CS2020
Data Structures and Algorithms

Welcome!

Administrativia

Discussion Groups:

- Tentative list up. Discussion at break.
- Problems for next week released on IVLE.

Tutorial:

- Today: Document Distance details / Java
  - 2pm – 3pm
  - 3pm – 4pm
  - 4pm – 5pm
- Choose any one!

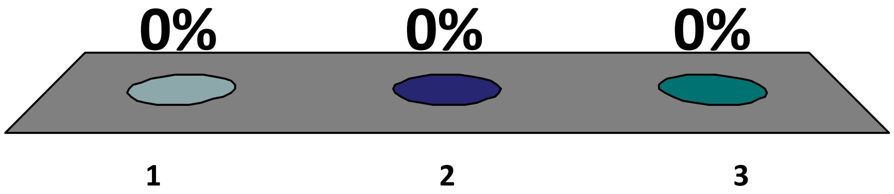
Administrativia

IVLE Forum

- Hints on getting Eclipse running
- Tips on using Eclipse
- Active discussion of DocumentDistance performance.

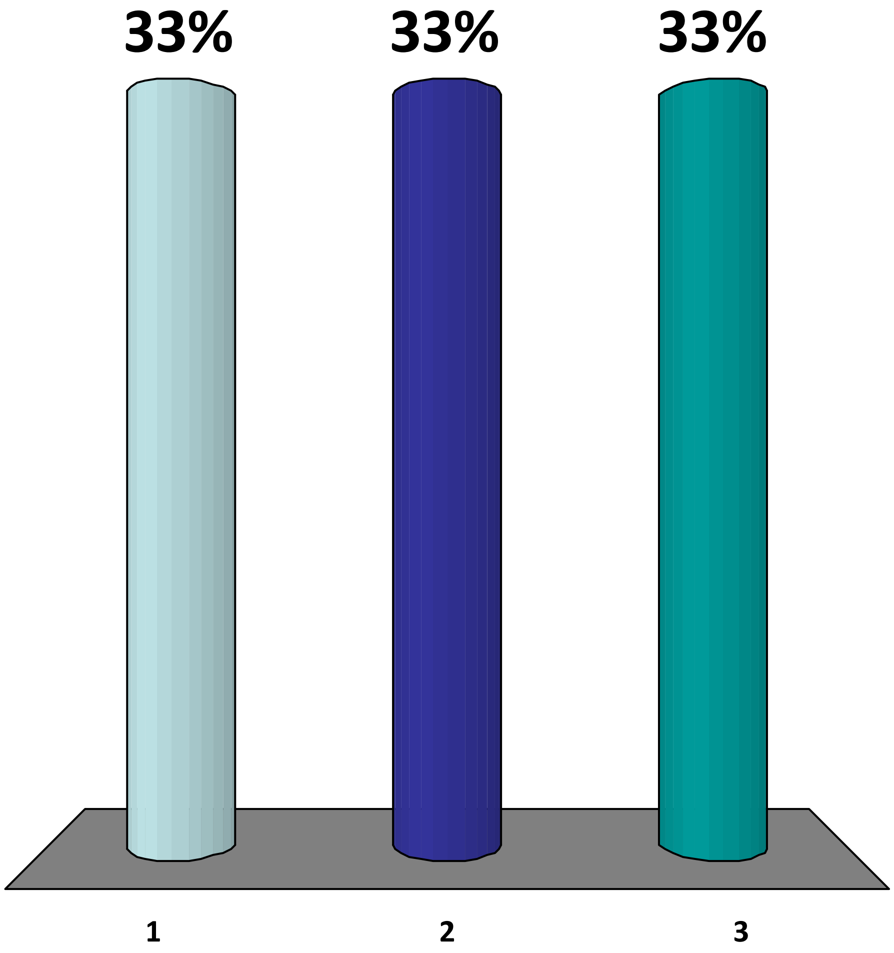
I remembered to bring my clicker to class?

1. Yes
2. No
3. Abstain



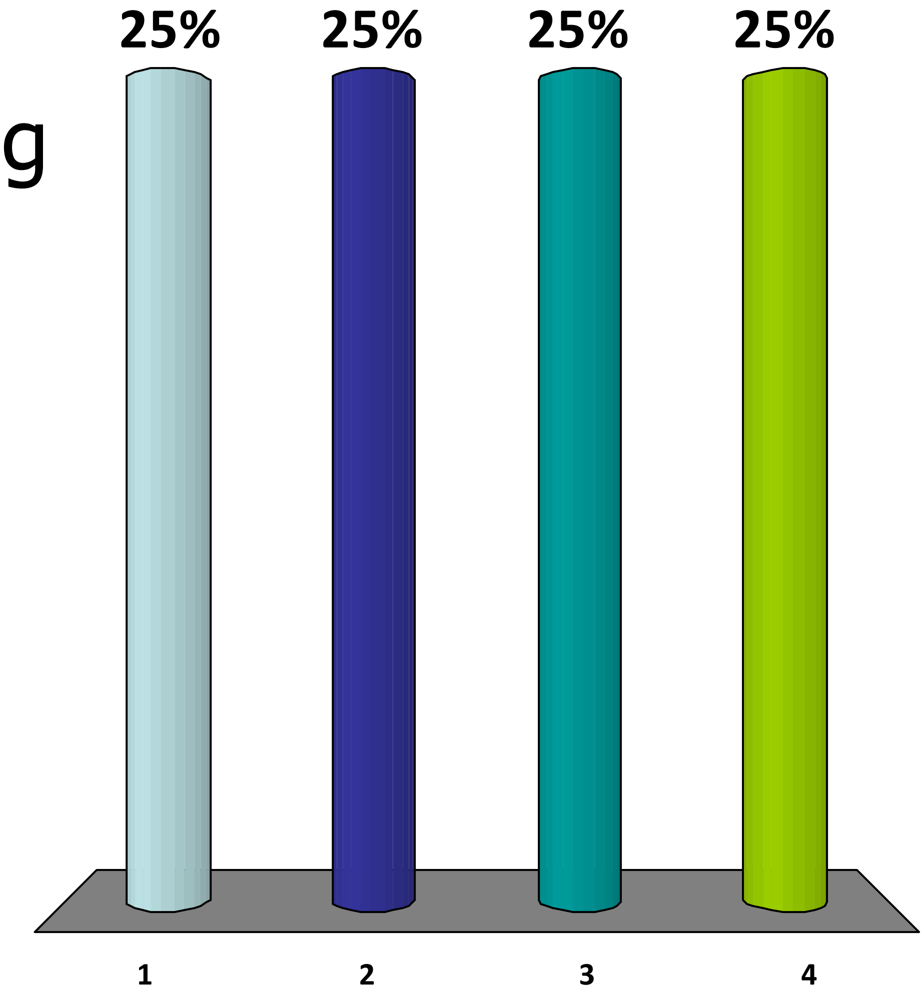
Have you registered at: cs2020.ddns.comp.nus.edu.sg?

1. Yes
2. No
3. I tried, but failed.



Have you successfully gotten Eclipse running?

1. Yes
2. Yes, but no profiling
3. Sort of
4. No

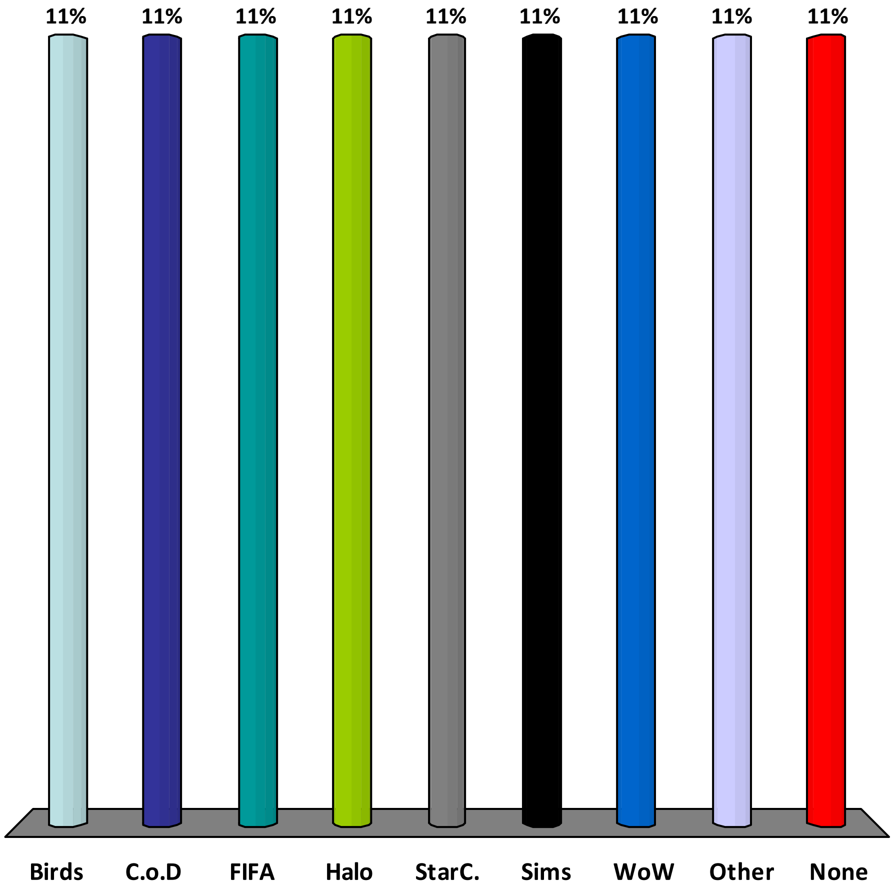


Which 2011 film are you most anticipating?

11%	1. The Green Hornet
11%	2. Jane Eyre
11%	3. Pirates of the Caribbean 4
11%	4. X-Men: First Class
11%	5. Transformers 3
11%	6. Harry Potter 6B
11%	7. Twilight Saga: Breaking Down
11%	8. Scream 4
11%	9. Happy Feet 2

What is your favorite video game?

1. Angry Birds
2. Call of Duty
3. FIFA
4. Halo
5. Starcraft
6. The Sims
7. World of Warcraft
8. Other
9. I don’t play video games.



Today

- Document Distance Implementation
  - Java intro
  - Object-oriented programming

- Sorting
  - Insertion Sort
  - Merge Sort

Programming Paradigms

Models of programming:

- Procedural (imperative) languages
- Functional languages
- Declarative languages
- Object-oriented languages

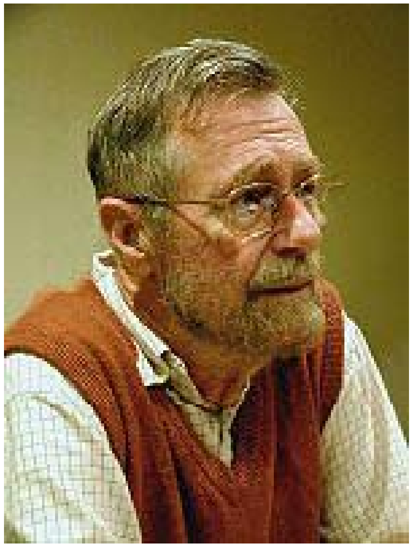
How to organize information?

How to think about a solution?

Programming Paradigms

Procedural Languages

- Examples:
  - Fortran, COBOL, BASIC, Pascal, C
- Organization:
  - Group instructions into “procedures” or “functions”
  - Each procedure modifies the **state**.
  - Don’t use GOTO statement (see)



- Advantages:
  - Readability
  - Procedure re-use

Programming Paradigms

Functional Languages

- Examples:
  - Scheme, Lisp
- Organization:
  - Everything is a function
  - Output depends only on input
  - No state, no mutable data
- Advantages:
  - Simplicity, elegance
  - Describe what you are doing with *verbs*.
  - Focus on computation, not data manipulation

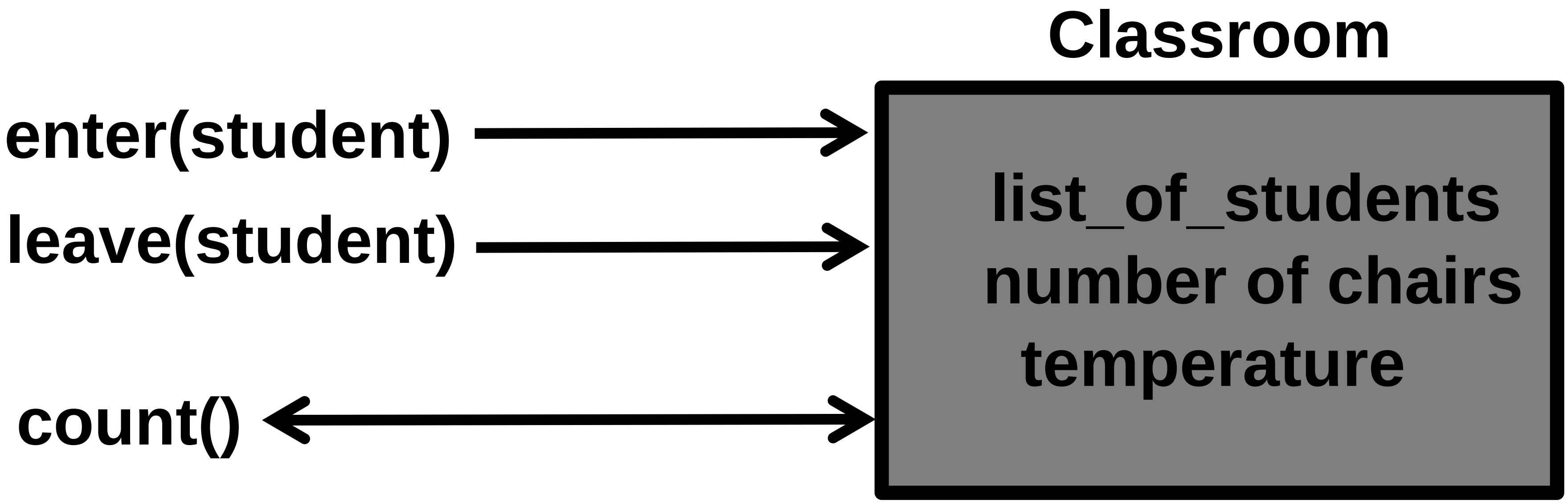
Programming Paradigms

- Object-oriented Languages
  - Examples:
    - Java, C++
  - Advantages:
    - Near-ubiquitous in industry
    - Modular
    - Code re-use
    - Easier to iterate / develop new versions
      - Information hiding
      - Pluggable

Object-oriented Programming

Object contains:

- State (i.e., data)
- Behavior (i.e., methods for modifying the state)



Classroom

enter(student)

leave(student)

count()

list_of_students
number of chairs
temperature

How to implement a **File System**?

