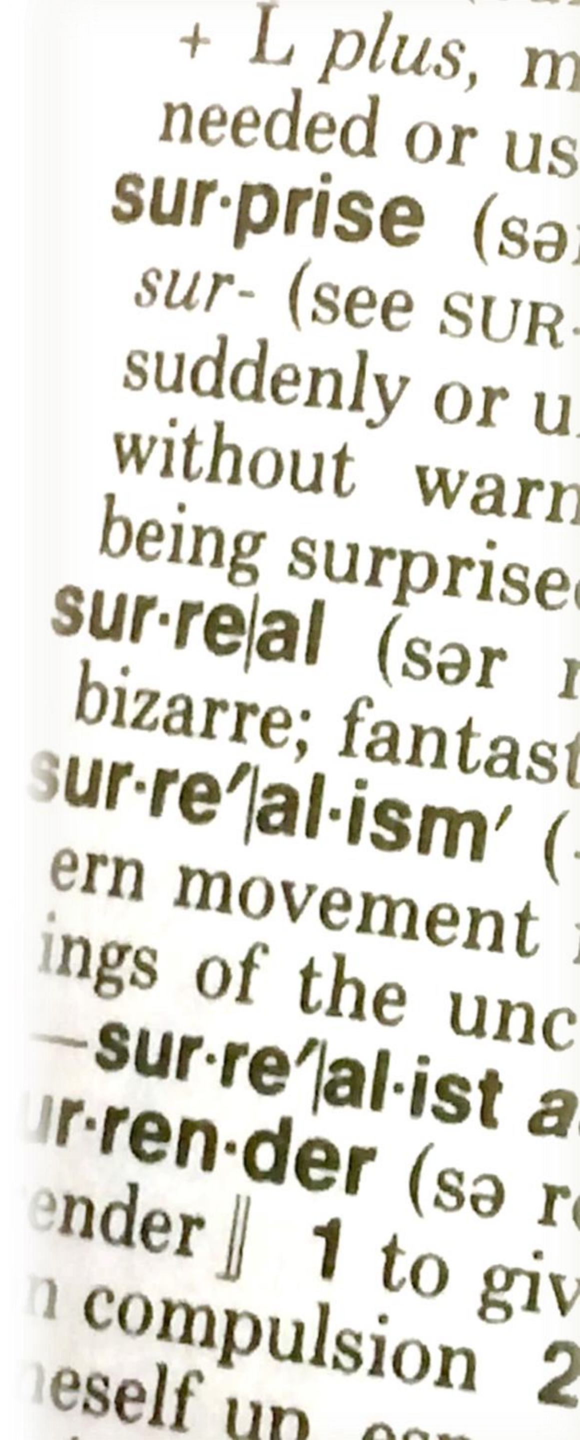
Motivation

How do you quickly find

- Your name in a name list?
- A book on a shelf?
- A word in a dictionary?

Sort

them beforehand!



Insertion Sort

"Let the first *p* items be sorted."

Insertion Sort

1) Initially $p = 1$

2) Let the first p elements be sorted.

3) Insert the $(p+1)$th element properly in the list so that now $p+1$ elements are sorted.

4) increment p and go to step (3)

How is Insertion Done?

3) Insert the $(p+1)$th element properly in the list...

- Scan leftwards
- Move every greater element one position to the right
  - Thus making room for the new element
- Stop when
  - a smaller or equal element is found
  - the left boundary is reached
- Move the new element in
- Animation

Pseudo Code for Insertion Sort

```
INSERTION-SORT(A)
    FOR p = 1 TO n-1
        key = A [p]
        i = p – 1
        WHILE i >= 0 AND A[i] > key
            A[i+1] = A[i]
            i = i – 1
        A[i+1] = key
```

Discussion

- What is the best case for insertion sort?
  - Best case running time?
- What is the worst case for insertion sort?
  - Worst case running time?
- What is the “average” running time?
  - Assume that all possible inputs are of the same probability.

