
Data Structures & Algorithms

Overview

Prof. Kyosun Kim
Department of Electronic Engineering
Incheon National University



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

Course Overview

- outline
- why study algorithms?
- resources (books)

Course Overview

- Programming and problem solving with applications.
- Algorithm: method for solving a problem.
- Data structure: method to store information.

Topic	Data Structures and Algorithms
data types	stack, queue, list, union-find, priority queue
sorting	quicksort, mergesort, heapsort, radix sorts
searching	hash table, BST, red-black tree, B-tree
graphs	BFS, DFS, Prim, Kruskal, Dijkstra
strings	KMP, Rabin-Karp, TST, Huffman, LZW
geometry	Graham scan, k-d tree, Voronoi diagram

Why study algorithms?

□ Their impact is broad and far-reaching

- ◆ **Internet.** Web search, packet routing, distributed file sharing.
- ◆ **Biology.** Human genome project, protein folding.
- ◆ **Computers.** Circuit layout, file system, compilers.
- ◆ **Computer graphics.** Movies, video games, virtual reality.
- ◆ **Security.** Cell phones, e-commerce, voting machines.
- ◆ **Multimedia.** CD player, DVD, MP3, JPG, DivX, HDTV.
- ◆ **Transportation.** Airline crew scheduling, map routing.
- ◆ **Physics.** N-body simulation, particle collision simulation.

Why study algorithms?

□ Old roots, new opportunities

- ◆ Study of algorithms dates at least to Euclid
- ◆ Some important algorithms were discovered by undergraduates!

300 BC



1920s



1940s



1950s



1960s



1970s



1980s



1990s

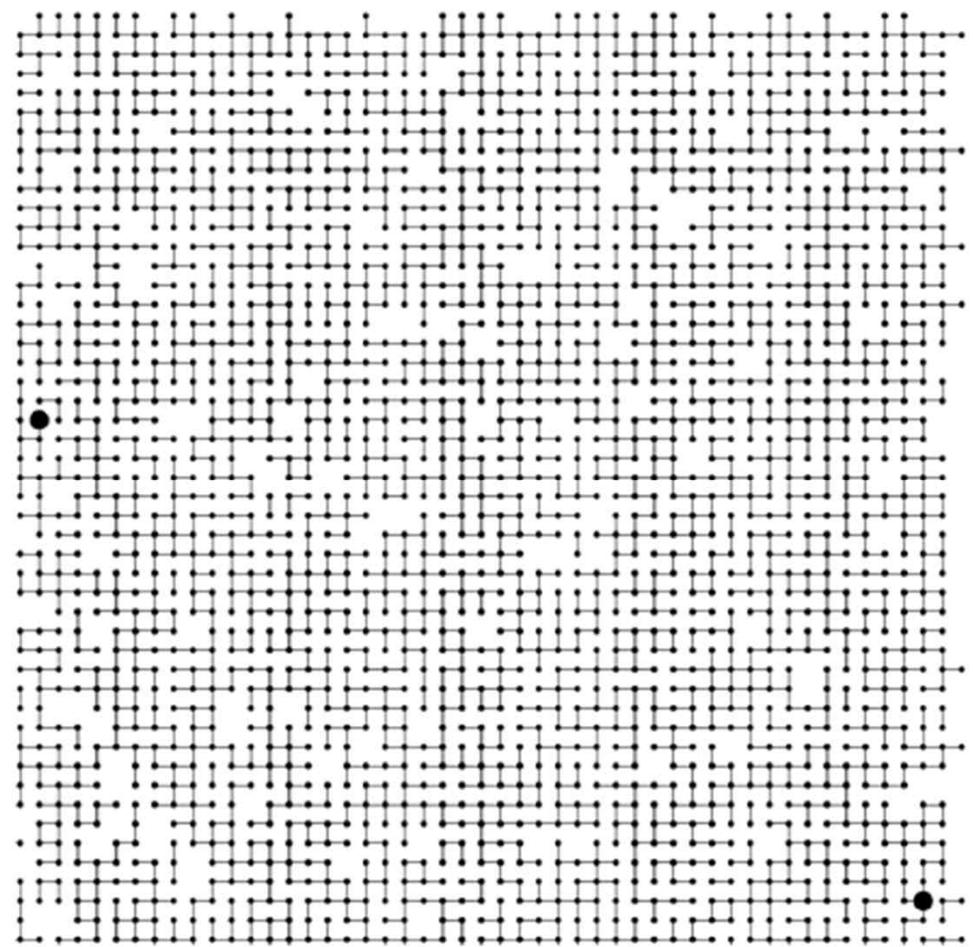


2000s



Why study algorithms?

- To be able solve problems that could not otherwise be addressed
 - ◆ Example: Network connectivity
 - ✓ [stay tuned]



Why study algorithms?

□ For intellectual stimulation

- ◆ For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing. - Francis Sullivan
- ◆ An algorithm must be seen to be believed. - D. E. Knuth

Why study algorithms?

- They may unlock the secrets of life and of the universe.
- ◆ Computational models are replacing mathematical models in scientific enquiry.

$$E = mc^2$$

$$F = ma$$

$$\left[-\frac{\hbar^2}{2m} \nabla^2 + V(r) \right] \Psi(r) = E \Psi(r)$$

20th century science
(formula based)

```
for (double t = 0.0; true; t = t + dt)
    for (int i = 0; i < N; i++)
    {
        bodies[i].resetForce();
        for (int j = 0; j < N; j++)
            if (i != j)
                bodies[i].addForce(bodies[j]);
    }
```

21st century science
(algorithm based)

- ◆ Algorithms: a common language for nature, human, and computer
 - Avi Wigderson

Why study algorithms?

- For fun and profit



Morgan Stanley

Microsoft

YAHOO!

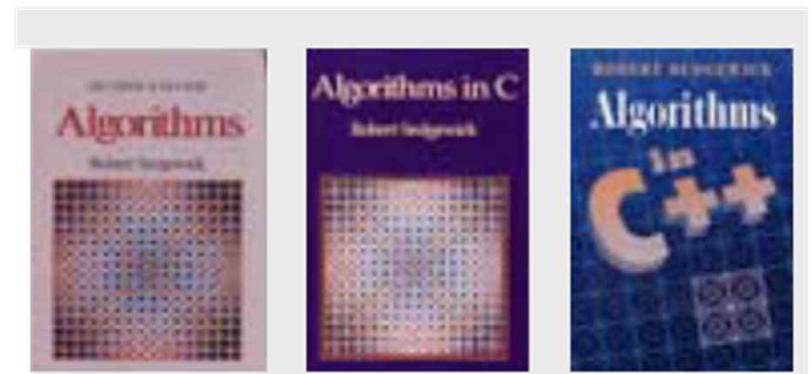
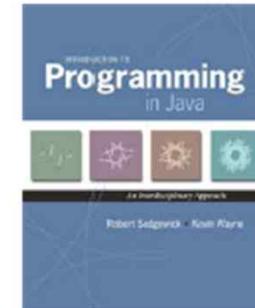
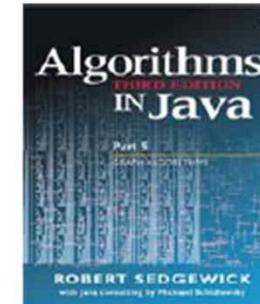
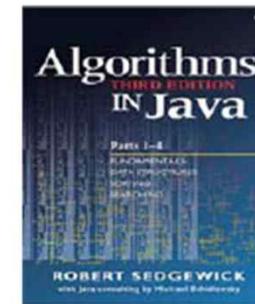
P I X A R

Why study algorithms?

- ◆ Their impact is broad and far-reaching
 - ◆ Old roots, new opportunities
 - ◆ To be able to solve problems that could not otherwise be addressed
 - ◆ For intellectual stimulation
 - ◆ They may unlock the secrets of life and of the universe
 - ◆ For fun and profit
-
- ◆ Anything else?