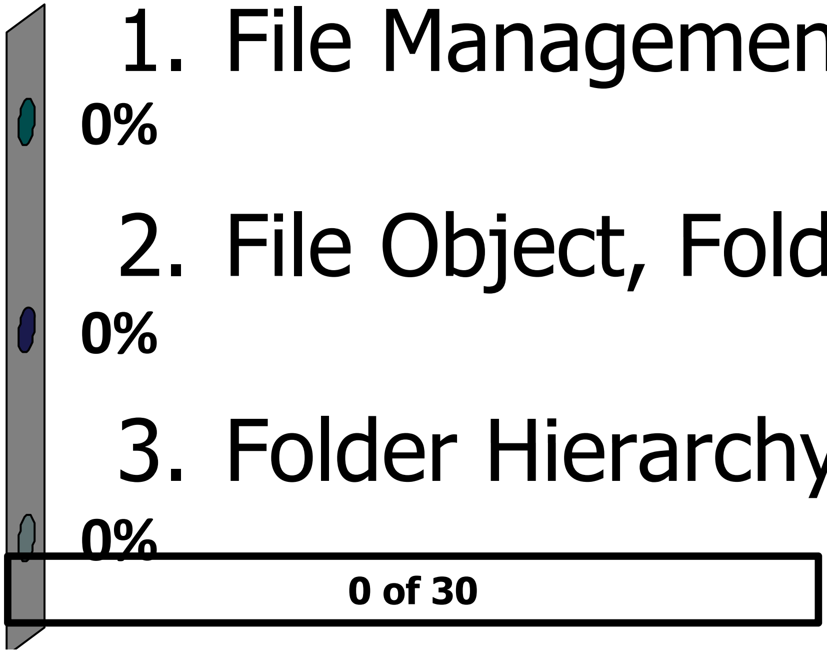
<u>Files</u>:

- Contain data
- Edited
- Rename
- Moved

<u>Folders</u>:

- Contain files
- Contain folders
- Rename
- Moved

1. File Management Obj, FileContents Obj
2. File Object, Folder Objects
3. Folder Hierarchy Obj, FolderContents Obj.



Objected-Oriented Java

```
class File
{
  String name;
  FileData contents;

  void rename(String newName){ name=newName; }
  FileData getData(){ return contents;}
  void setData(FileData newdata){ contents = newdata;}
}
```

Objected-Oriented Java

```
class Folder
{
    String name;
    Folder[] children;
    File[] files;

    int getNumFiles(){ …}
    File getFile(int ii){ … }
    …
}
```

Access Control

- (none specified)
  - within the same package
- public
  - everywhere
- private:
  - only in the same class
- protected:
  - within the same package, and by subclasses

(Recommended) Access Control

- State (i.e., variables):
  - private: class encapsulates state

- Behavior (i.e., methods):
  - public: class makes functionality available
  - private/protected: class uses certain functionality internally

Object-oriented Java

Creating and using objects:

```
Folder root = new Folder();
File homework = new File (“hw-one.txt”);
root.addfile(homework);
```

Constructors: Initialize new objects

```
class File
{
   String filename;

   File(String name)
   {
      filename = name;
   }
   …
}
```

Object-oriented Java

Creating and using objects:

```
Folder root = new Folder();
File homework = new File (“hw-one.txt”);
root.addfile(homework);
```

Document Distance

Basic object/class:

VectorTextFile

Functionality:

- Reads in file
- Norm of vector (i.e., file)
- Dot-product of two vectors (i.e., files)
- Angle between two vectors (i.e., files)

Document Distance

Basic object/class:

VectorTextFile

Constructor: (given: filename)

- Reads in file
- Parses file into words
- Sorts words
- Counts word frequencies

Document Distance

Basic object/class:

- VectorTextFile

Public functionality:

- double norm()
- int DotProduct(VectorTextFile A, VectorTextFile B)
- double Angle(VectorTextFile A, VectorTextFile B)

All other functionality is private / internal!

Document Distance

Seconary object/class: WordCountPair

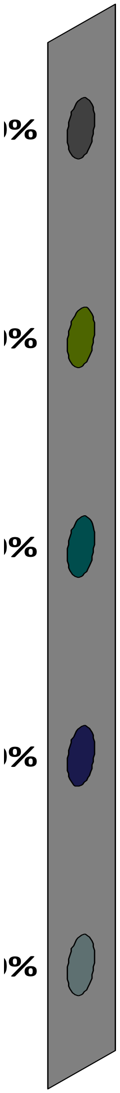
Encapsulates:

- String word
- int count

Functionality:

- Constructor: sets word and counts
- getWord()
- getCount()

I found the VectorTextFile class:



1. Easy to understand.
2. Make sense, but many confusing details….
3. The Java syntax confuses me.
4. I don’t understand the problem / vectors.
5. I haven’t yet looked at it.

Performance Profiling, V2

(*Dracula* vs. *Lewis & Clark*)

Step	Function	Running Time
Create vectors:	Read each file	1.09s
	Parse each file	3.68s
	Sort words in each file	332.13s
	Count word frequencies	0.30s
Dot product:		6.06s
Norm:		3.80s
Angle:		6.06s
Total:		**11minutes ≈ 680.49s**

Sorting

Problem definition:

Input: array A[1..n] of words / numbers

Output: array B[1..n] that is a permutation of A such that:

B[1] ≤ B[2] ≤ … ≤ B[n]

Example:

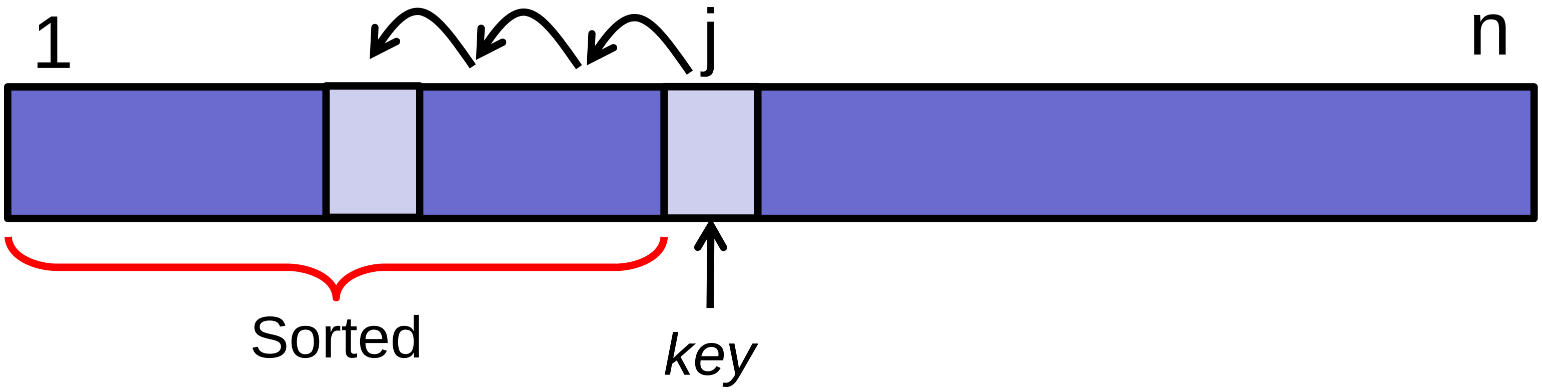
A = [9, 3, 6, 6, 6, 4] → [3, 4, 6, 6, 6, 9]

Insertion Sort

```
Insertion-Sort(A, n)
    for j ← 2 to n
        key ← A[j]
        Insert key into the sorted array A[1..j-1]
```

Invariant: A[1..j-1] is sorted

Illustration:



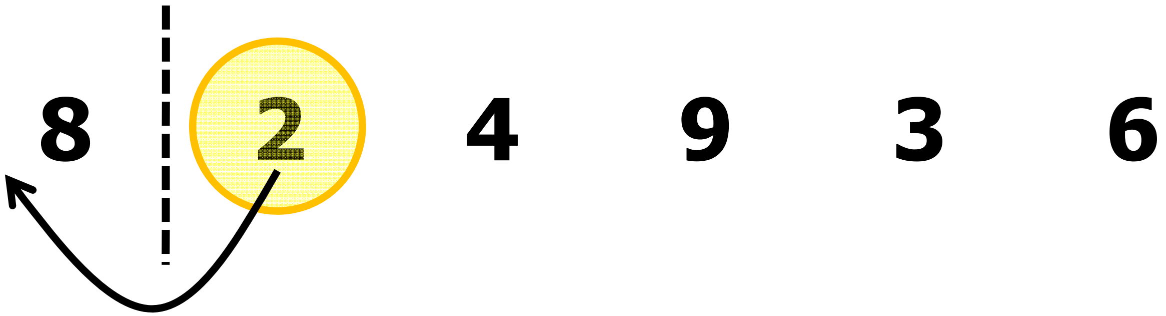
Insertion Sort

```
Insertion-Sort(A, n)
    for j ← 2 to n
        key ← A[j]
        i ← j-1
        while (i > 0) and (A[i] > key)
            A[i+1] ← A[i]
            i ← i-1
        A[i+1] ← key
```

Invariant: A[1..j-1] is sorted

Insertion Sort

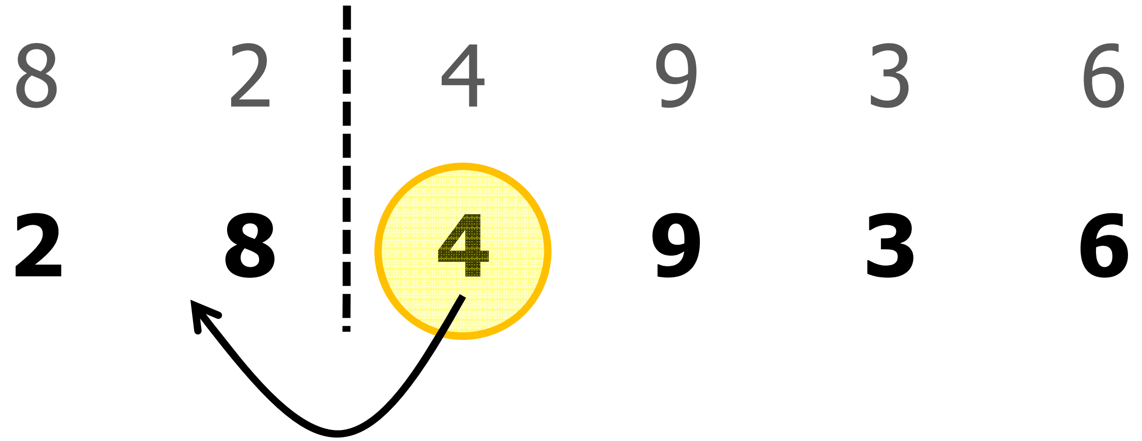
Example:



8 2 4 9 3 6

Insertion Sort

Example:

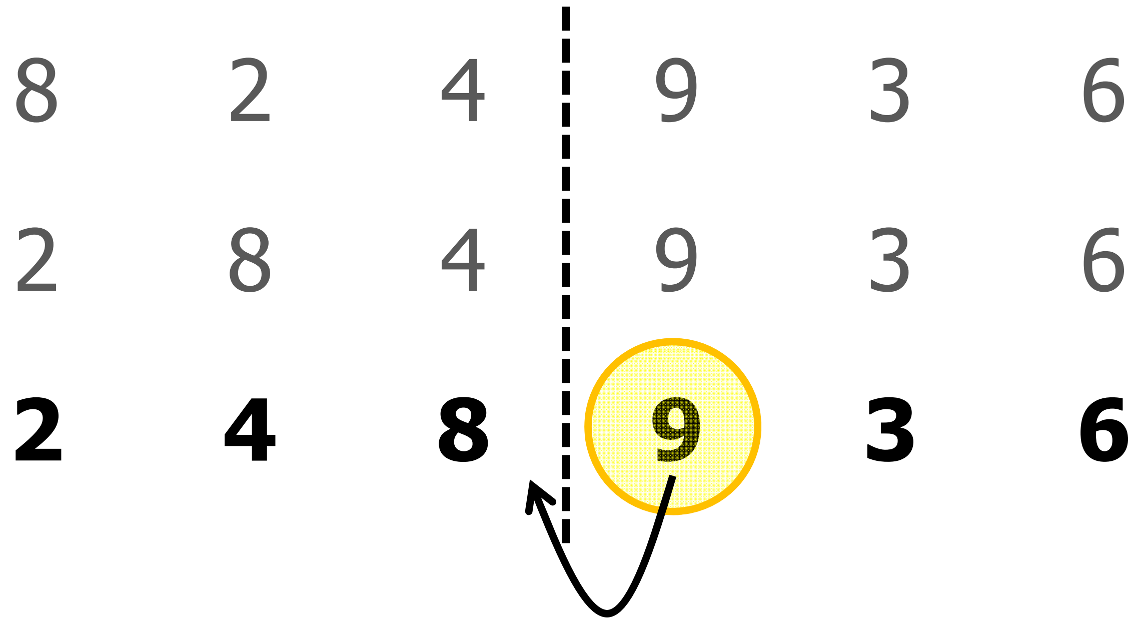


8 2 4 9 3 6

2 8 4 9 3 6

Insertion Sort

Example:



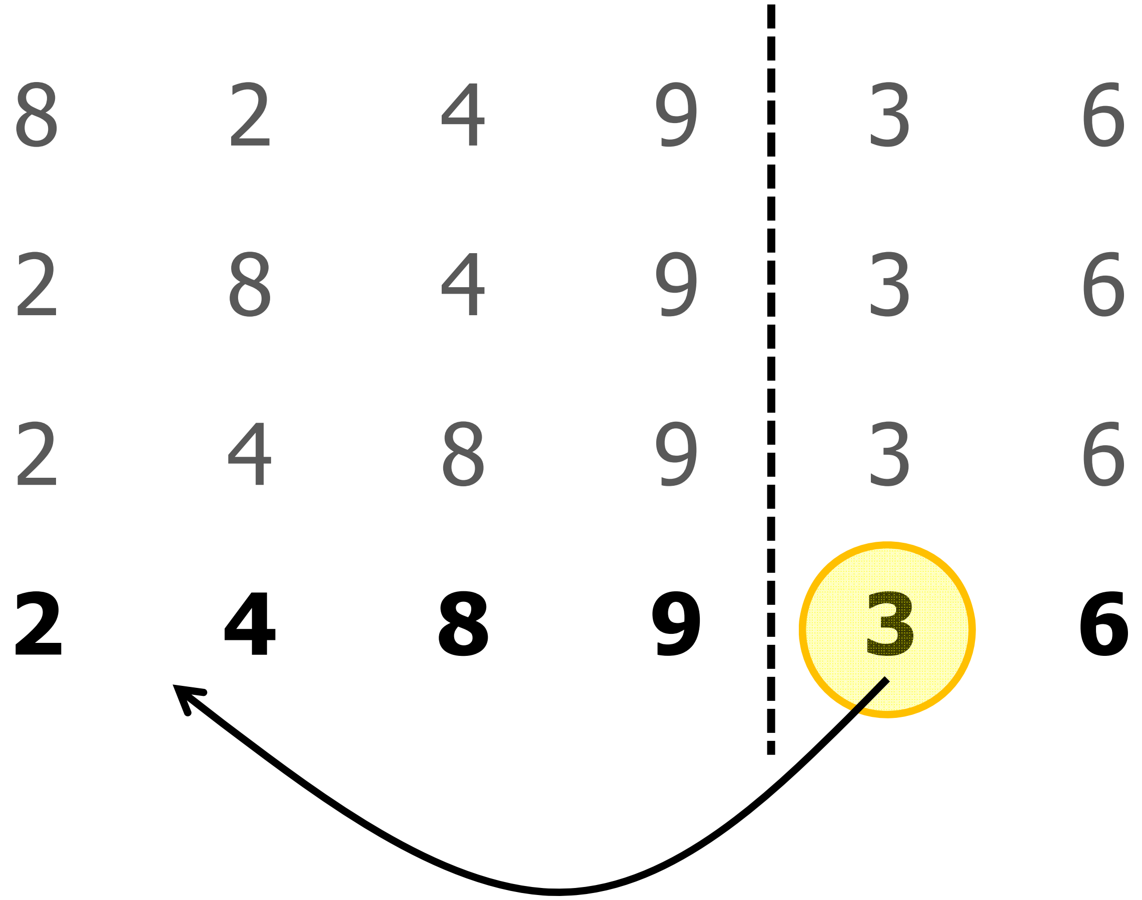
8 2 4 | 9 3 6

2 8 4 | 9 3 6

2 4 8 | 9 3 6

Insertion Sort

Example:



8 2 4 9 3 6

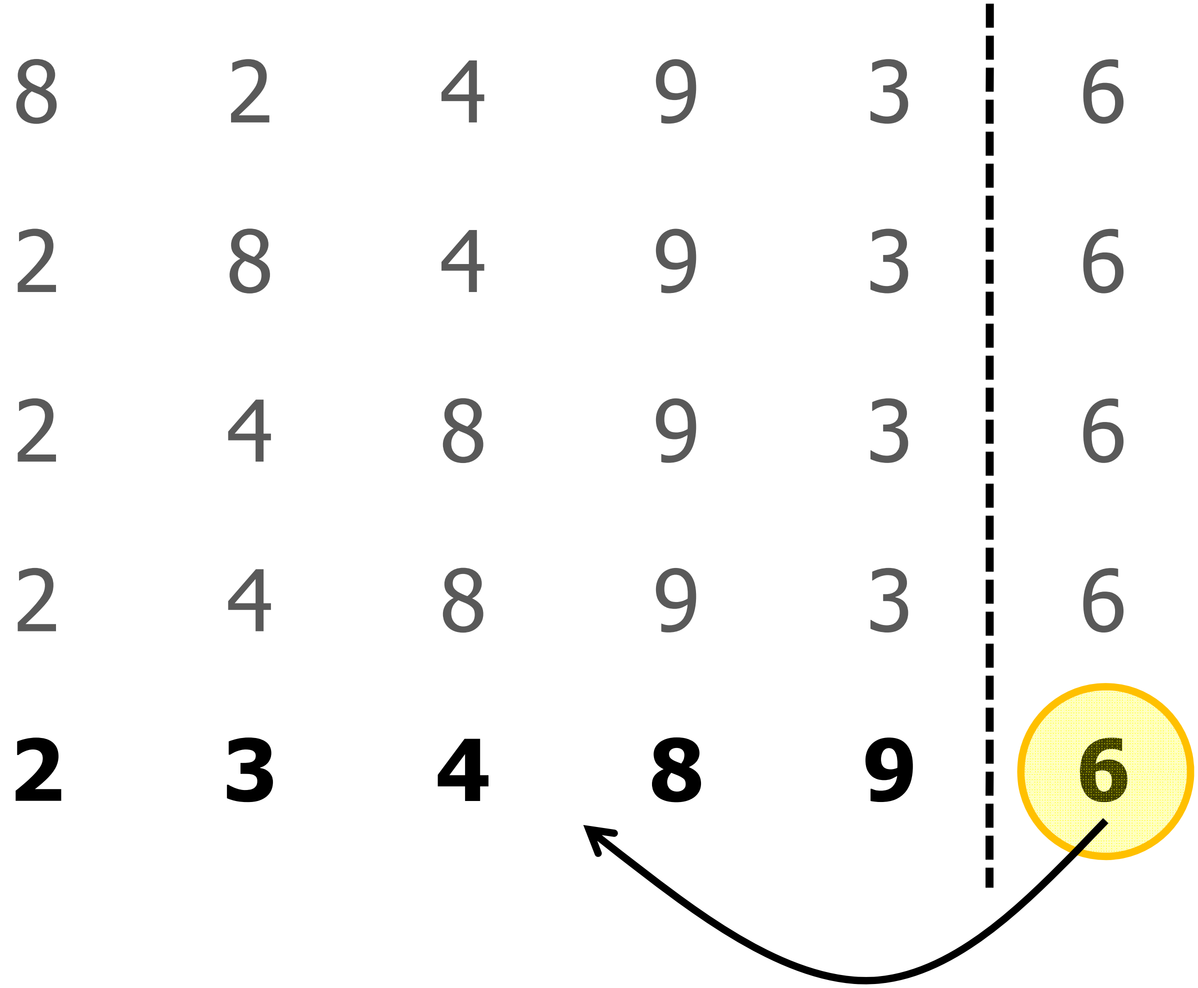
2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

Insertion Sort

Example:

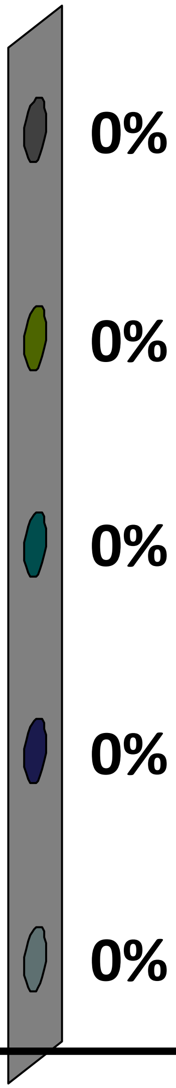


8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6
2	**3**	**4**	**8**	**9**	**6**

Insertion Sort

Example:	8	2	4	9	3	6
	2	8	4	9	3	6
	2	4	8	9	3	6
	2	4	8	9	3	6
	2	3	4	8	9	6
	2	**3**	**4**	**6**	**8**	**9**

What is the running time of Insertion Sort?



1. O(n)
2. O(n log n)
3. O(n√n)
4. O(n^2)
5. O(2^n)

0 of 30

Insertion Sort

Running time:

- Depends on the input!

Best-case:

- Already sorted: O(n)

Insertion Sort

Running time:

- Depends on the input!

Best-case:

- Already sorted: O(n)

Average-case:

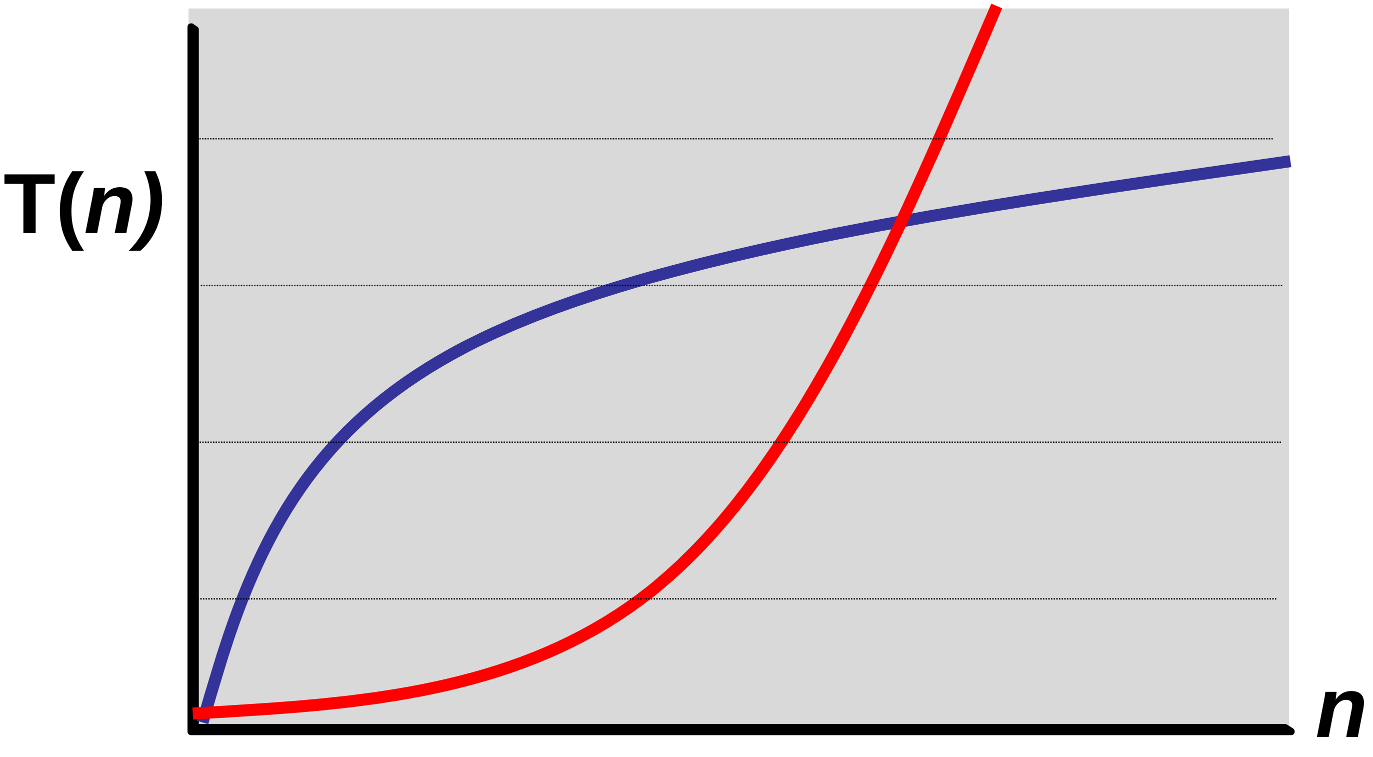
- Assume inputs are chosen at random…

Worst-case:

- Bound on how long it takes.