Analysis of Insertion Sort

Best-case Running Time	$O(n)$
Worst-case Running Time	$O(n^2)$
Average Running Time	$O(n^2)$

- Insertion sort is an instable sorting algorithm
  - The running time largely depends on the input
  - It is considered an $O(n^2)$ algorithm

Fun Animation

- Something you may perform in a talent show ヽ(✿ﾟ▽ﾟ)ノ

[The insertion sort dance]

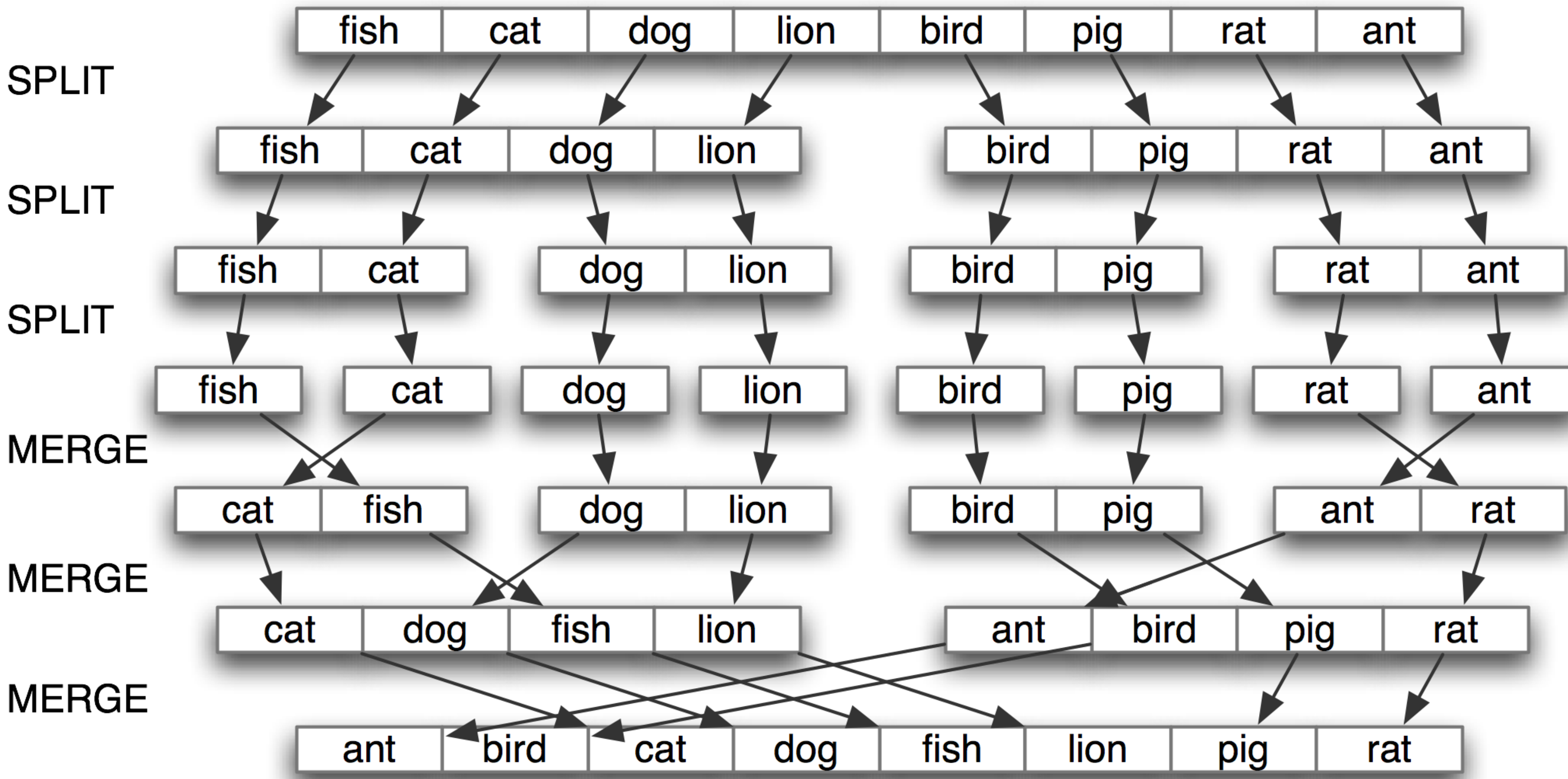
Merge Sort

A divide-and-conquer (DC) algorithm

Merge Sort

- Divide the list into two smaller lists of about equal sizes
- Sort each smaller list *recursively*
- Merge the two sorted lists to get one sorted list
- Animation

Merge Sort Example



Questions to Ponder

How do we divide the list?
How much time is needed?

How do we merge the two sorted lists?