Resources (books)

- Algorithms in Java, 3rd edition
 - ◆ Parts 1-4. [sorting, searching]
 - ◆ Part 5. [graph algorithms]
- Introduction to Programming in Java
 - ◆ basic programming model
 - ◆ elementary AofA and data structures
- Algorithms in Pascal(!)/C/C++, 2nd edition
 - ◆ strings
 - ◆ elementary geometric algorithms
- This is Reformatted From
 - ◆ Lecture Note
 - ◆ Algorithms, Fall, 2008.
 - ◆ Robert Sedgewick
 - ◆ Princeton University



Data Structures & Algorithms

1. Union-Find Algorithms

- network connectivity
- quick find
- quick union
- improvements
- applications



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

Subtext of today's lecture (and this course)

- Steps to developing a usable algorithm.
 - ◆ Define the problem.
 - ◆ Find an algorithm to solve it.
 - ◆ Fast enough?
 - ◆ If not, figure out why.
 - ◆ Find a way to address the problem.
 - ◆ Iterate until satisfied.
- The scientific method
- Mathematical models and computational complexity

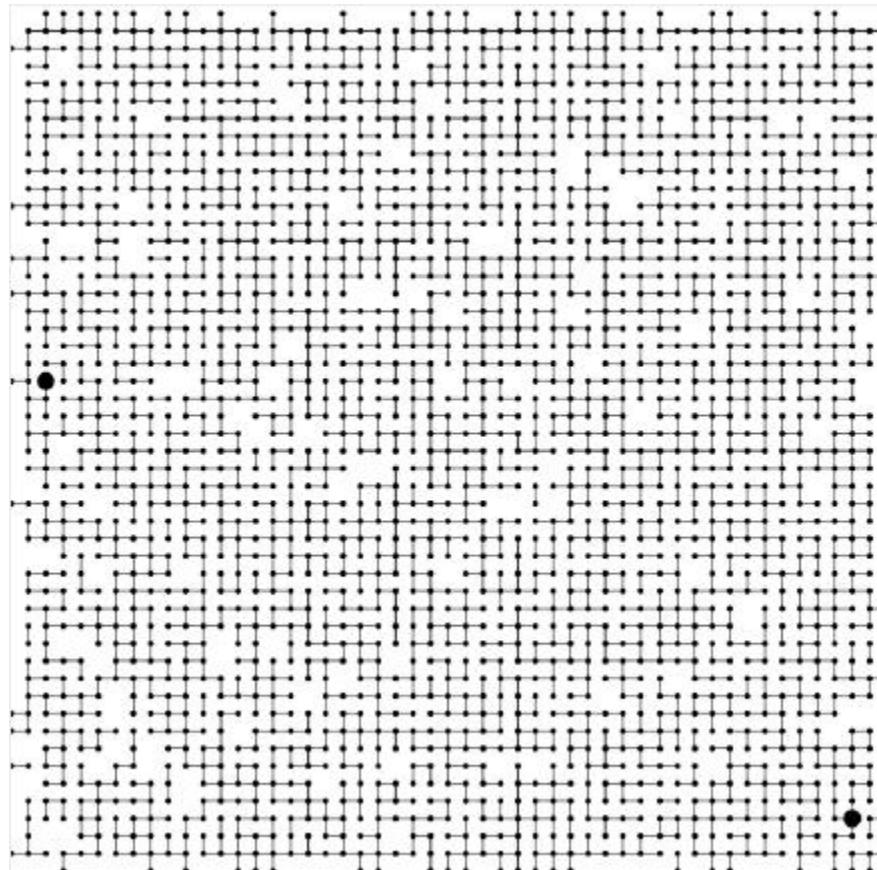
1. Union-Find Algorithms

- **network connectivity**
- **quick find**
- **quick union**
- **improvements**
- **applications**

Network connectivity

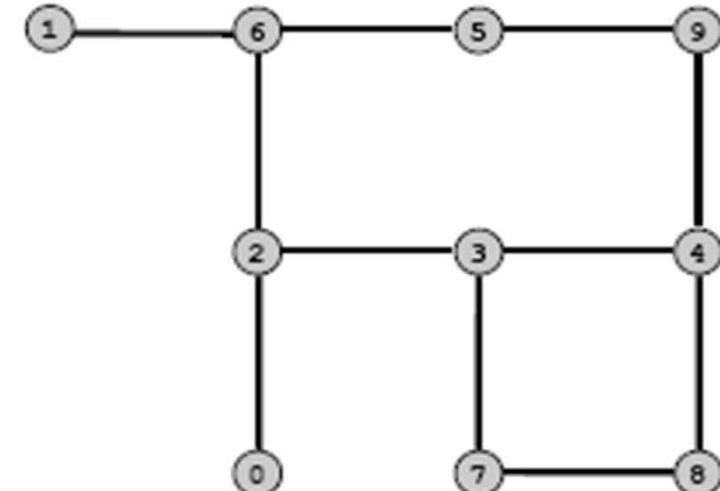
□ Basic abstractions

- ◆ set of objects
- ◆ union command: connect two objects
- ◆ find query: is there a path connecting one object to another?



Objects

- Union-find applications involve manipulating **objects** of all types.
 - ◆ Computers in a network.
 - ◆ Web pages on the Internet.
 - ◆ Transistors in a computer chip.
 - ◆ Variable name aliases.
 - ◆ Pixels in a digital photo.
 - ◆ Metallic sites in a composite system.
- When programming, convenient to name them 0 to N-1.
 - ◆ Hide details not relevant to union-find.
 - ◆ Integers allow quick access to object-related info (**use as array index**).
 - ◆ Could use **symbol table** to translate from object names



Union-find abstractions

- Simple model captures the essential nature of connectivity.
 - ◆ Objects.

0 1 2 3 4 5 6 7 8 9

grid points

- ◆ Disjoint sets of objects.

0 1 {2 3 9} {5 6} 7 {4 8}

subsets of connected grid points

- ◆ **Find** query: are objects 2 and 9 in the same set?

0 1 {2 3 9} {56} 7 {48}

are two grid points connected?

- ◆ **Union** command: merge sets containing 3 and 8.

0 1 {2 3 4 8 9} {56} 7

add a connection between
two grid points

Connected components

- Connected component: set of mutually connected vertices
 - ◆ Each union command reduces by 1 the number of components