Big-O Notation

- How does an algorithm scale?
  - For large inputs, what is the running time?
  - T(n) = running time on inputs of size n



Big-O Notation

Definition:

$T(n) = O(f(n))$ if and only if:

- there exists a constant c
- there exists a constant n_0

for all $n > n_0$:

$$T(n) < cf(n)$$

Big-O Notation

Example:

$g(n) = 4n^2 + 24n - 16$

$< 100^n \quad (\text{for } n>0)$

$= O(100^n)$

Big-O Notation

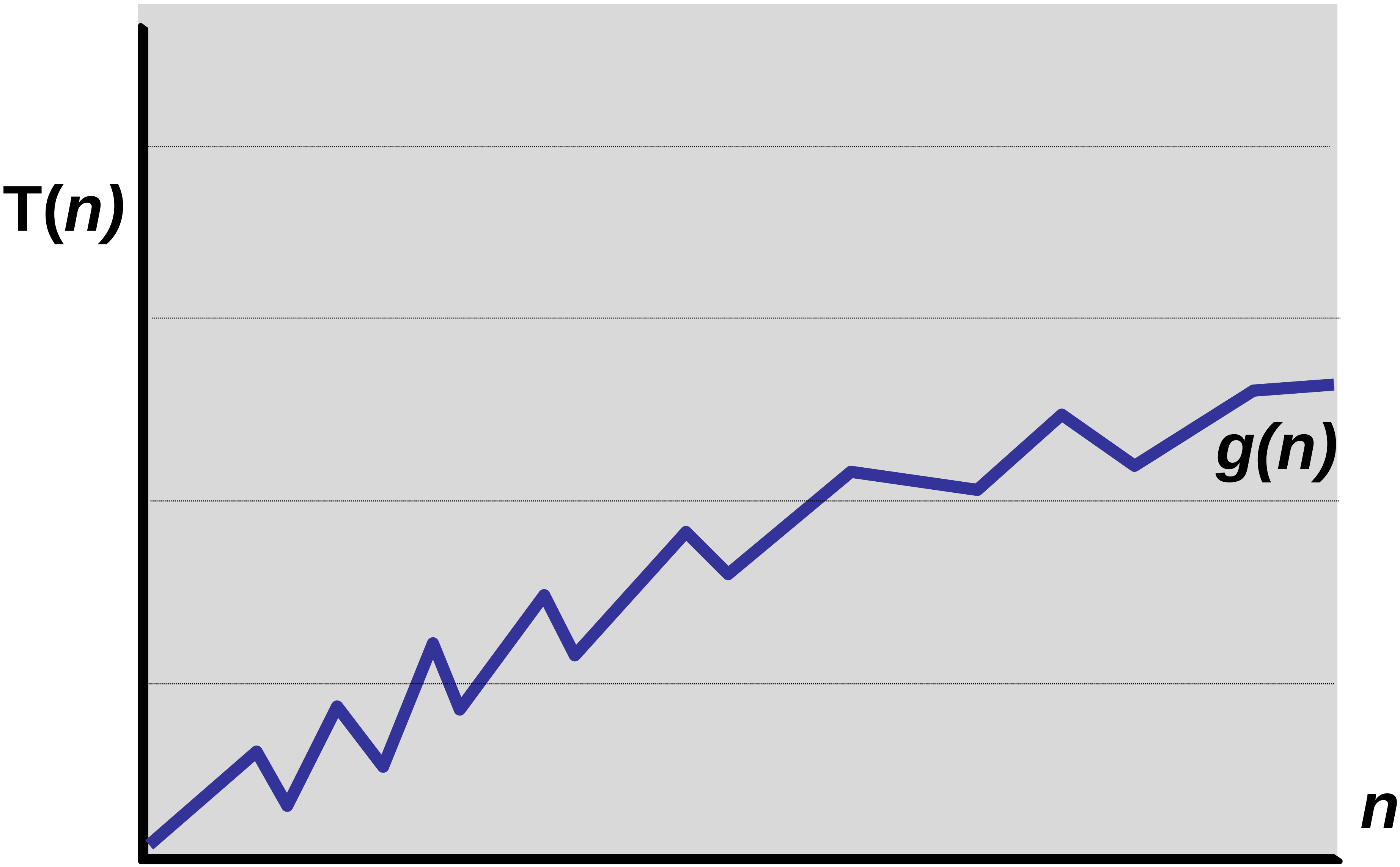
Example:

$g(n) = 4n^2 + 24n - 16$

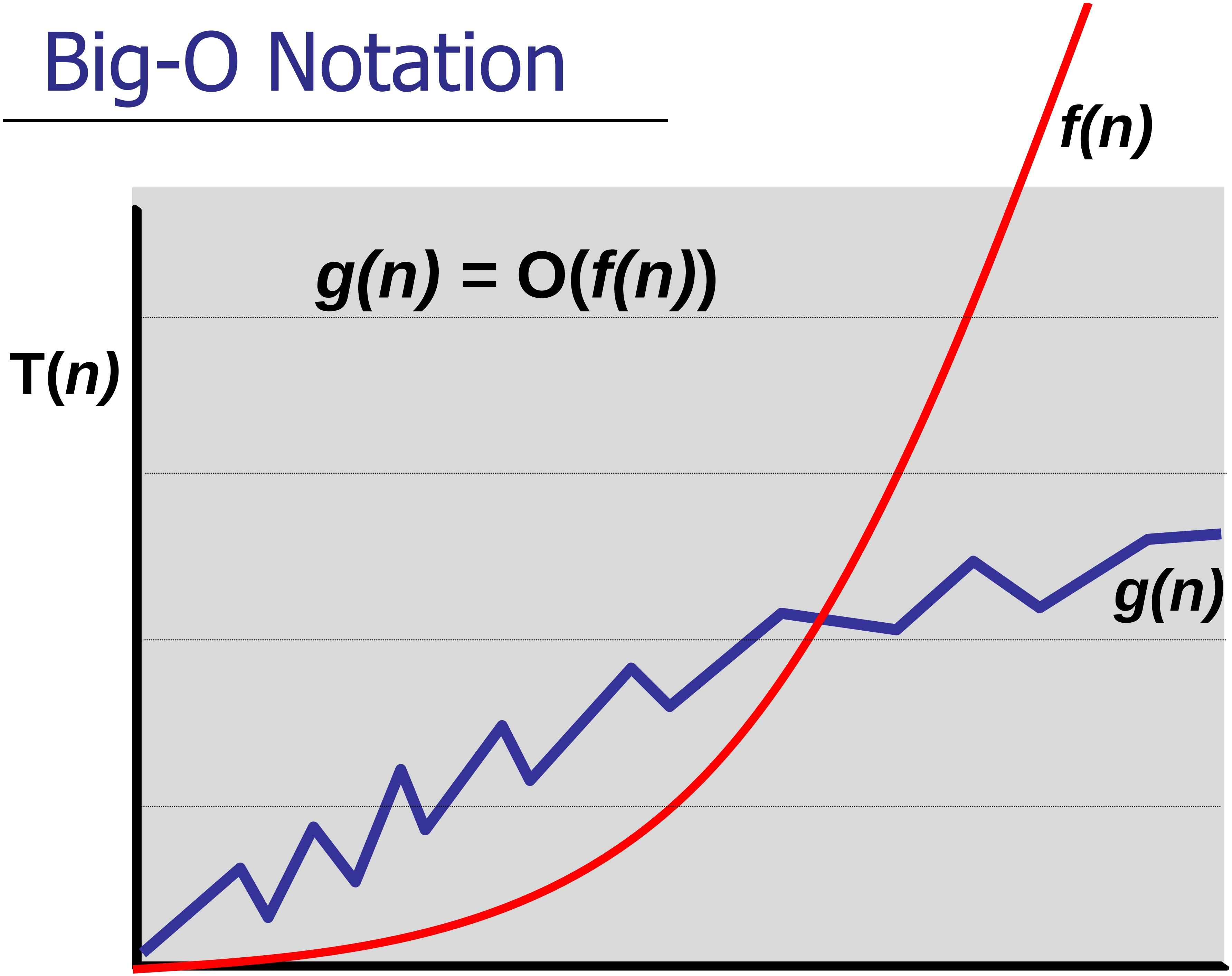
$< 28n^2 \quad (\text{for } n>0)$

$= O(n^2)$

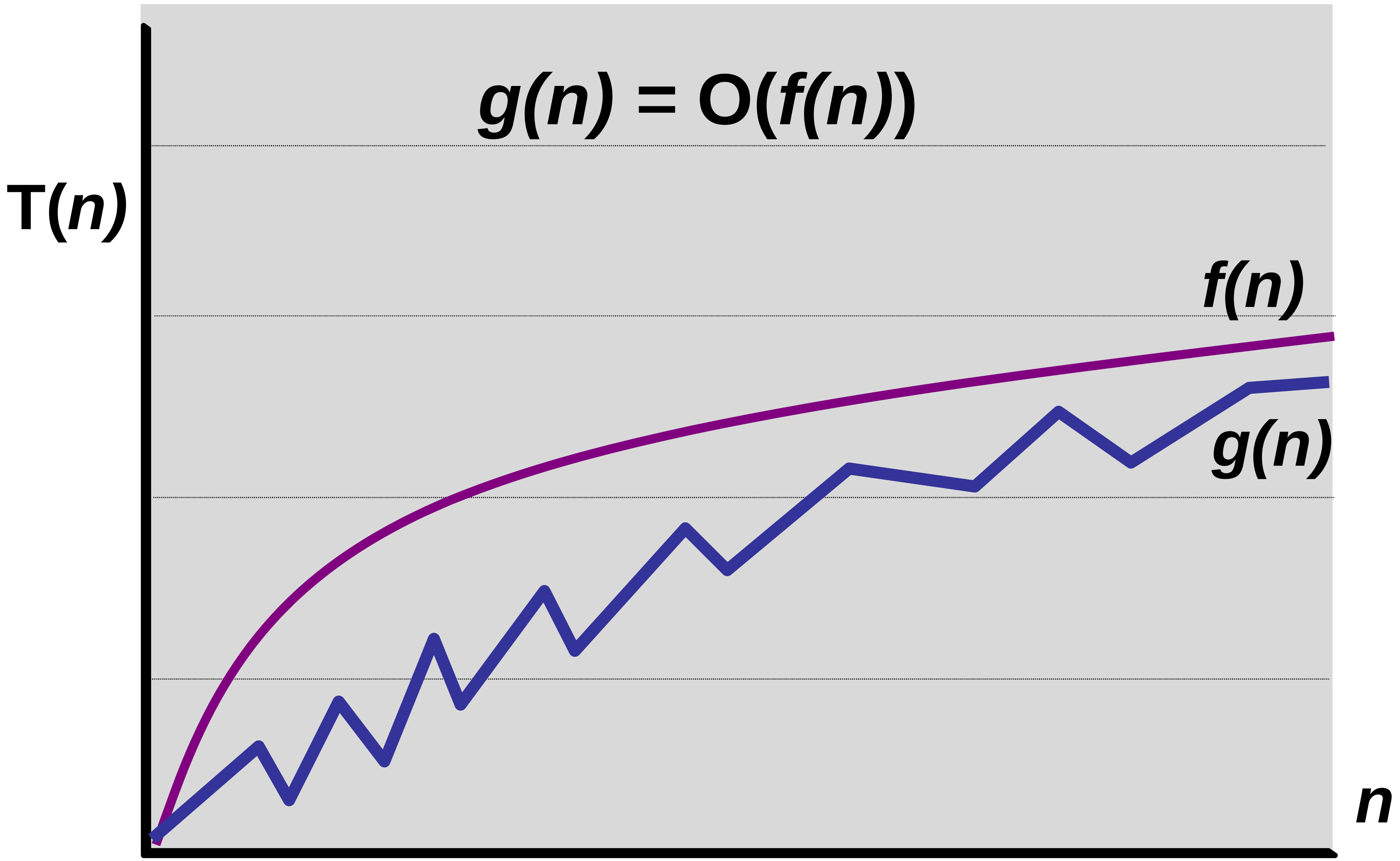
Big-O Notation



Big-O Notation



Big-O Notation



Insertion Sort Analysis

```
Insertion-Sort(A, n)
    for  j ← 2 to n
        key ← A[j]
        i ← j-1
        while (i > 0) and (A[i] > key)
            A[i+1] ← A[i]
            i ← i-1
        A[i+1] ← key
```

Repeat at most j times.