How much time is needed?

Dividing

- If the input list is a linked list, dividing takes $\Theta(N)$ time
  - We scan the linked list, stop at the $\lfloor N/2 \rfloor$ th entry and cut the link
- If the input list is an array A[0..N-1]: dividing takes $O(1)$ time
  1. represent a sublist by two indexes `left` and `right`
  2. to divide `A[left..Right]`, we compute `center=(left+right)/2` and obtain `A[left..Center]` and `A[center+1..Right]`
- Array is usually used as the data structure for sorting

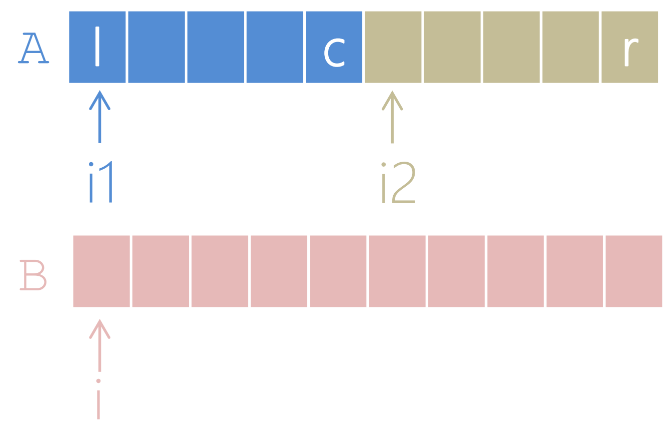
Mergesort

```
MERGESORT(A, left, right)
1.    IF left>=right
2.        RETURN
3.    center = (left+right) / 2
4.    MERGESORT(left, center)
5.    MERGESORT(center+1, right)
6.    MERGE(A, left, center, right)
```