in	out
3 4	3 4
4 9	4 9
8 0	8 0
2 3	2 3
5 6	5 6
2 9	
5 9	5 9
7 3	7 3

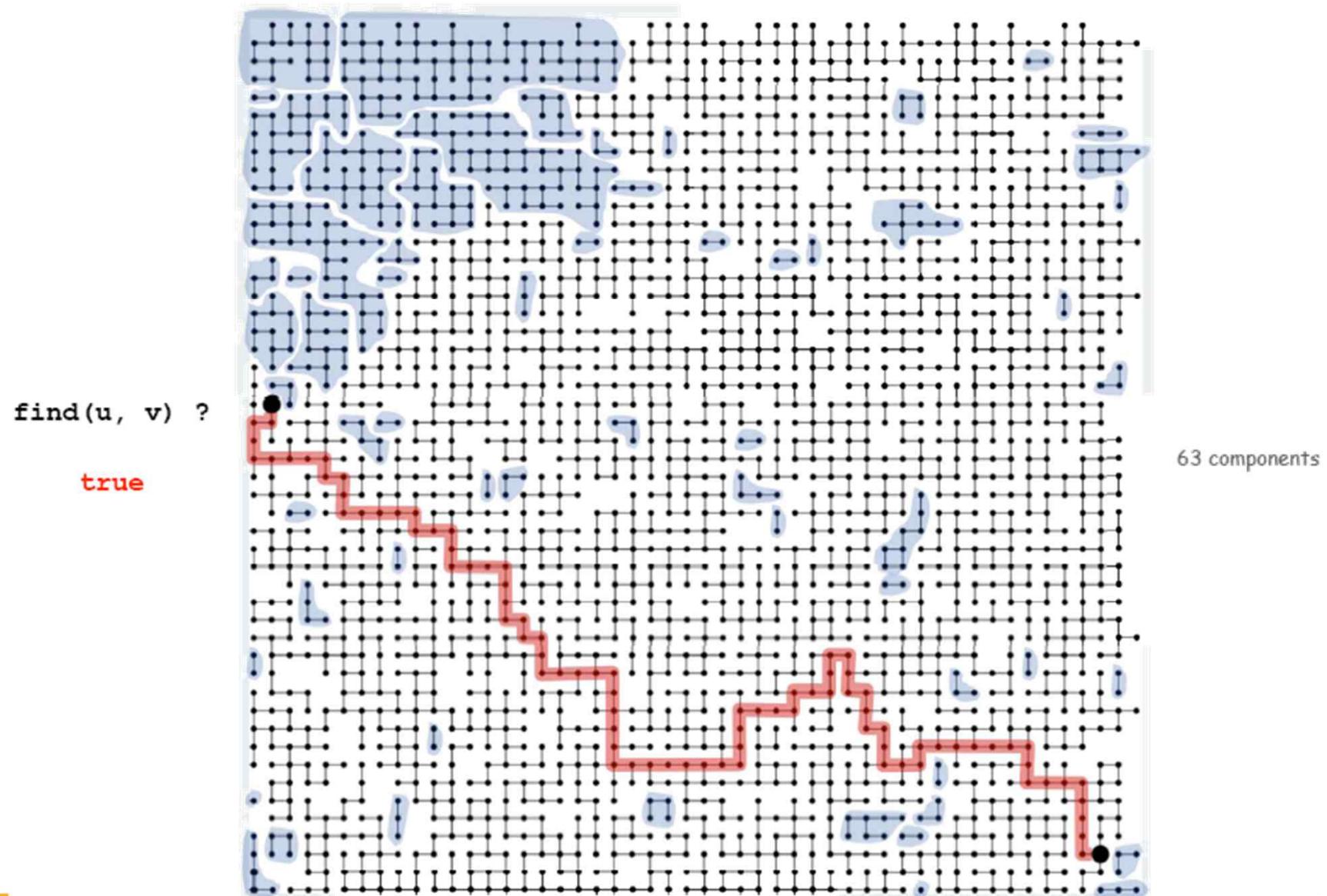
3 = 10-7 components

7 union commands

Network connectivity: larger example



Network connectivity: larger example



Union-find abstractions

- Objects.
 - Disjoint sets of objects.
 - Find queries: are two objects in the same set?
 - Union commands: replace sets containing two items by their union
-
- Goal. Design efficient data structure for union-find.
-
- Find queries and union commands may be intermixed.
 - Number of operations M can be huge.
 - Number of objects N can be huge.



1. Union-Find Algorithms

- network connectivity
- **quick find**
- quick union
- improvements
- applications

Quick-find [eager approach]

□ Data structure.

- ◆ Integer array $\text{id}[]$ of size N .
- ◆ Interpretation: p and q are connected if they have the same id .

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	9	9	6	6	7	8	9

5 and 6 are connected
2, 3, 4, and 9 are connected

Quick-find [eager approach]

□ Data structure.

- ◆ Integer array $\text{id}[]$ of size N .
- ◆ Interpretation: p and q are connected if they have the same id.

i	0	1	2	3	4	5	6	7	8	9
$\text{id}[i]$	0	1	9	9	9	6	6	7	8	9

5 and 6 are connected
2, 3, 4, and 9 are connected

□ Find. Check if p and q have the same id.

$\text{id}[3] = 9; \text{id}[6] = 6$
3 and 6 not connected

□ Union. To merge components containing p and q , change all entries with $\text{id}[p]$ to $\text{id}[q]$.

i	0	1	2	3	4	5	6	7	8	9
$\text{id}[i]$	0	1	6	6	6	6	6	7	8	6

union of 3 and 6
2, 3, 4, 5, 6, and 9 are connected

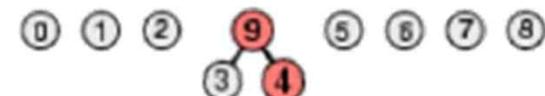
problem: many values can change

Quick-find example

3-4 0 1 2 4 4 5 6 7 8 9



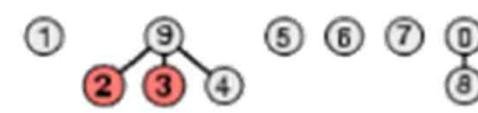
4-9 0 1 2 9 9 5 6 7 8 9



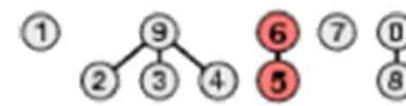
8-0 0 1 2 9 9 5 6 7 0 9



2-3 0 1 9 9 9 5 6 7 0 9



5-6 0 1 9 9 9 6 6 7 0 9



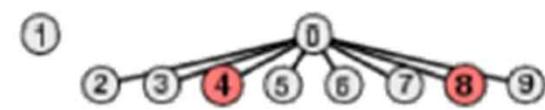
5-9 0 1 9 9 9 9 9 7 0 9



7-3 0 1 9 9 9 9 9 9 0 9



4-8 0 1 0 0 0 0 0 0 0 0



6-1 1 1 1 1 1 1 1 1 1 1



problem: many values can change

Quick-find: Java implementation

```
public class QuickFind
{
    private int[] id;

    public QuickFind(int N)
    {
        id = new int[N];
        for (int i = 0; i < N; i++)
            id[i] = i;
    }

    public boolean find(int p, int q)
    {
        return id[p] == id[q];
    }

    public void unite(int p, int q)
    {
        int pid = id[p];
        for (int i = 0; i < id.length; i++)
            if (id[i] == pid) id[i] = id[q];
    }
}
```

set id of each object to itself

1 operation

N operations

Quick-find is too slow

- Quick-find algorithm may take $\sim MN$ steps to process M union commands on N objects
- Rough standard (for now).
 - ◆ 10^9 operations per second.
 - ◆ 10^9 words of main memory.
 - ◆ Touch all words in approximately 1 second.
- Ex. Huge problem for quick-find.
 - ◆ 10^{10} edges connecting 10^9 nodes.
 - ◆ Quick-find takes more than 10^{19} operations.
 - ◆ **300+ years** of computer time!
- Paradoxically, quadratic algorithms get worse with newer equipment.
 - ◆ New computer may be 10x as fast.
 - ◆ But, has 10x as much memory so problem may be 10x bigger.
 - ◆ With quadratic algorithm, takes 10x as long!



1. Union-Find Algorithms

- network connectivity
- quick find
- quick union
- improvements
- applications

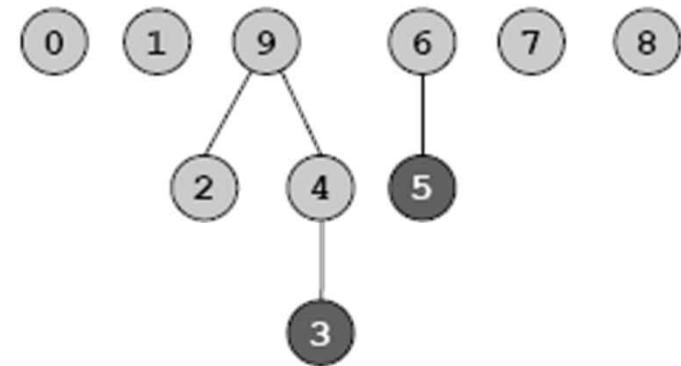
Quick-union [lazy approach]

□ Data structure.