Insertion Sort Analysis

Worst-case: j ← 2 **to** n

2 + 3 + 4 + … + n =

$$\sum_{j=2}^{n} \Theta(j) = \Theta\left(n^{2}\right)$$

Consider: list reverse sorted

[10 9 8 7 6 5 4 3 2 1]

Insertion Sort Analysis

Average-case analysis:

- Assume all inputs equally likely

$$\sum_{j=2}^{n} \Theta\left({}^{j}\!/_{2} \right) = \Theta\left(n^2\right)$$

- In expectation, still θ**(n²)**

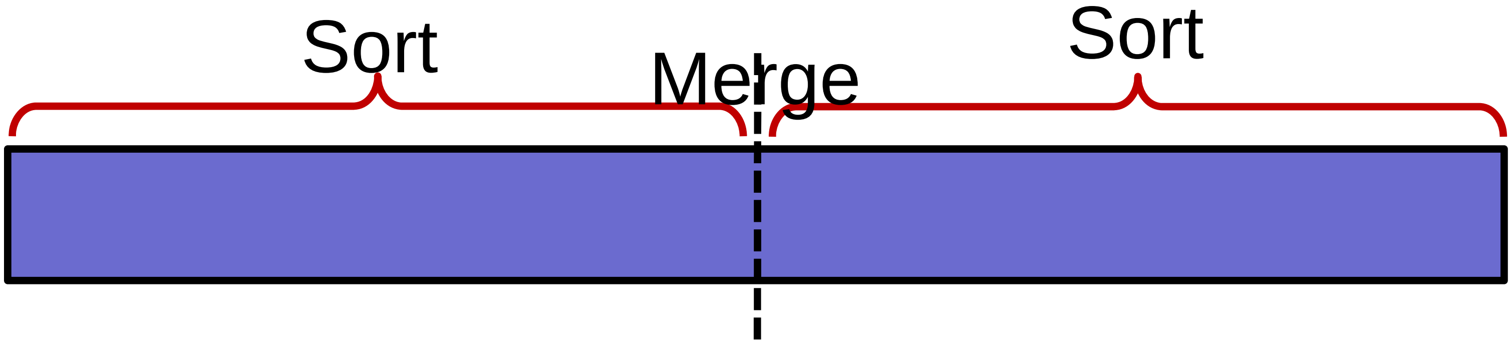
Performance Profiling, V2

(*Dracula* vs. *Lewis & Clark*)

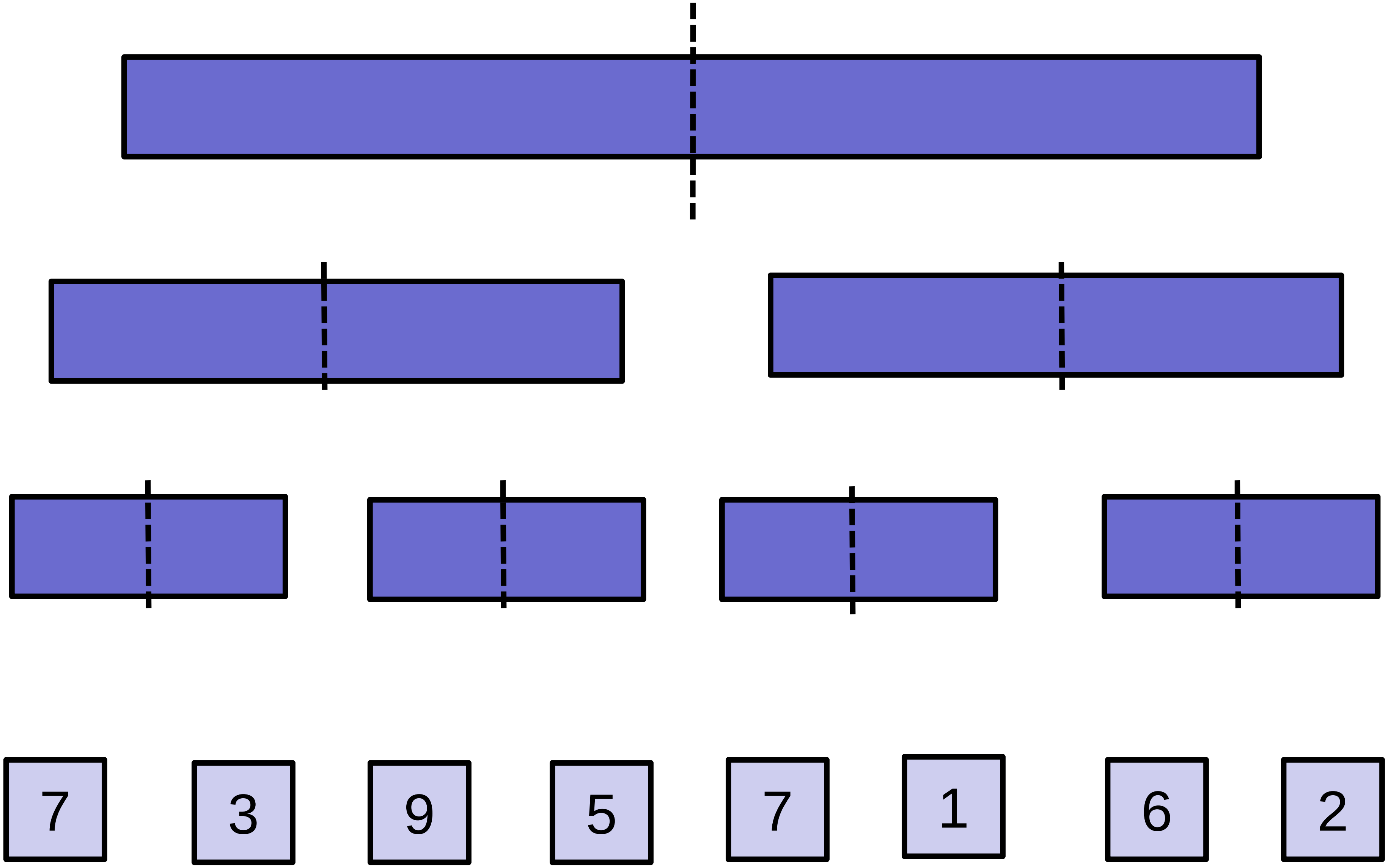
Step	Function	Running Time
Create vectors:	Read each file	1.09s
	Parse each file	3.68s
	Sort words in each file	332.13s
	Count word frequencies	0.30s
Dot product:		6.06s
Norm:		3.80s
Angle:		6.06s
Total:		**11minutes ≈ 680.49s**

Merge-Sort

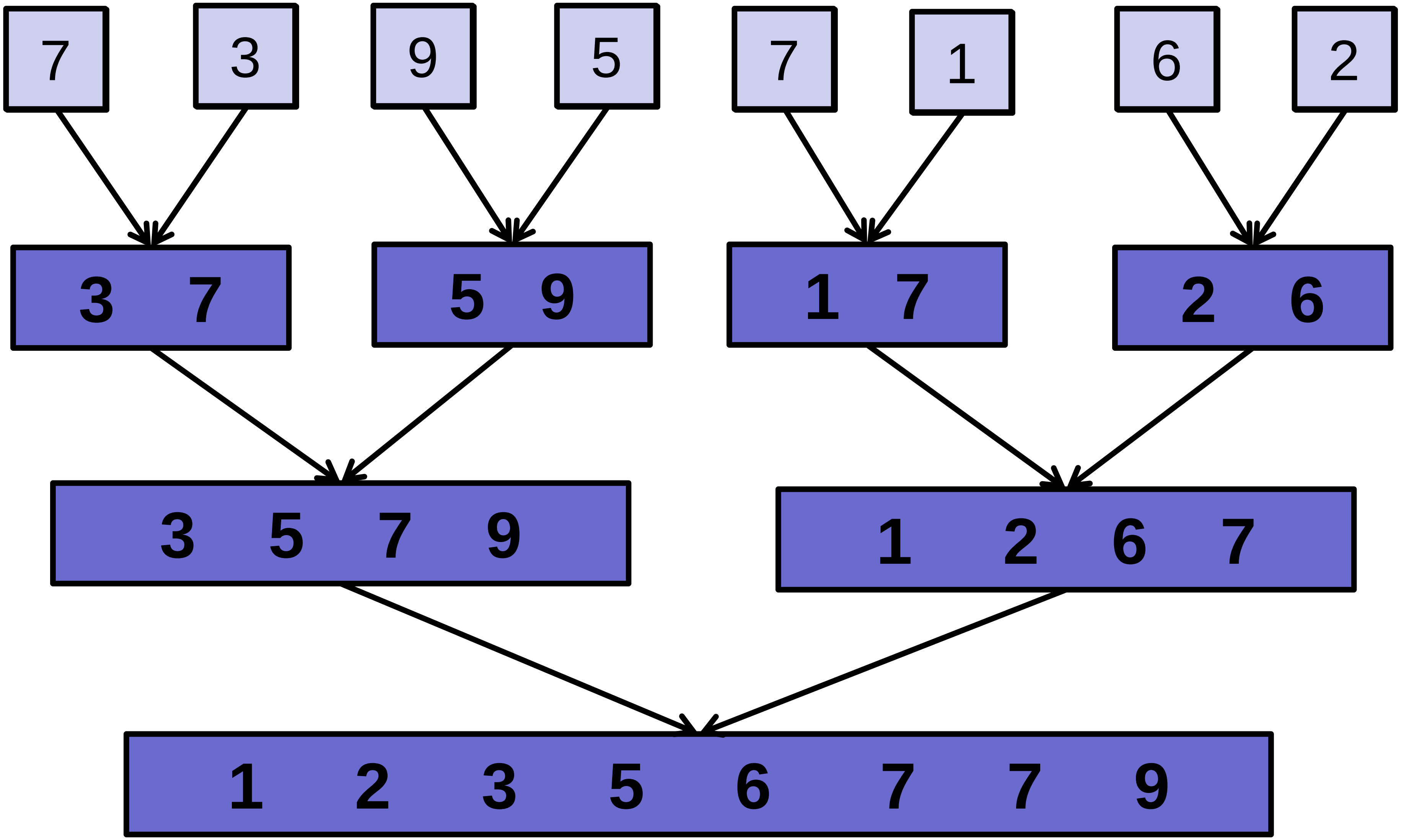
```
Merge-Sort(A, n)
    if (n=1) then return;
    else: Recurse!
        X ← Merge-Sort(A[1..n/2],  n/2);
        Y ← Merge-Sort(A[n/2+1, n],  n/2);
    return Merge (X,Y, n/2);
```



Divide-and-Conquer



Merging



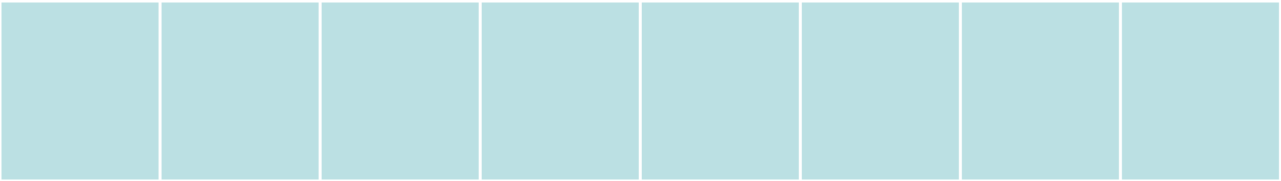
Merging Two Sorted Lists

Key subroutine: Merge

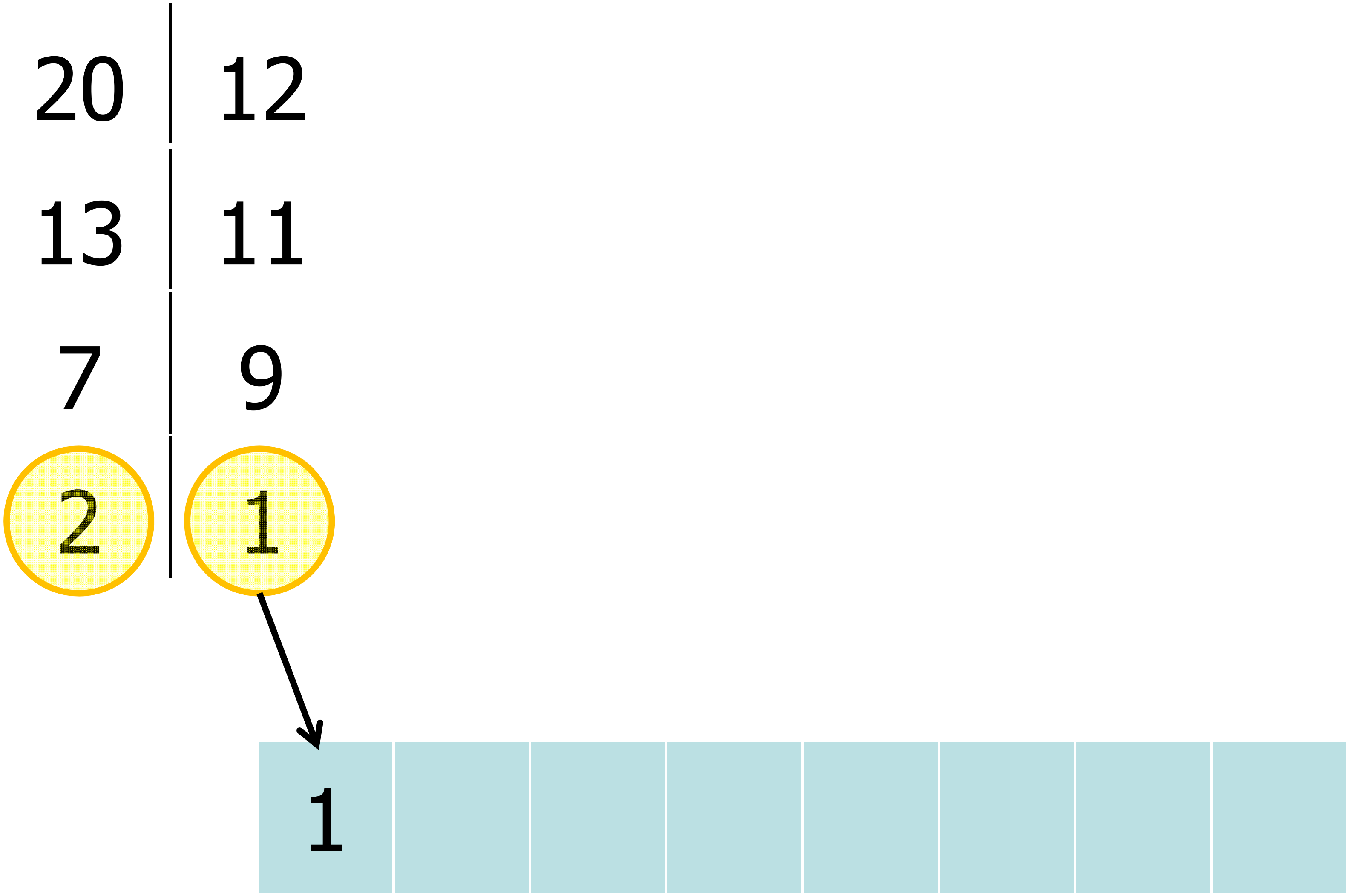
- How?
- How fast??