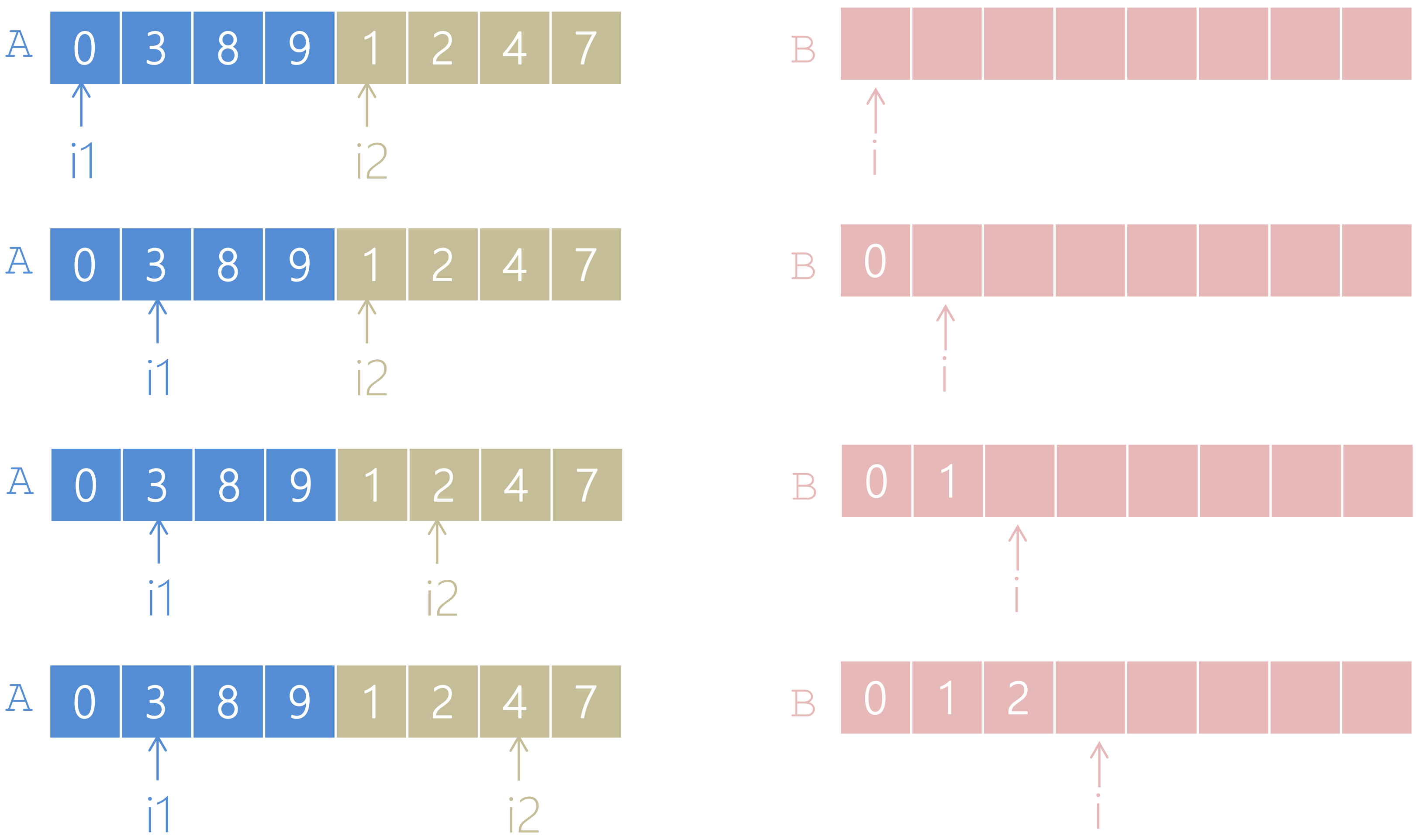
Merging

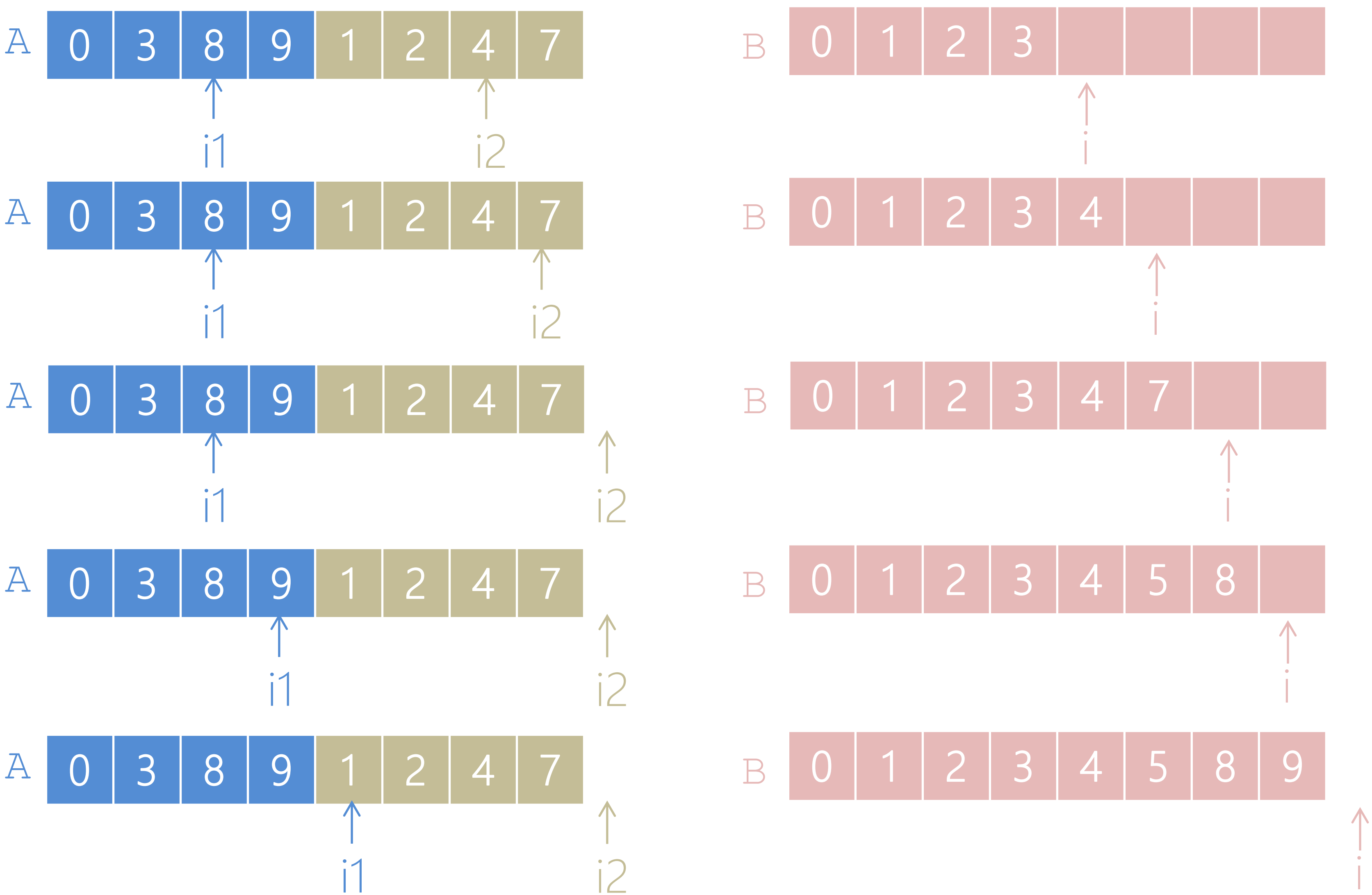
```
MERGE(A, left, center, right)
    i1 = left, i2 = center+1, i=0
    WHILE i1<=center AND i2<=right
        IF A[i1]<A[i2]
            B[i++] = A[i1++]
        ELSE
            B[i++] = A[i2++]
    FOR i1 TO center
        B[i++] = A[i1++]
    FOR i2 TO right
        B[i++] = A[i2++]
    Copy B to A[left..right]
```

- Merge two sorted sub-arrays `A[left..center]` and `A[center+1, right]` into `A[left..right]`
- Use an extra array, `B`.



Merge Example




A 0 3 8 9 1 2 4 7
i1
i2
B 0 1 2 3
i
A 0 3 8 9 1 2 4 7
i1
i2
B 0 1 2 3 4
i
A 0 3 8 9 1 2 4 7
i1
i2
B 0 1 2 3 4 7
i
A 0 3 8 9 1 2 4 7
i1
i2
B 0 1 2 3 4 5 8
i
A 0 3 8 9 1 2 4 7
i1
i2
B 0 1 2 3 4 5 8 9
i

Discussion on Merge

- Suppose that `A[left..right]` contains `n` elements
  - What is the worst-case running time?
  - What is the best-case running time?
  - What is the extra storage cost?