- ◆ Integer array $\text{id}[]$ of size N .
- ◆ Interpretation: $\text{id}[i]$ is parent of i .
- ◆ Root of i is $\text{id}[\text{id}[\text{id}[\dots \text{id}[i] \dots]]]$.

keep going until it doesn't change

i	0	1	2	3	4	5	6	7	8	9
$\text{id}[i]$	0	1	9	4	9	6	6	7	8	9



3's root is 9; 5's root is 6

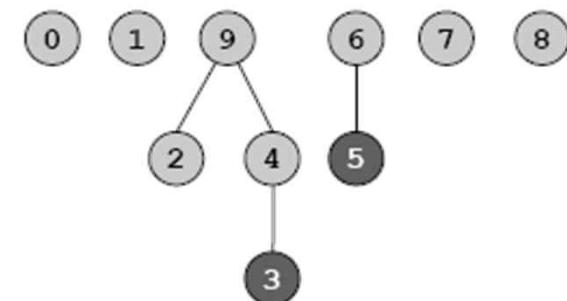
Quick-union [lazy approach]

□ Data structure.

- ◆ Integer array $\text{id}[]$ of size N .
- ◆ Interpretation: $\text{id}[i]$ is parent of i .
- ◆ Root of i is $\text{id}[\text{id}[\text{id}[\dots \text{id}[i] \dots]]]$.

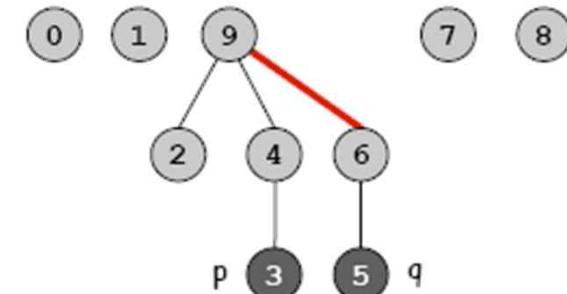
i	0	1	2	3	4	5	6	7	8	9
$\text{id}[i]$	0	1	9	4	9	6	6	7	8	9

keep going until it doesn't change



3's root is 9; 5's root is 6
3 and 5 are not connected

□ Find. Check if p and q have the same root.



□ Union. Set the id of q 's root to the id of p 's root.

i	0	1	2	3	4	5	6	7	8	9
$\text{id}[i]$	0	1	9	4	9	6	9	7	8	9

only one value changes

Quick-union example

3-4 0 1 2 4 4 5 6 7 8 9

4-9 0 1 2 4 9 5 6 7 8 9

8-0 0 1 2 4 9 5 6 7 0 9

2-3 0 1 9 4 9 5 6 7 0 9

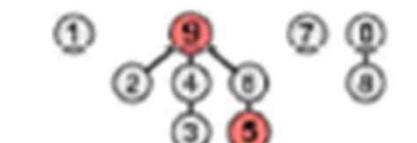
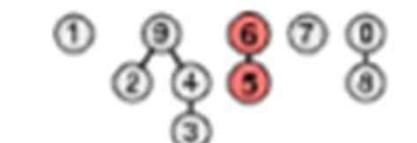
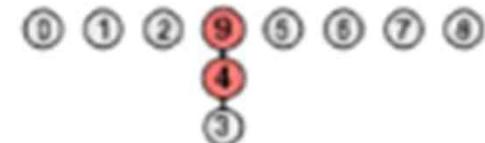
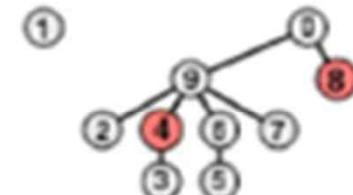
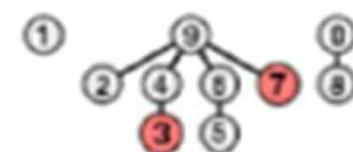
5-6 0 1 9 4 9 6 6 7 0 9

5-9 0 1 9 4 9 6 9 7 0 9

7-3 0 1 9 4 9 6 9 9 0 9

4-8 0 1 9 4 9 6 9 9 0 0

6-1 1 1 9 4 9 6 9 9 0 0



problem: trees can get tall

Algorithms

Quick-union: Java implementation

```
public class QuickUnion
{
    private int[] id;

    public QuickUnion(int N)
    {
        id = new int[N];
        for (int i = 0; i < N; i++) id[i] = i;
    }

    private int root(int i)
    {
        while (i != id[i]) i = id[i];
        return i;
    }

    public boolean find(int p, int q)
    {
        return root(p) == root(q);
    }

    public void unite(int p, int q)
    {
        int i = root(p);
        int j = root(q);
        id[i] = j;
    }
}
```

time proportional
to depth of i

time proportional
to depth of p and q

time proportional
to depth of p and q

Quick-union is also too slow

- Quick-find defect.
 - ◆ Union too expensive (N steps).
 - ◆ Trees are flat, but too expensive to keep them flat.
- Quick-union defect.
 - ◆ Trees can get tall.
 - ◆ Find too expensive (could be N steps)
 - ◆ Need to do find to do union

algorithm	union	find
Quick-find	N	1
Quick-union	N^*	N ← worst case

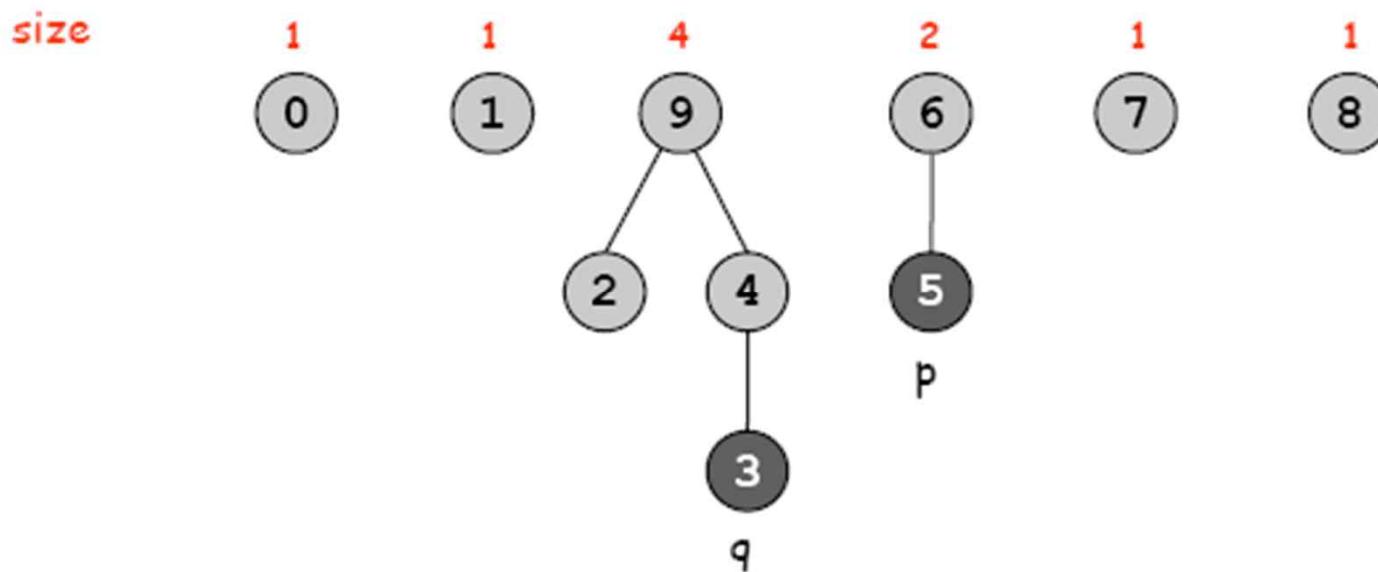
* includes cost of find

1. Union-Find Algorithms

- network connectivity
- quick find
- quick union
- improvements
- applications

Improvement 1: Weighting

- Weighted quick-union.
 - ◆ Modify quick-union to avoid tall trees.
 - ◆ Keep track of size of each component.
 - ◆ Balance by linking small tree below large one.
- Ex. Union of 5 and 3.
 - ◆ Quick union: link 9 to 6.
 - ◆ Weighted quick union: link 6 to 9.