Merging Two Sorted Lists

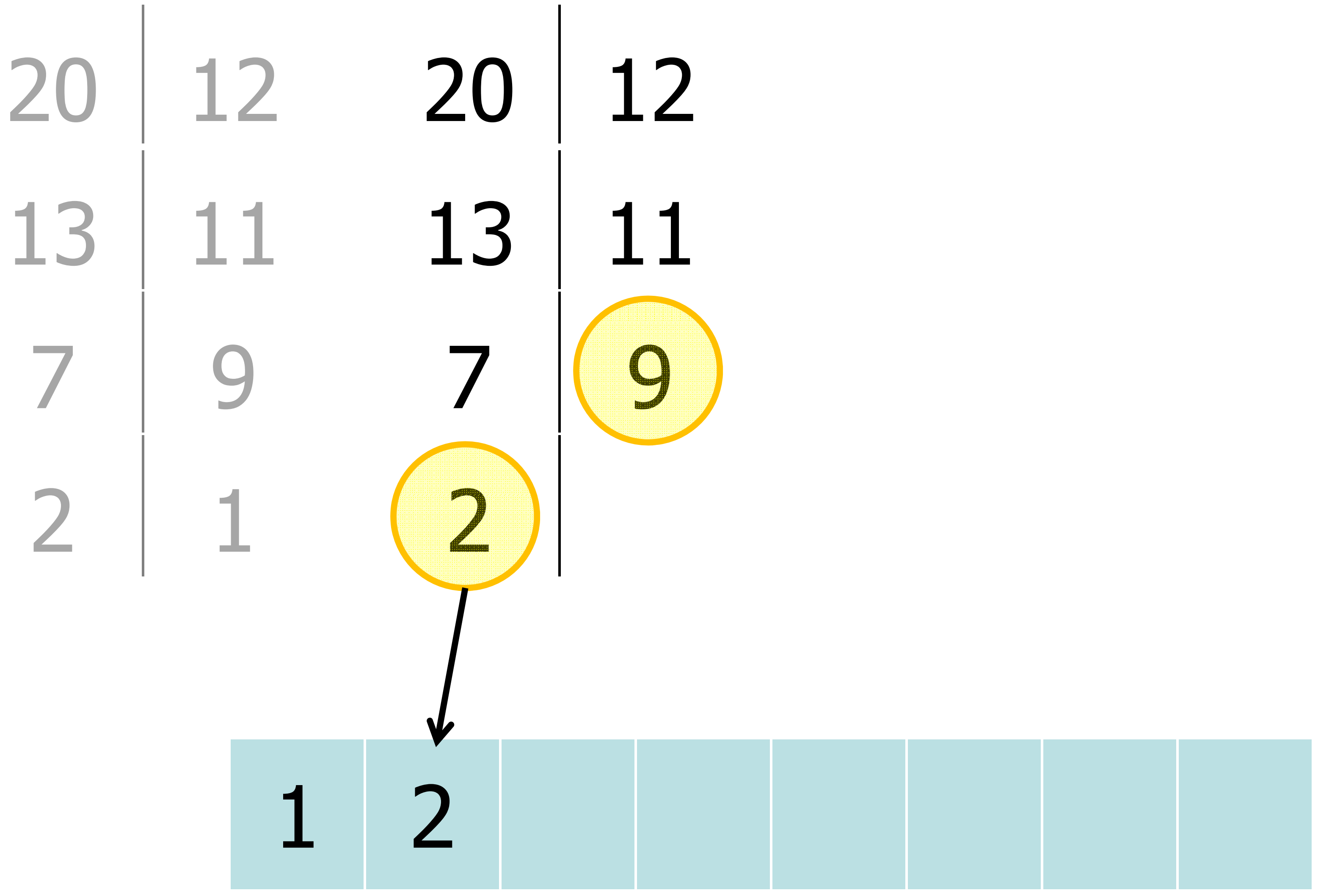
20	12
13	11
7	9
2	1



Merging Two Sorted Lists



Merging Two Sorted Lists



20 | 12 20 | 12

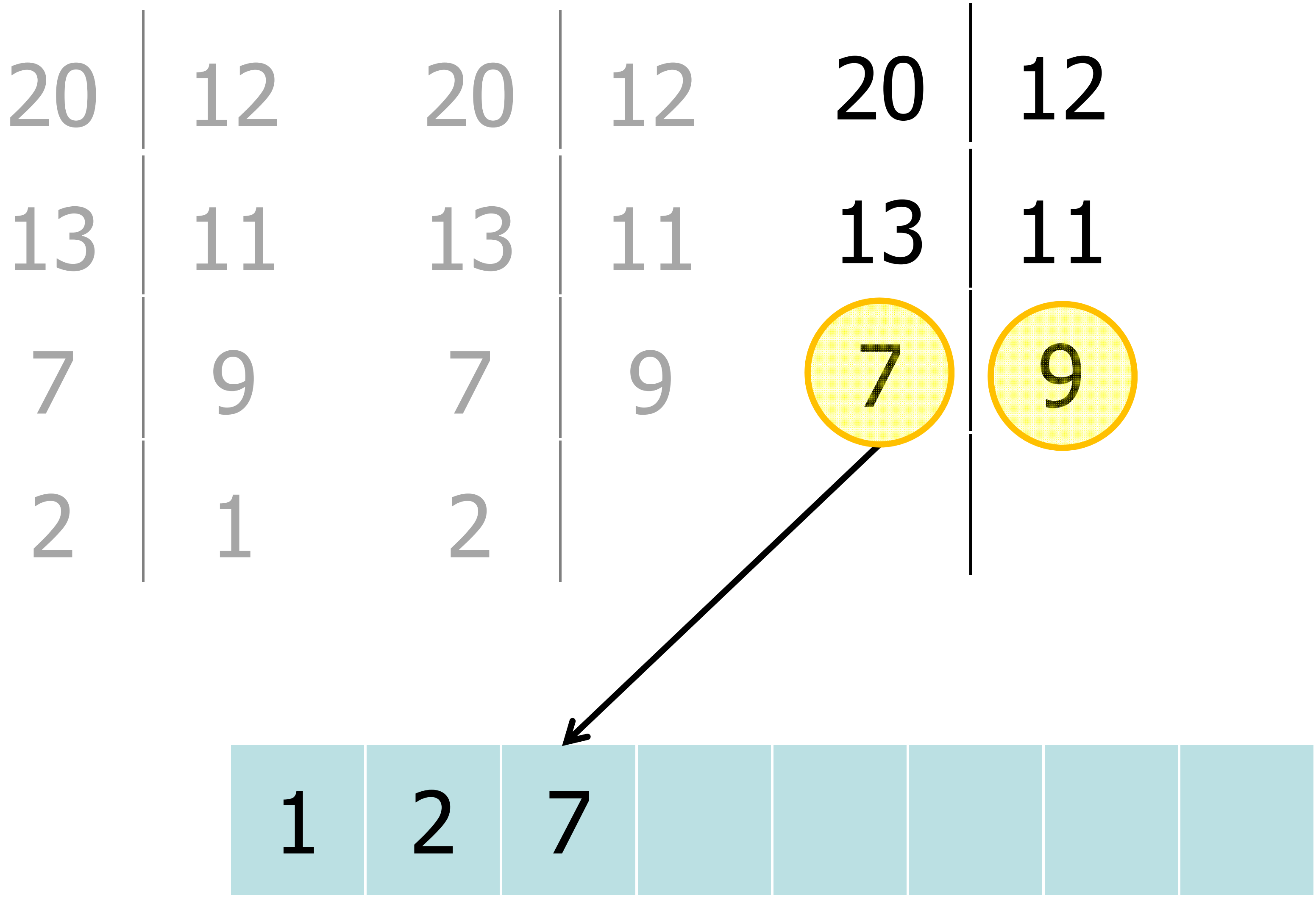
13 | 11 13 | 11

7 | 9 7 | 9

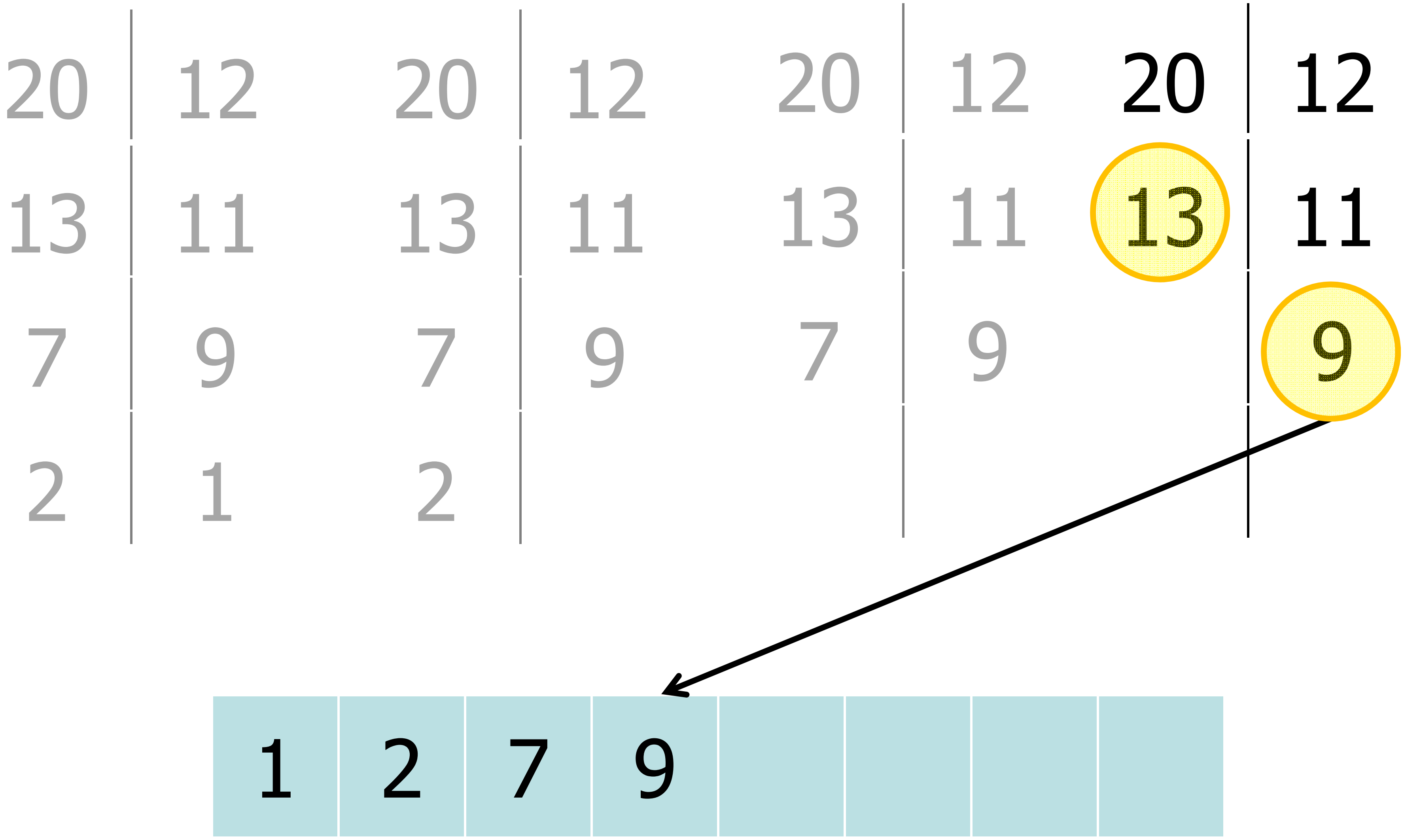
2 | 1 2 |

1 2

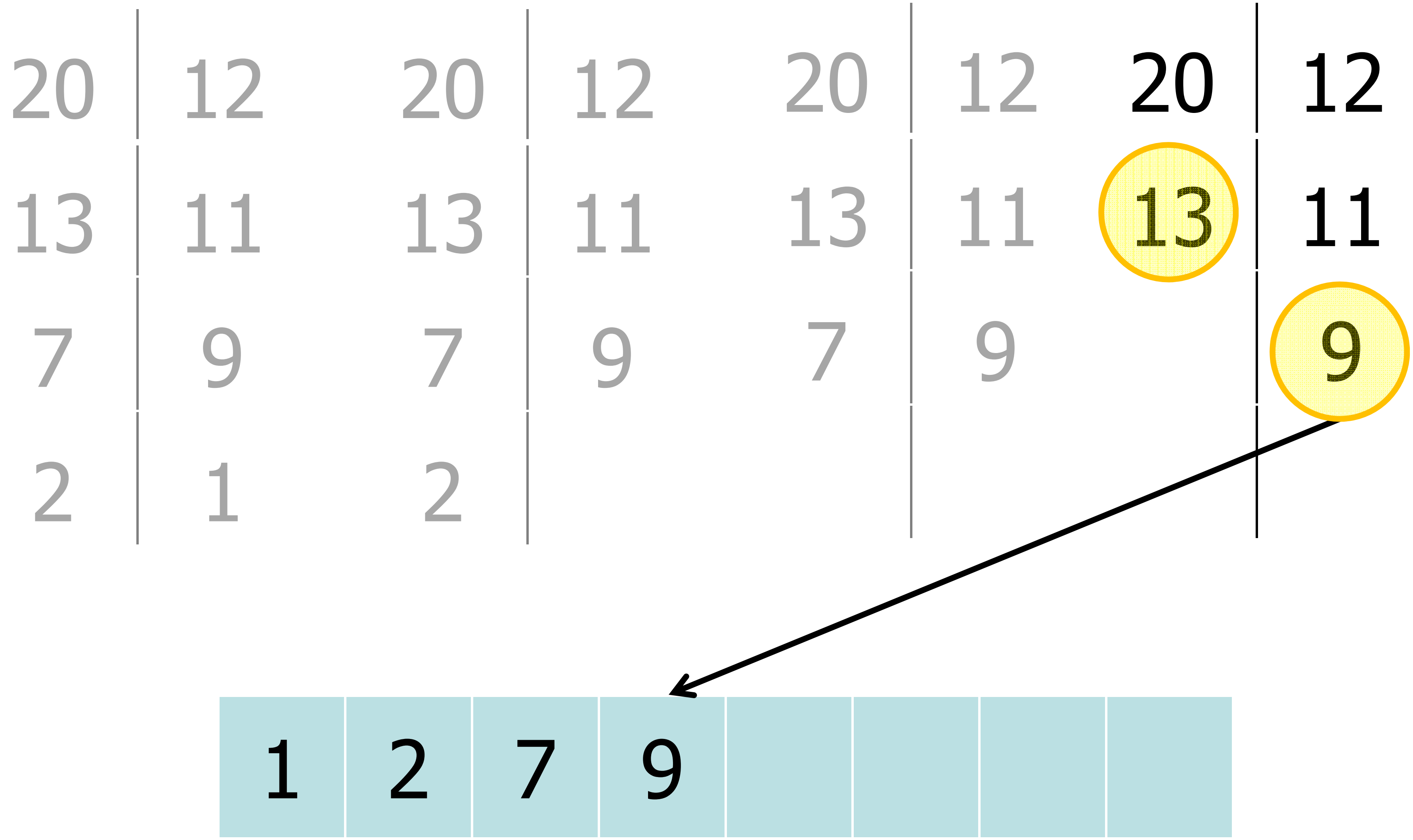
Merging Two Sorted Lists



Merging Two Sorted Lists



Merging Two Sorted Lists



Merging Two Sorted Lists

20	12
13	11
7	9
2	1

20	12
13	11
7	9
2	

20	12
13	11
7	9

20	12
13	11

1	2	7	9	11	12	13	20

Merge: Running Time

Given two lists:

- A of size n/2
- B of size n/2

Total running time: O(n) = cn

- In each iteration, move one element to final list

Merge-Sort Analysis

Let $T(n)$ be the worst-case running time for an array of n elements.