Analysis of Merge Sort

- Let `T(n)` denote the worst-case running time of MergeSort where `n` is the number of items to be sorted
- Assume that `n` is a power of 2.

Divide: O(1) time
Conquer: 2T(n/2) time
Combine: O(n) time

- Recurrence equation:

$$T(n) = \begin{cases} 2T(n/2) + O(n), & n > 1 \\ O(1), & n = 1 \end{cases}$$

Analysis of Merge Sort

- Solve the recurrence relation,

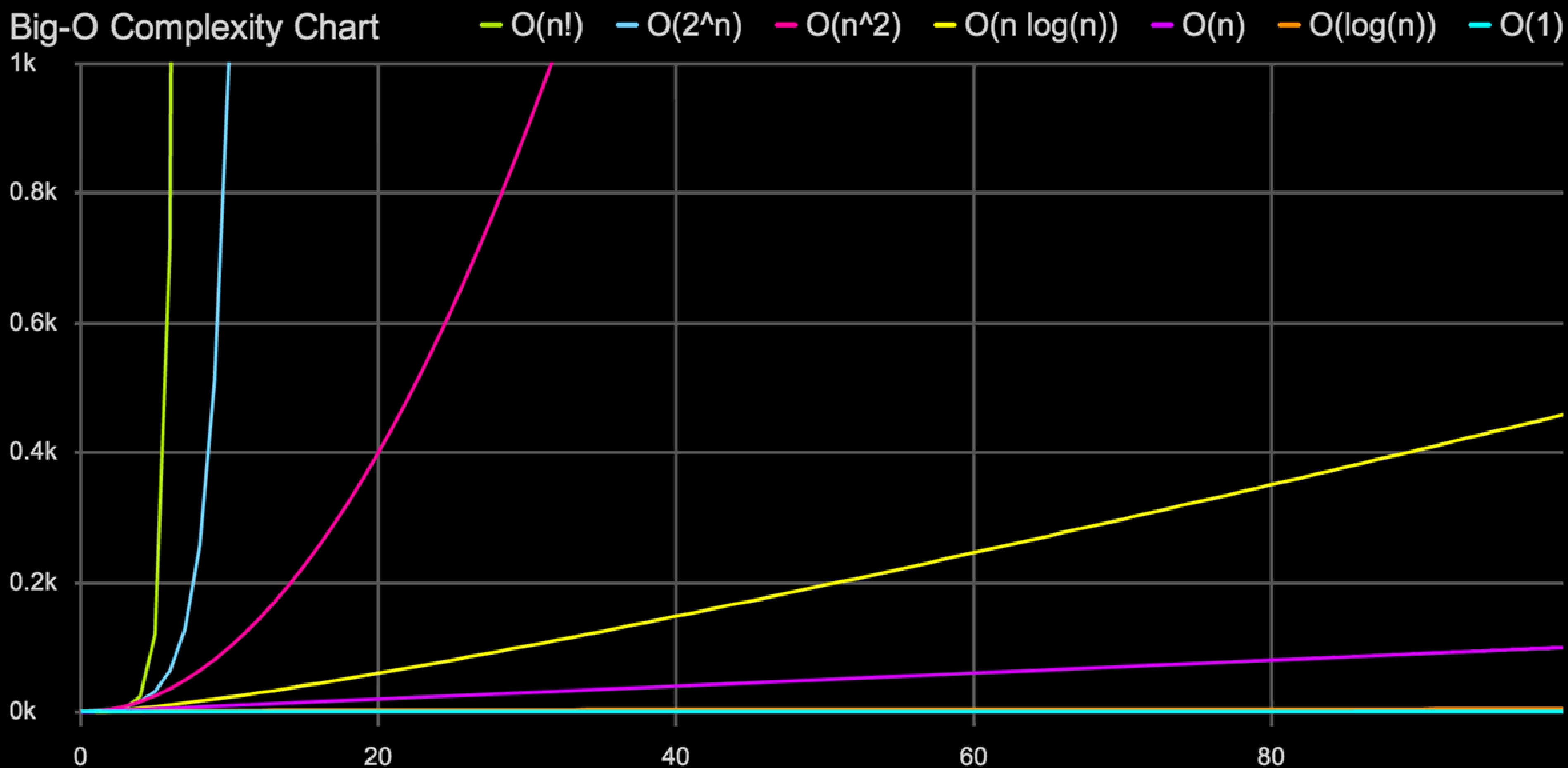
$$T(n) = O(n \log n)$$

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&= 2[2T(n/2^2) + n/2] + n \\