Weighted quick-union example

3-4 0 1 2 3 3 5 6 7 8 9

4-9 0 1 2 3 3 5 6 7 8 3

8-0 8 1 2 3 3 5 6 7 8 3

2-3 8 1 3 3 3 5 6 7 8 3

5-6 8 1 3 3 3 5 5 7 8 3

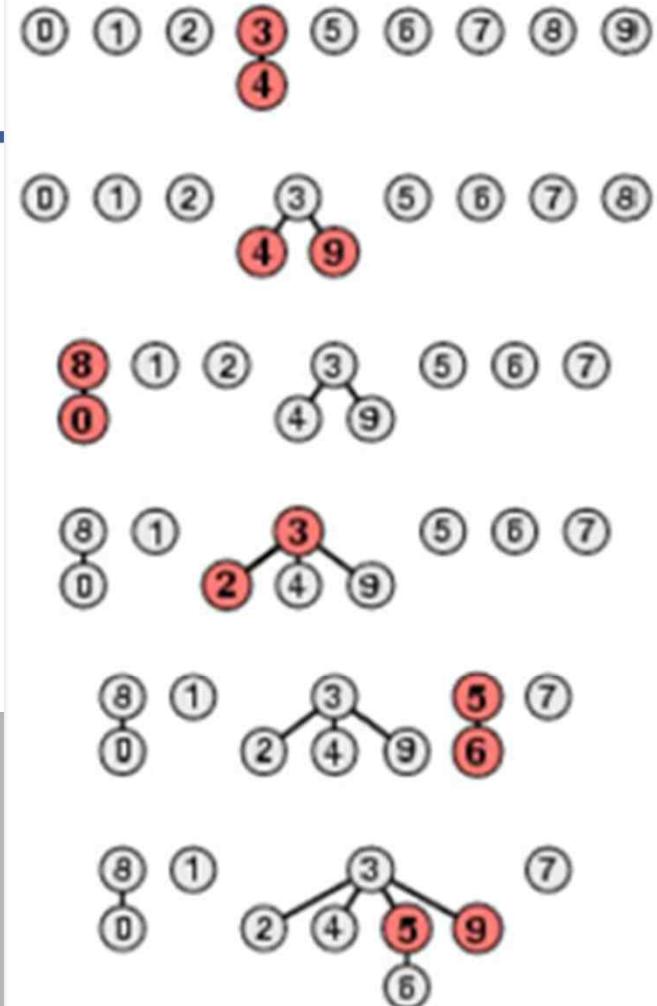
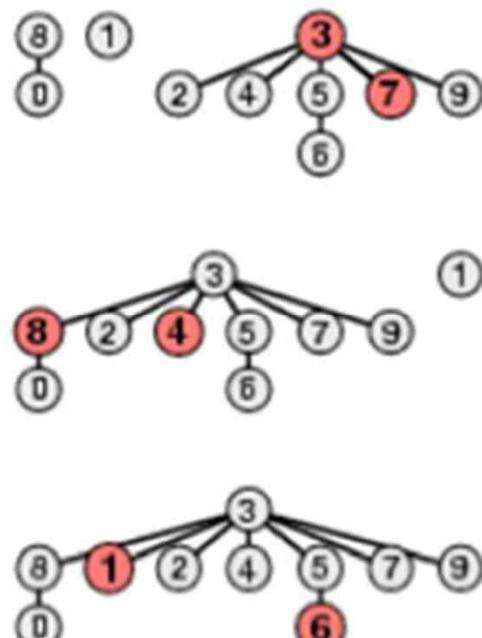
5-9 8 1 3 3 3 3 5 7 8 3

7-3 8 1 3 3 3 3 5 3 8 3

4-8 8 1 3 3 3 3 5 3 3 3

6-1 8 3 3 3 3 3 5 3 3 3

no problem: trees stay flat →



Weighted quick-union: Java implementation

□ Java implementation.

- ◆ Almost identical to quick-union.
- ◆ Maintain extra array `sz[]` to count number of elements in the tree rooted at i .

□ Find. Identical to quick-union.

□ Union. Modify quick-union to

- ◆ merge smaller tree into larger tree
- ◆ update the `sz[]` array.

```
if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else sz[i] < sz[j] { id[j] = i; sz[i] += sz[j]; }
```

Weighted quick-union analysis

□ Analysis.

- ◆ Find: takes time proportional to depth of p and q.
- ◆ Union: takes constant time, given roots.
- ◆ Fact: depth is at most $\lg N$. [needs proof]

Data Structure	Union	Find
Quick-find	N	1
Quick-union	N^*	N
Weighted QU	$\lg N^*$	$\lg N$

* includes cost of find

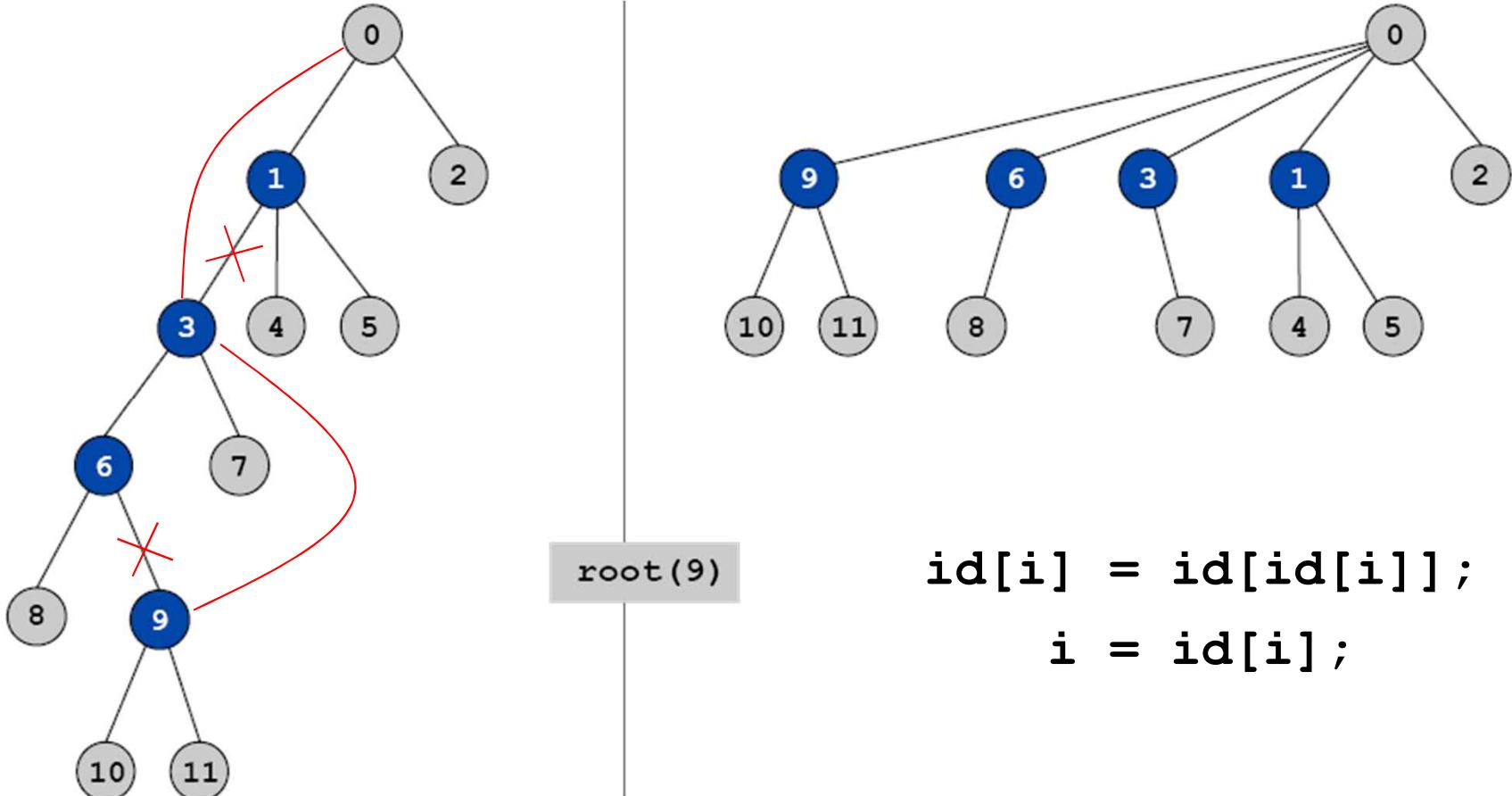
□ Stop at guaranteed acceptable performance?