```
Merge-Sort(A, n)
    if (n=1) then return;  <-------------------- $\theta(1)$
    else:
        X ← Merge-Sort(…);  <------------ $T(n/2)$
        Y ← Merge-Sort(…);  <-------------- $T(n/2)$
     return Merge (X,Y, n/2);  <--------------- $\theta(n)$
```

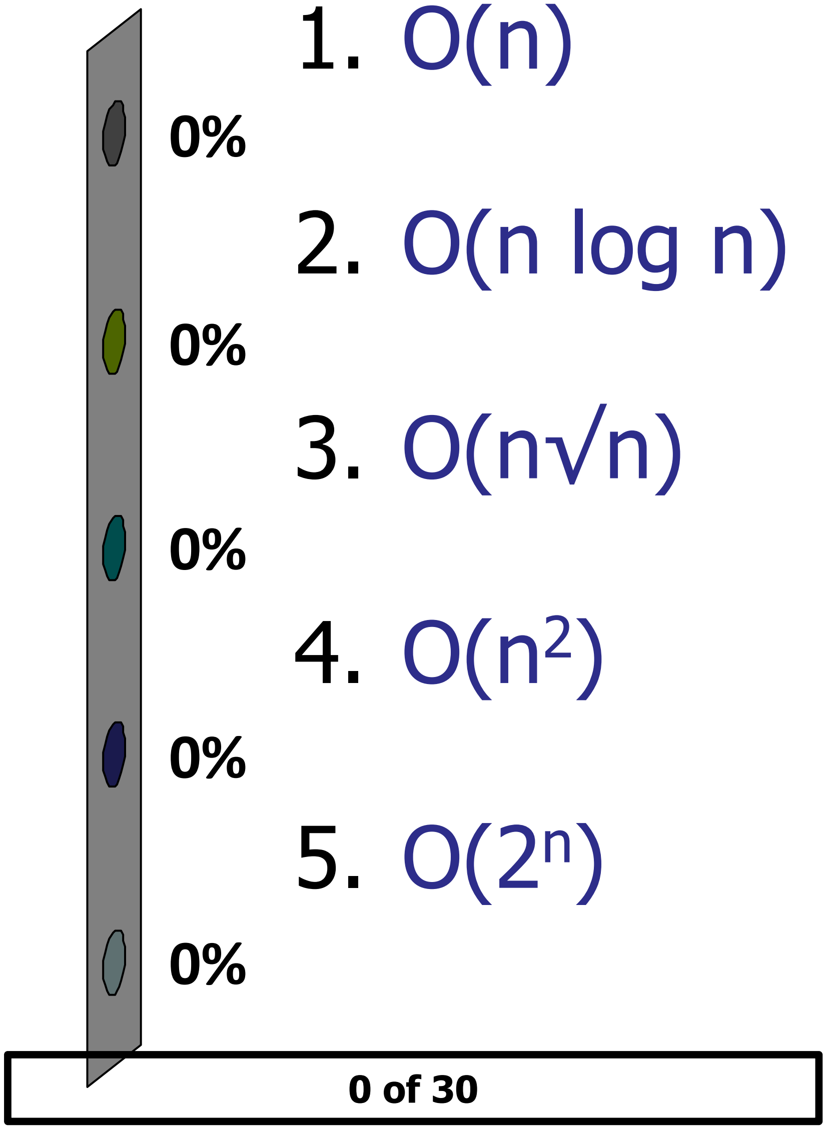
Merge-Sort Analysis

Let $T(n)$ be the worst-case running time for an array of n elements.

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

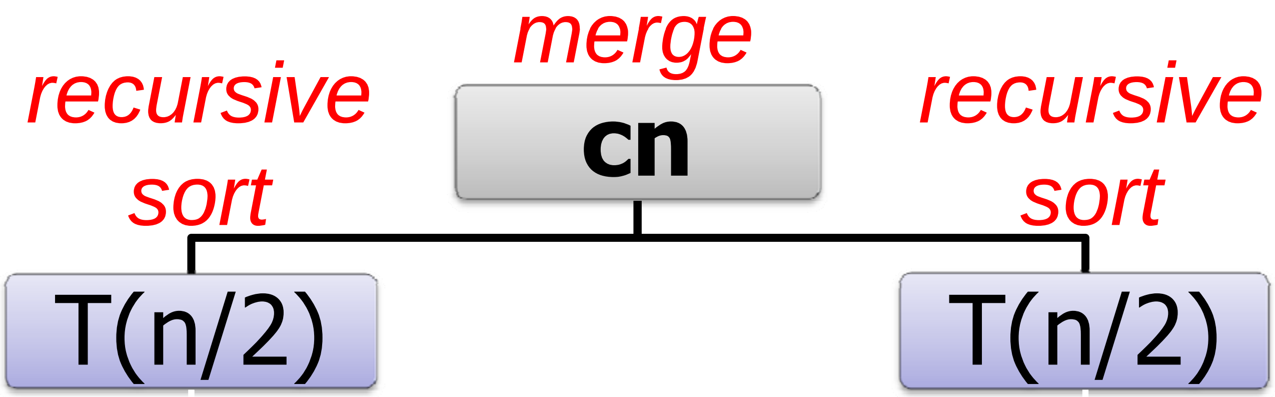
What is the running time of Merge-Sort?

1. O(n)
2. O(n log n)
3. $O(n\sqrt{n})$
4. $O(n^2)$
5. $O(2^n)$



Merge-Sort Analysis

$T(n) = 2T(n/2) + cn$



merge

cn

recursive sort

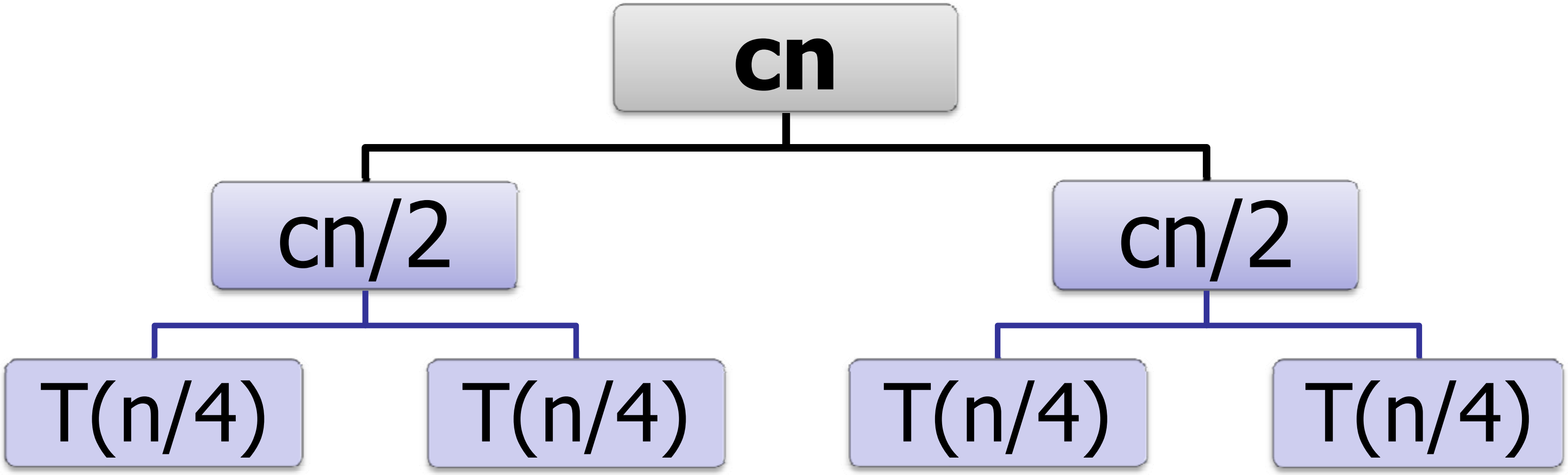
recursive sort

T(n/2)

T(n/2)