&= 2^2T(n/2^2) + 2n \\
&= 2^3T(n/2^3) + 3n \\
&= 2^iT(n/2^i) + i*n
\end{aligned}
$$

Let i=log(n),

$$
\begin{aligned}
&= nT(n/n) + n*\log(n) \\
&= O(n*\log(n))
\end{aligned}
$$

n*log(n) is much faster than n²!



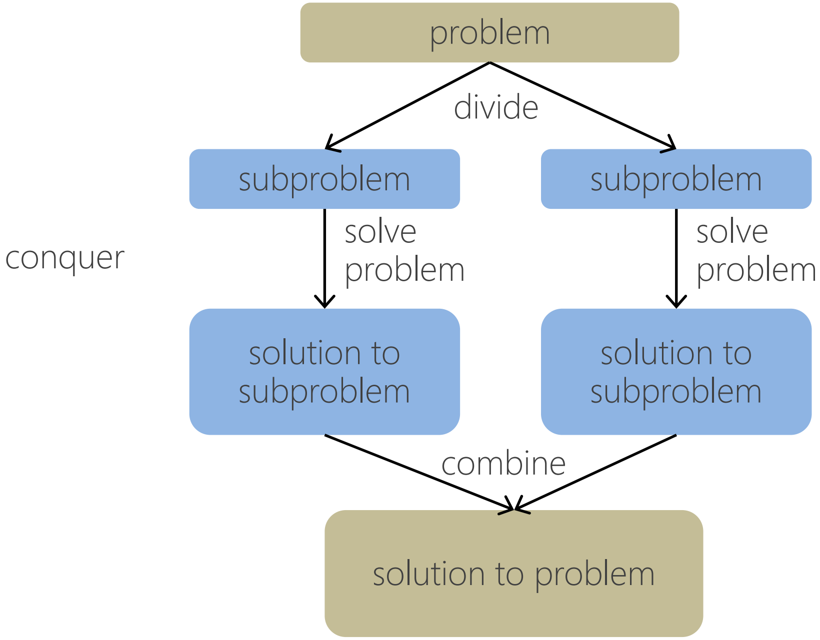
Divide and Conquer

If the problem is large, break it into sub-problems that are smaller in size but are similar in structure to the original problem, recursively solve the sub-problems, and finally combine the sub-solutions into a final solution that solves the original problem.