- ◆ No, easy to improve further.

Improvement 2: Path compression

□ Path compression.

- ◆ Just after computing the root of i ,
- ◆ set the id of each examined node to $\text{root}(i)$.



Weighted quick-union with path compression

□ Path compression.

- ◆ Standard implementation: add second loop to root() to set the id of each examined node to the root.
- ◆ Simpler one-pass variant: make every other node in path point to its grandparent.

```
public int root(int i)
{
    while (i != id[i])
    {
        id[i] = id[id[i]];
        i = id[i];
    }
    return i;
}
```

only one extra line of code !

□ In practice.

- ◆ No reason not to! Keeps tree almost completely flat.



Weighted quick-union with path compression

3-4 0 1 2 3 3 5 6 7 8 9

4-9 0 1 2 3 3 5 6 7 8 3

8-0 8 1 2 3 3 5 6 7 8 3

2-3 8 1 3 3 3 5 6 7 8 3

5-6 8 1 3 3 3 5 5 7 8 3

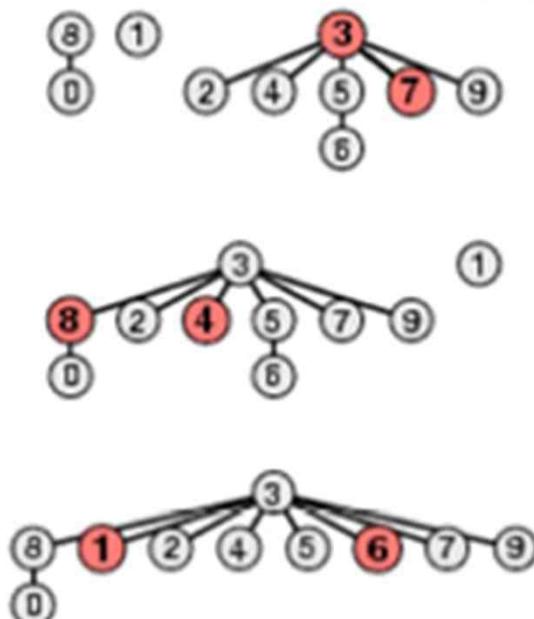
5-9 8 1 3 3 3 3 5 7 8 3

7-3 8 1 3 3 3 3 5 3 8 3

4-8 8 1 3 3 3 3 5 3 3 3

6-1 8 3 3 3 3 3 3 3 3 3

no problem: trees stay **VERY** flat →



WQUPC performance

□ Theorem.

- ◆ Starting from an empty data structure, any sequence of M union and find operations on N objects takes $O(N + M \lg^* N)$ time.
- ◆ Proof is **very** difficult.
- ◆ But the algorithm is still simple!

number of times needed to take
the \lg of a number until reaching 1

□ Linear algorithm?

- ◆ Cost within constant factor of reading in the data.
- ◆ In theory, WQUPC is not quite linear.
- ◆ In practice, WQUPC is **linear**.

↑
because $\lg^* N$ is a constant
in this universe

□ Amazing fact:

- ◆ In theory, no **linear** linking strategy exists

N	$\lg^* N$
1	0
2	1
4	2
16	3
65536	4
265536	5

Summary

Algorithm	Worst-case time
Quick-find	$M N$
Quick-union	$M N$
Weighted QU	$N + M \log N$
Path compression	$N + M \log N$
Weighted + path	$(M + N) \lg^* N$

M union-find ops on a set of N objects

□ Ex. Huge practical problem.

- ◆ 10^{10} edges connecting 10^9 nodes.
- ◆ WQUPC reduces time from 3,000 years to 1 minute.
- ◆ Supercomputer won't help much.
- ◆ Good algorithm makes solution possible.

□ Bottom line.

- ◆ WQUPC makes it possible to solve problems that could not otherwise be addressed



1. Union-Find Algorithms

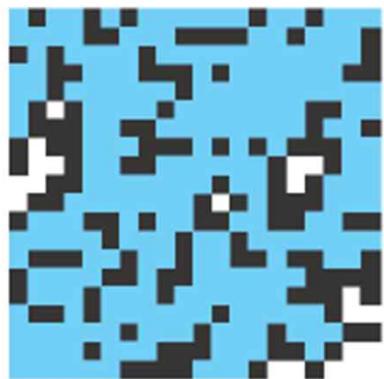
- network connectivity
- quick find
- quick union
- improvements
- applications

Union-find applications

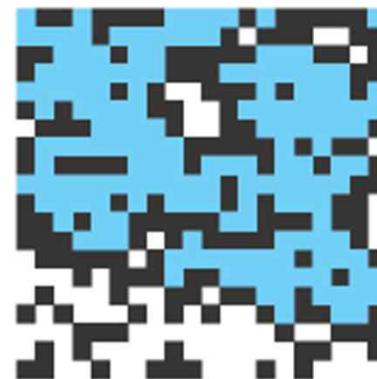
- Network connectivity.
- Percolation.
- Image processing.
- Least common ancestor.
- Equivalence of finite state automata.
- Hinley-Milner polymorphic type inference.
- Kruskal's minimum spanning tree algorithm.
- Games (Go, Hex)
- Compiling equivalence statements in Fortran.

Percolation

- A model for many physical systems
 - ◆ N-by-N grid.
 - ◆ Each square is vacant or occupied.
 - ◆ Grid percolates if top and bottom are connected by vacant squares.