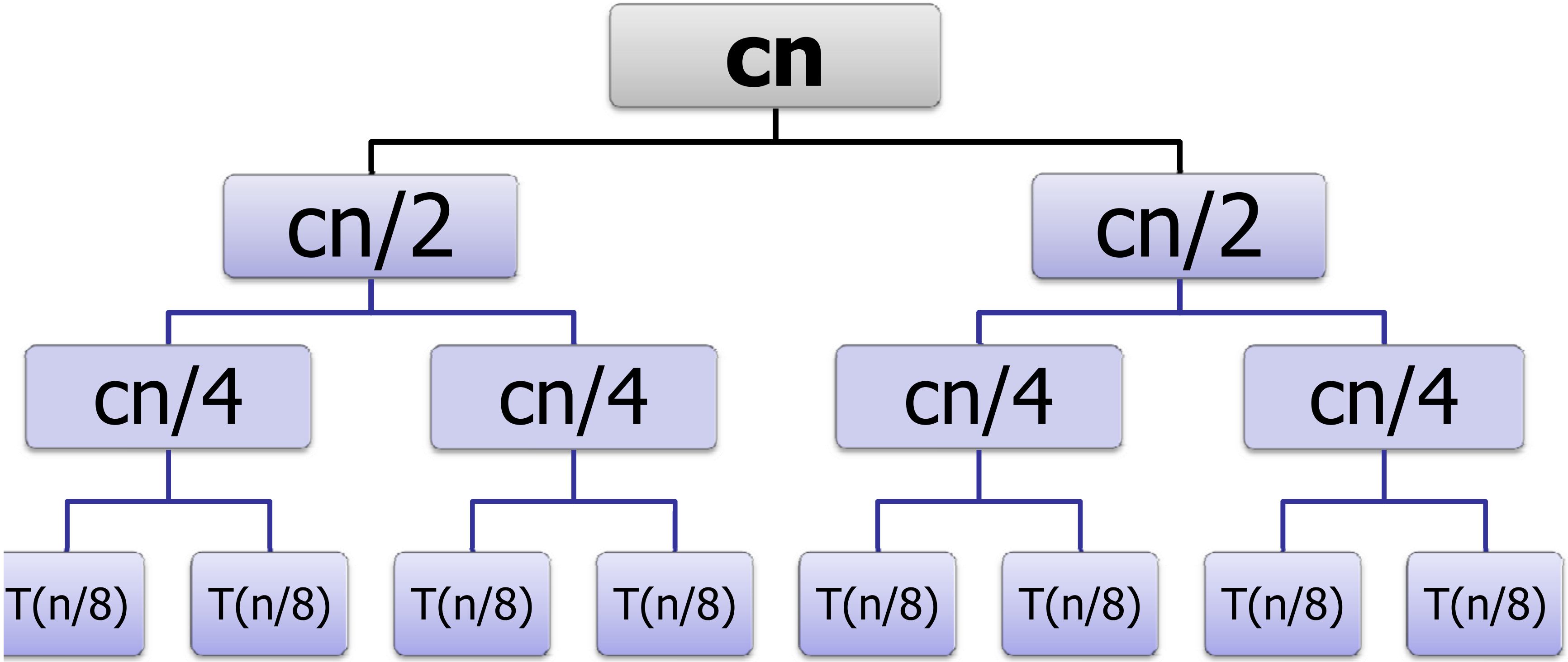
Merge-Sort Analysis

T(n) = 2T(n/2) + cn



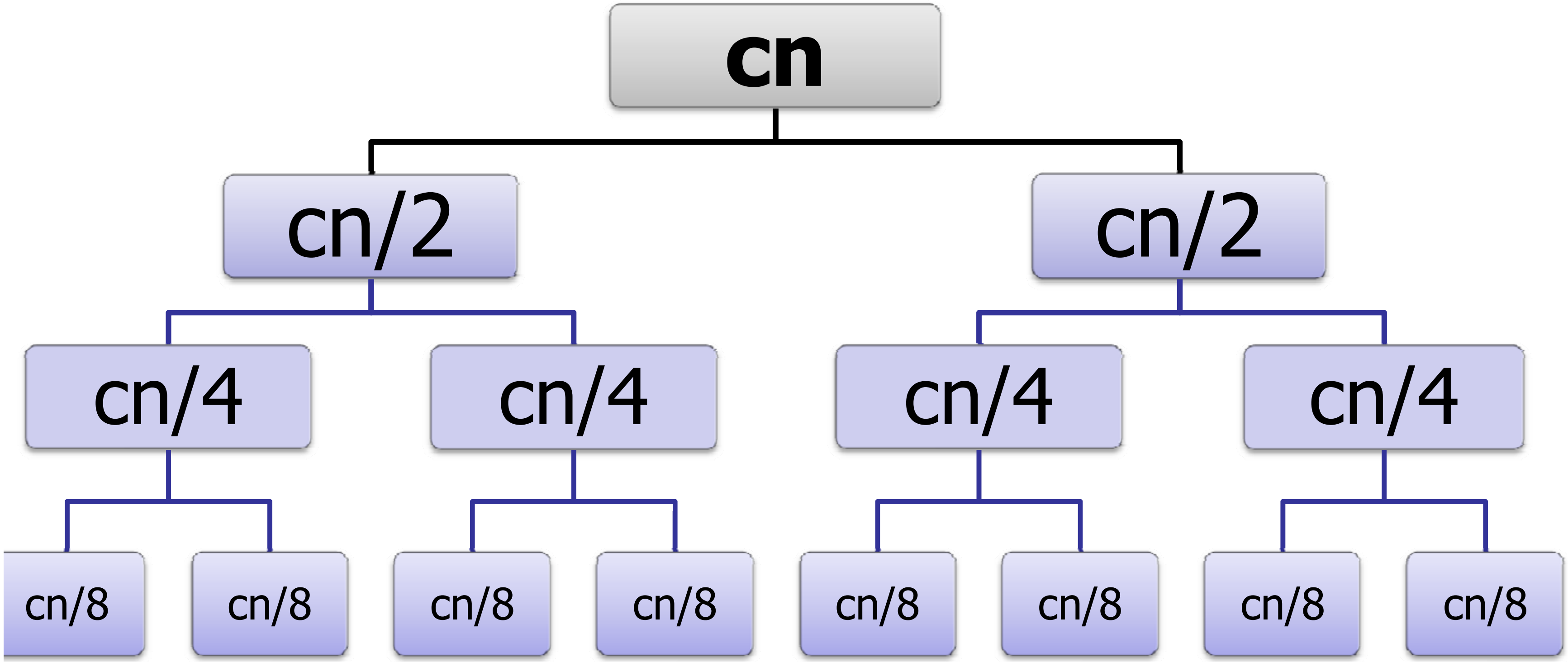
Merge-Sort Analysis

$T(n) = 2T(n/2) + cn$



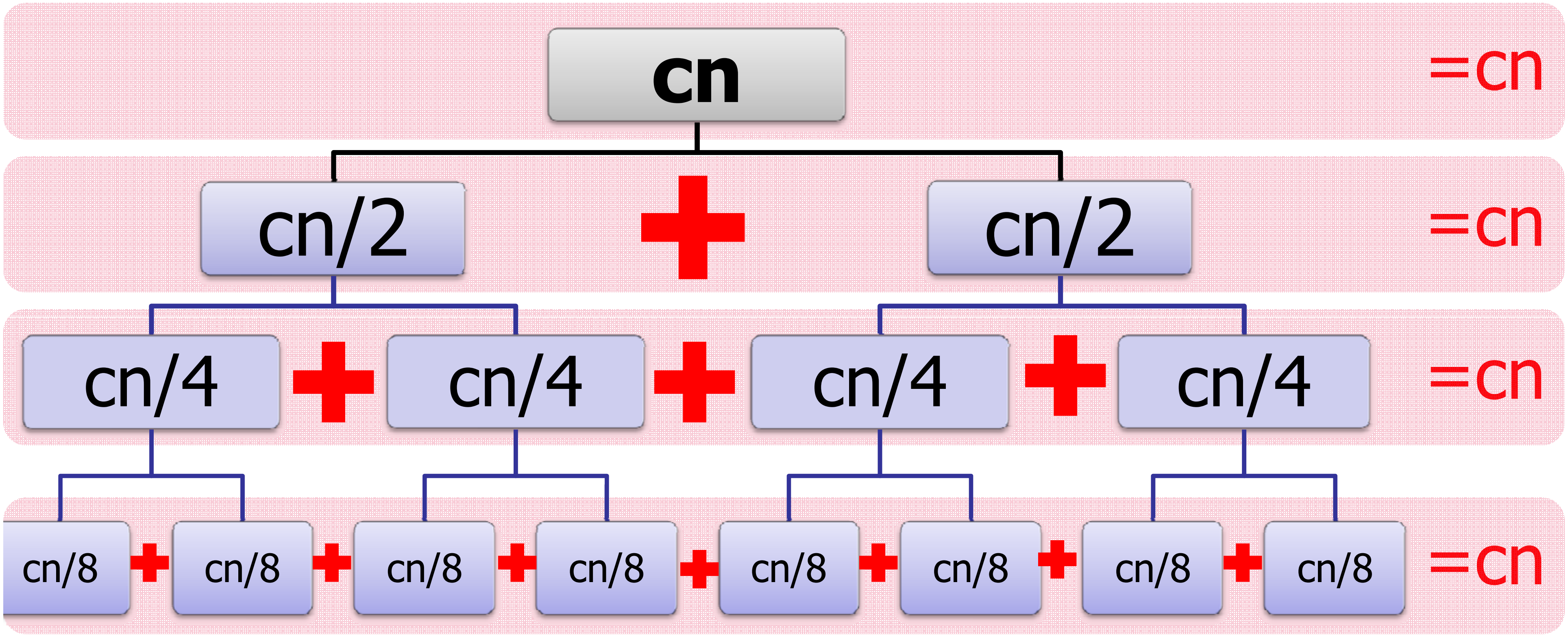
Merge-Sort Analysis

$T(n) = 2T(n/2) + cn$



Merge-Sort Analysis

$T(n) = 2T(n/2) + cn$



cn log n

Sorting Analysis

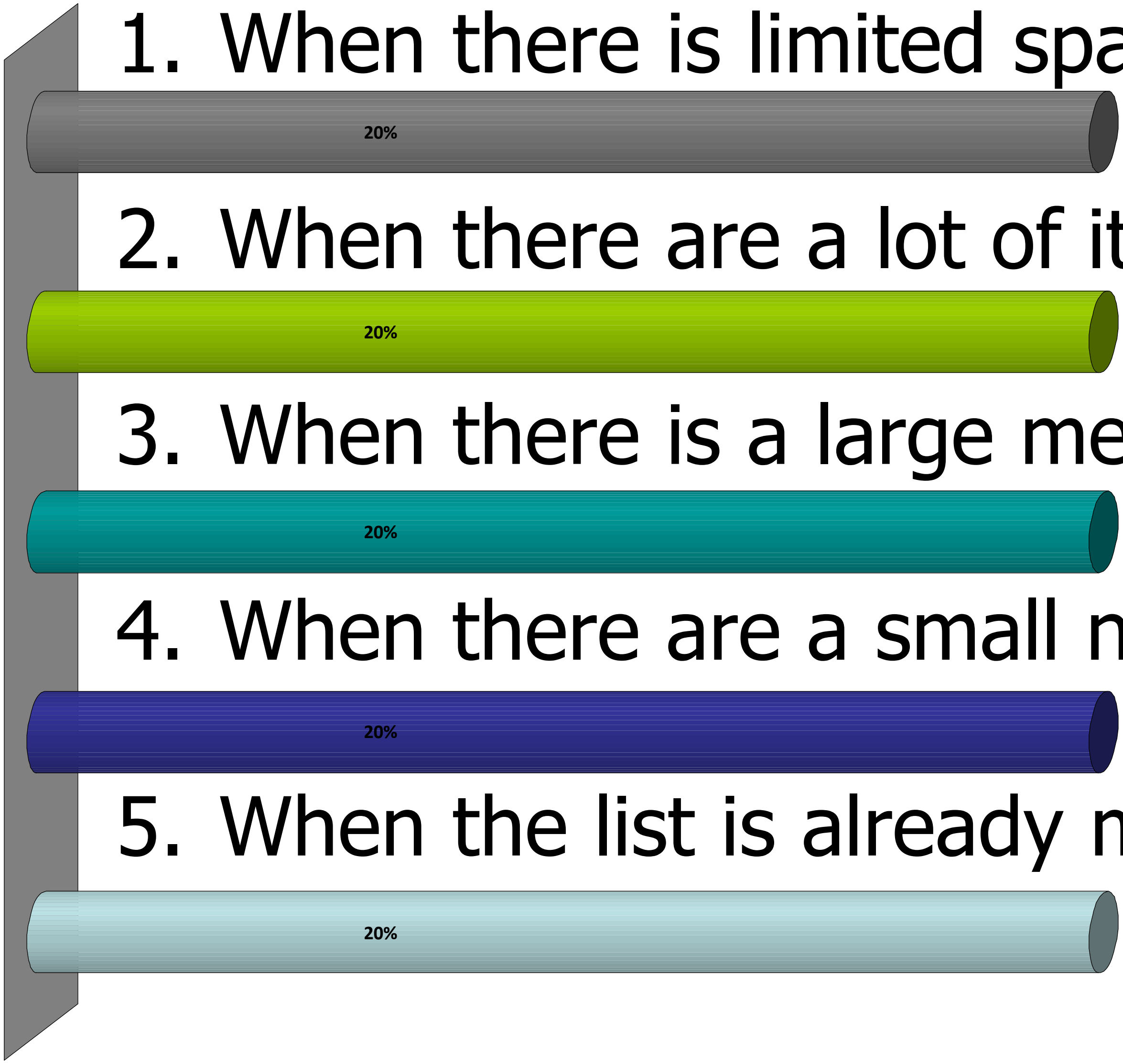
Summary:

InsertionSort: $O(n^2)$

MergeSort: $O(n \log n)$

When is it better to use InsertionSort instead of MergeSort?

1. When there is limited space.
2. When there are a lot of items to sort.
3. When there is a large memory cache.
4. When there are a small number of items.
5. When the list is already mostly sorted.



Sorting Analysis

When the list is mostly sorted:

- InsertionSort is fast!
- MergeSort is O(n log n)

How “close to sorted” should a list be for InsertionSort to be faster?

Sorting Analysis

- Small number of items to sort:
  - MergeSort is slow!
  - Caching performance, branch prediction, etc.
  - User InsertionSort for n < 1024, say.

- Base case of recursion:
  - Use slower sort.

Sorting Analysis

Limited space:

- Need extra space to do merge.
- Merge copies data to new array.
- How much extra space??

- In-place sorting…. in a few weeks.

For next time…

Monday lecture:

- Divide-and-Conquer

Friday tutorial:

- Details of Document Distance implementation

Discussion Groups:

- Starting next week. Sign up in CORS.

Problem Set 1:

- Released. Due next week.