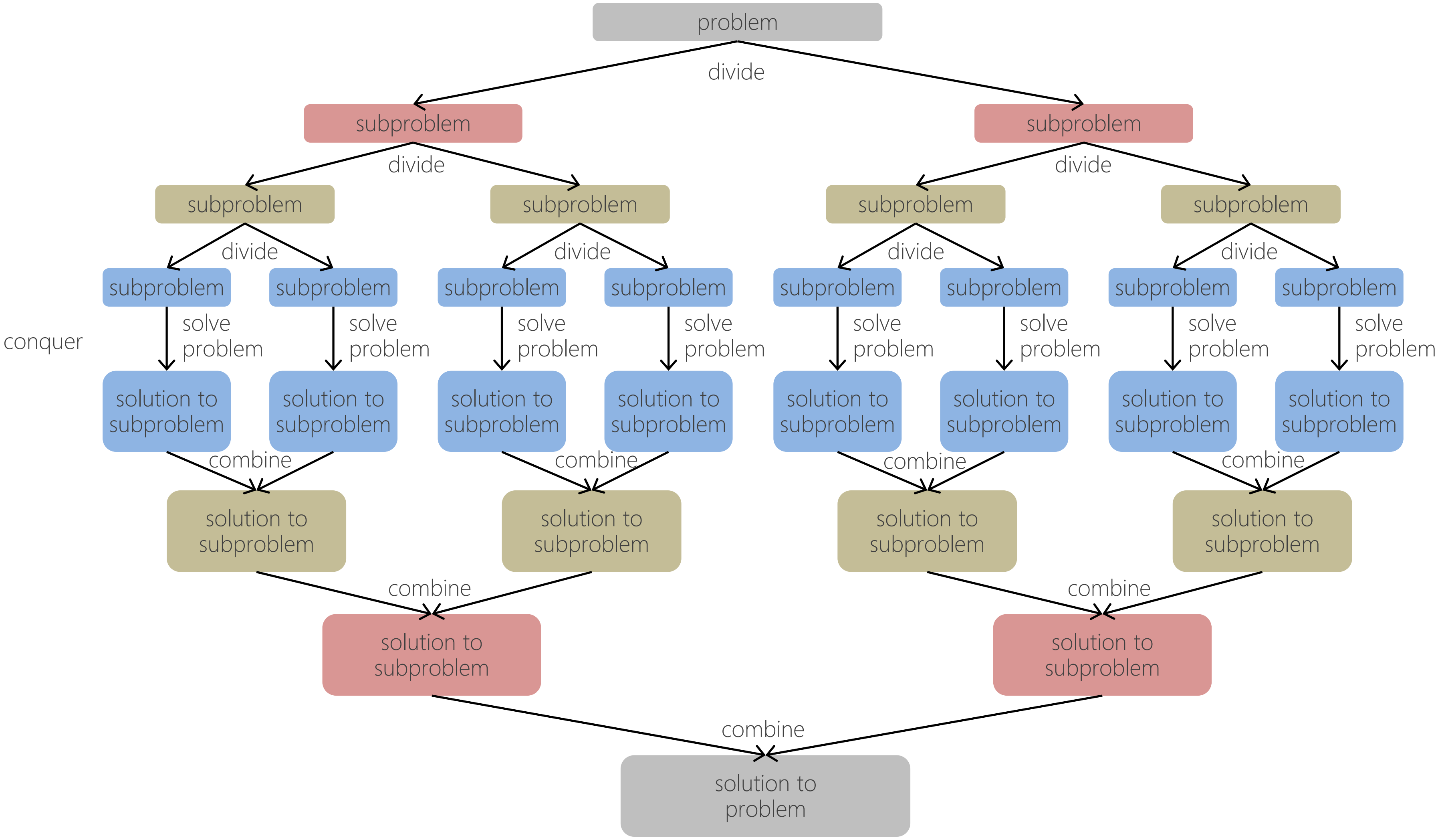
Three Phases of DC

- Divide: top → bottom
  - Divide a problem into sub-problems
- Conquer: bottom level
  - Solve the sub-problems recursively
  - If the sub-problems are small enough, solve them as base cases
- Combine: bottom → top
  - Combine the solutions to the sub-problems into that of the original problem
  - Usually the key!

Divide-Conquer-Combine



Bigger Divide-Conquer-Combine



Task

- Submit T6.cpp to iSpace which includes at least three functions:
  - void InsertionSort(int *A, int n)
    - *A* is an array of integers and *n* is the size of *A*
    - Sort *A* using insertion sort
  - void MergeSort(int *A, int left, int right)
    - Sort sub-array *A*[*left..right*] using merge sort
  - int main(void)
    - Generate an array, A1, consisting of 10^5 random integers
      - Note: Use malloc to claim an array
    - Generate another array *A2* which is identical to *A1*
    - Sort *A1* using InsertionSort() and *A2* using MergeSort()
    - Print the elapsed time in milliseconds during which both search functions run, respectively