percolates

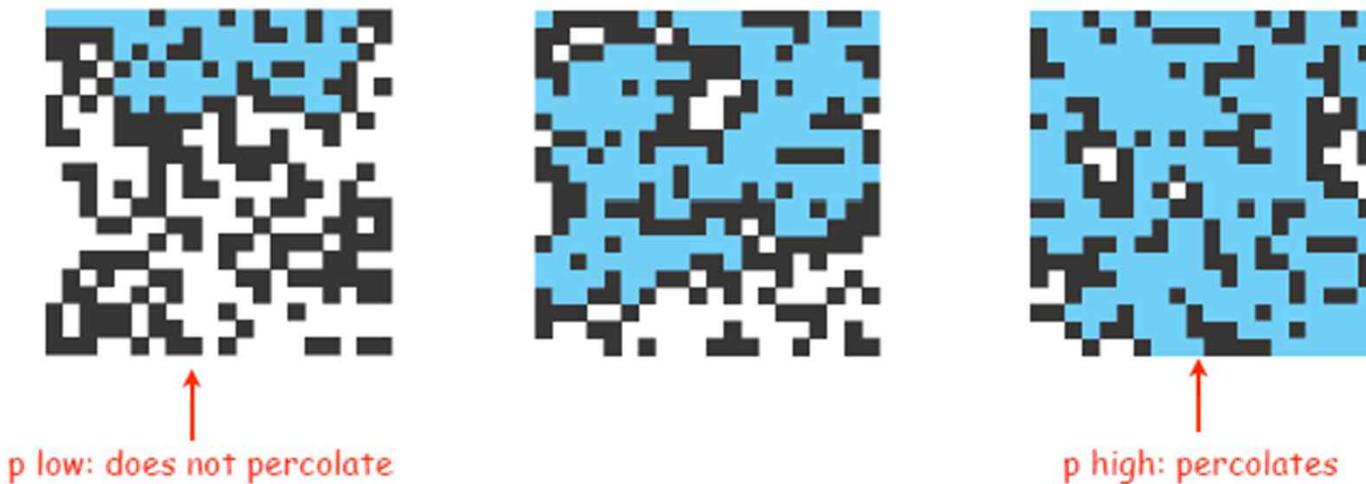


does not percolate

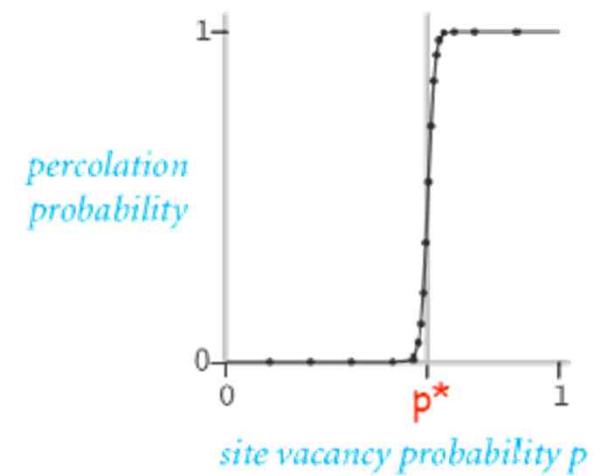
model	system	vacant site	occupied site	percolates
electricity	material	conductor	insulated	conducts
fluid flow	material	empty	blocked	porous
social interaction	population	person	empty	communicates

Percolation phase transition

- ☐ Likelihood of percolation depends on site vacancy probability p

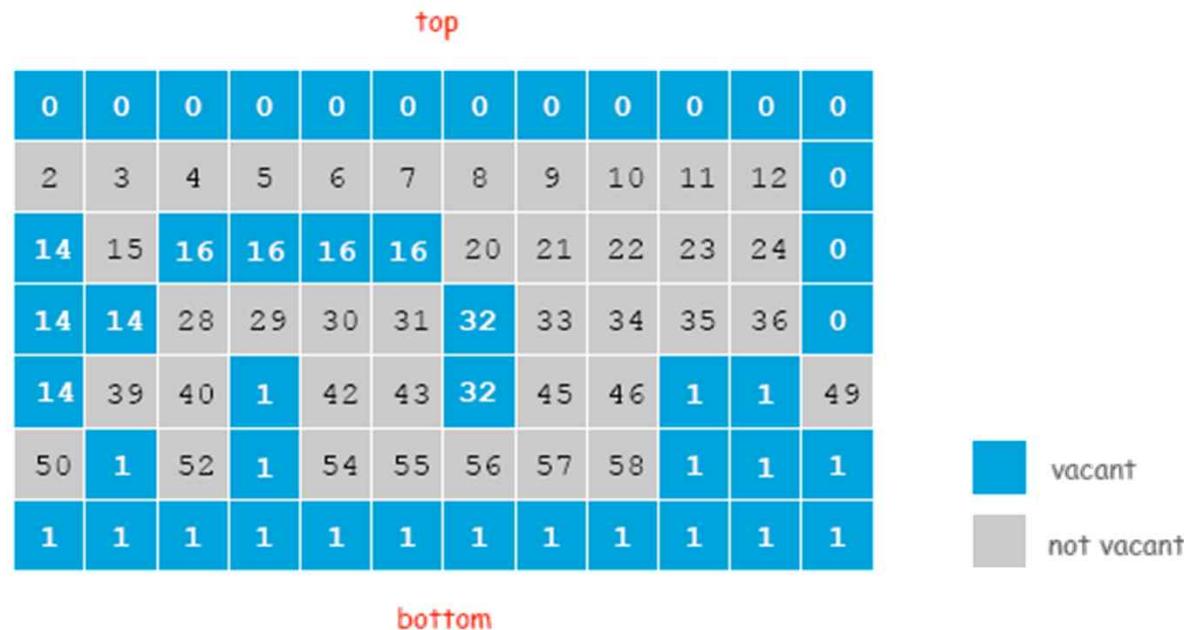


- ☐ Experiments show a threshold p^*
 - ◆ $p > p^*$: almost certainly percolates
 - ◆ $p < p^*$: almost certainly does not percolate
- ☐ Q. What is the value of p^* ?



UF solution to find percolation threshold

- Initialize whole grid to be “not vacant”
- Implement “make site vacant” operation that does `union()` with adjacent sites
- Make all sites on top and bottom rows vacant
- Make random sites vacant until `find(top, bottom)`
- Vacancy percentage estimates p^*

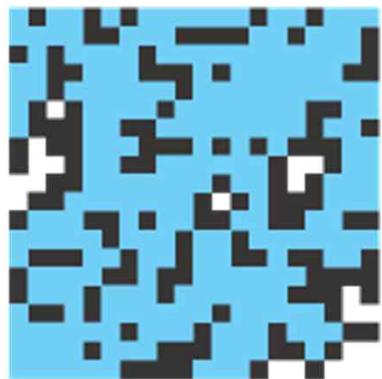


Percolation

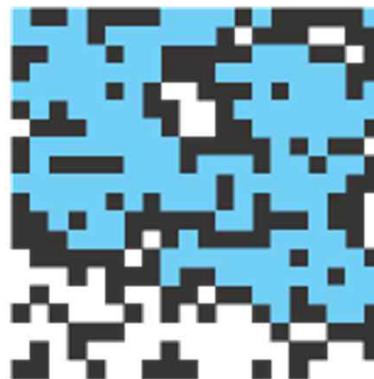
Q. What is percolation threshold p^* ?

A. about 0.592746 for large square lattices.

percolation constraint known
only via simulation



percolates



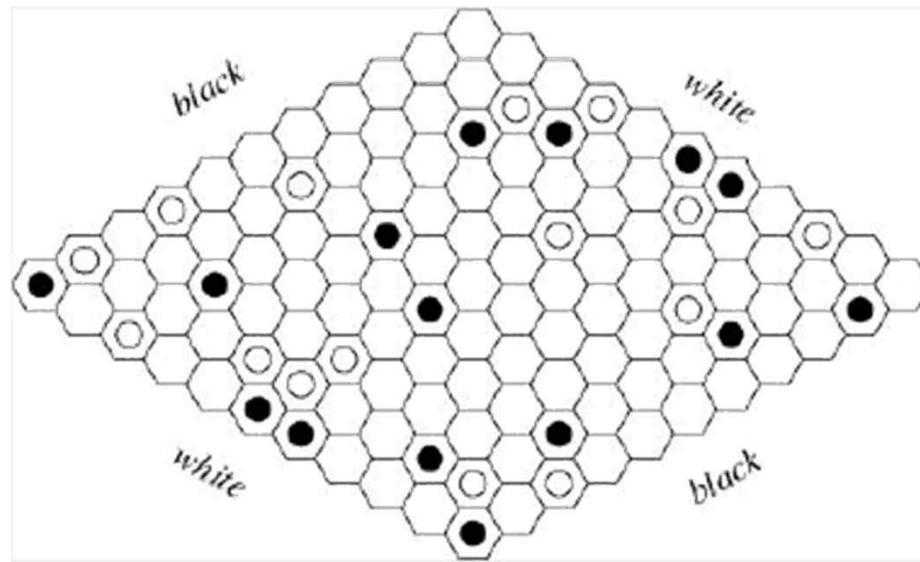
does not percolate

Q. Why is UF solution better than solution in IntroProgramming 2.4?

Hex

□ Hex. [Piet Hein 1942, John Nash 1948, Parker Brothers 1962]

- ◆ Two players alternate in picking a cell in a hex grid.
- ◆ Black: make a black path from upper left to lower right.
- ◆ White: make a white path from lower left to upper right.



Reference: <http://mathworld.wolfram.com/GameofHex.html>

□ Union-find application. Algorithm to detect when a player has won.

Subtext of today's lecture (and this course)

- Steps to developing a usable algorithm.
 - ◆ Define the problem.
 - ◆ Find an algorithm to solve it.
 - ◆ Fast enough?
 - ◆ If not, figure out why.
 - ◆ Find a way to address the problem.
 - ◆ Iterate until satisfied.
- The scientific method
- Mathematical models and computational complexity

Data Structures & Algorithms

2. Stacks & Queues

- stacks
- dynamic resizing
- queues
- generics
- applications



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

Stacks and Queues

□ Fundamental data types.

- ◆ Values: sets of objects
- ◆ Operations: **insert, remove, test if empty.**
- ◆ Intent is clear when we insert.
- ◆ Which item do we remove?

□ Stack.

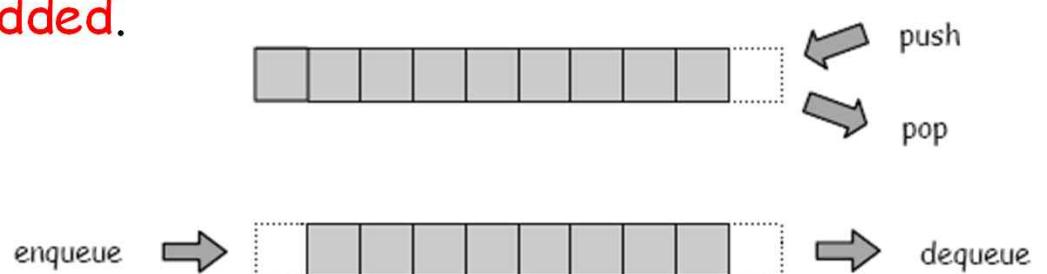
- ◆ Remove the item **most recently added.**
- ◆ Analogy: cafeteria trays, Web surfing.

LIFO="last in first out"

□ Queue.

- ◆ Remove the item **least recently added.**
- ◆ Analogy: Registrar's line.

FIFO="first in first out"



Client, Implementation, Interface

- Separate interface and implementation so as to:
 - ◆ Build layers of abstraction.
 - ◆ Reuse software.
 - ◆ Ex: stack, queue, symbol table.

Interface: description of data type, basic operations.

Client: program using operations defined in interface.

Implementation: actual code implementing operations.

Client, Implementation, Interface

- Separate interface and implementation so as to:
 - ◆ Build layers of abstraction.
 - ◆ Reuse software.
 - ◆ Ex: stack, queue, symbol table.
 - ◆ Design: creates modular, re-usable libraries.
 - ◆ Performance: use optimized implementation where it matters.

Interface: description of data type, basic operations.

Client: program using operations defined in interface.

Implementation: actual code implementing operations.

2. Stacks & Queues

- stacks
- dynamic resizing
- queues
- generics
- applications

Stacks

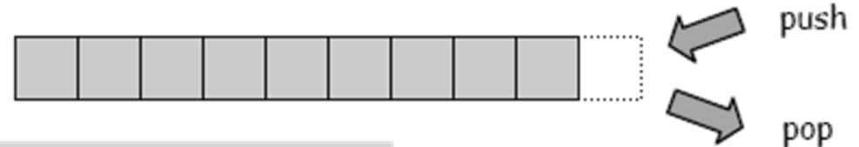
□ Stack operations.

- ◆ push()
- ◆ pop()
- ◆ isEmpty()

Insert a new item onto stack.

Remove and return the item most recently added.

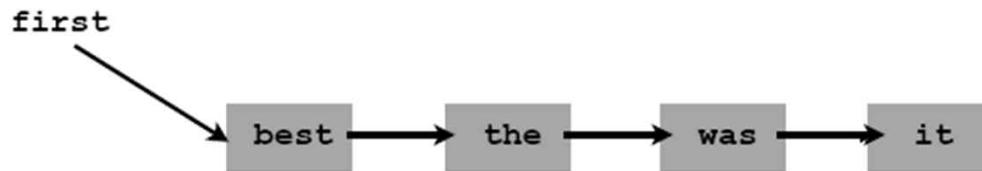
Is the stack empty?



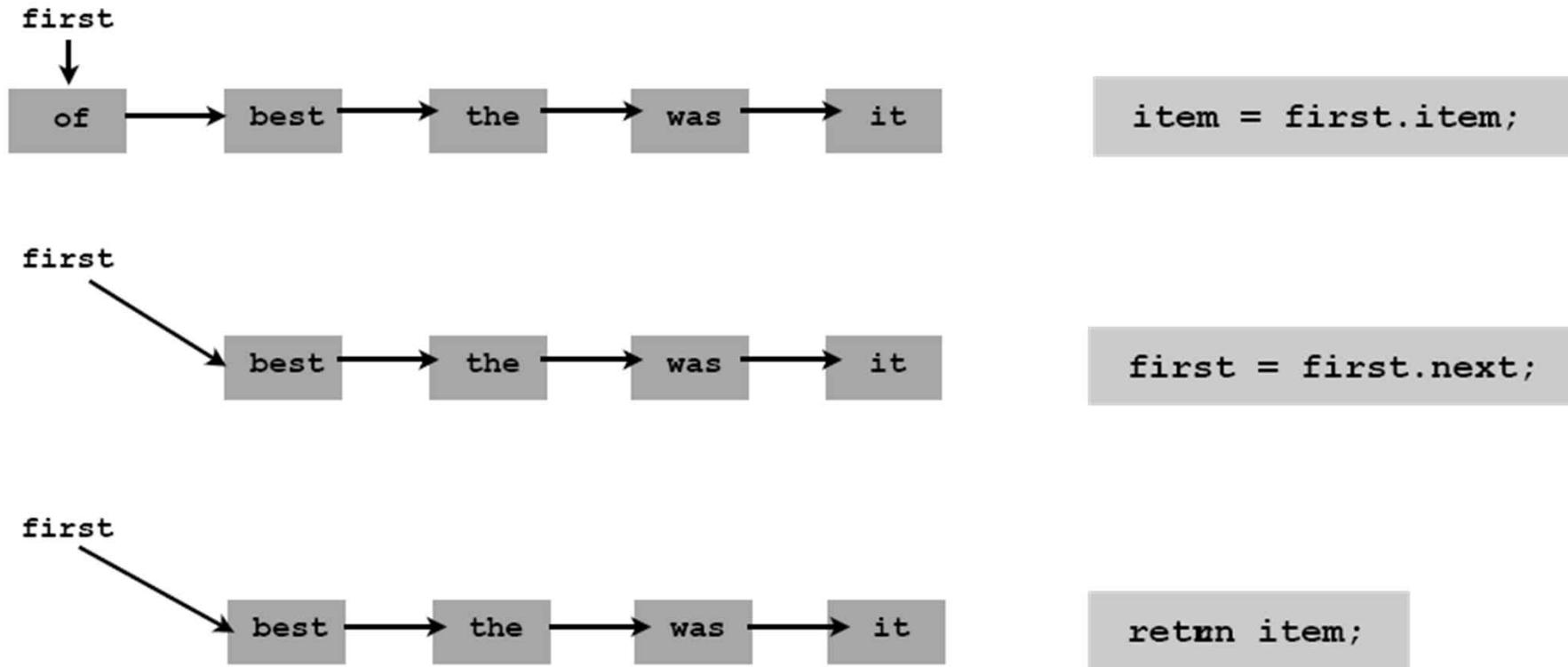
```
public static void main(String[] args)
{
    StackOfStrings stack = new StackOfStrings();
    while(!StdIn.isEmpty())
    {
        String s = StdIn.readString();
        stack.push(s);
    }
    while(!stack.isEmpty())
    {
        String s = stack.pop();
        StdOut.println(s);
    }
}
```

a sample stack client

Stack push: Linked-list implementation



Stack pop: Linked-list implementation



Stack: Linked-list implementation

```
public class StackOfStrings
{
    private Node first = null;

    private class Node
    {
        String item;           ← "inner class"
        Node next;
    }

    public boolean isEmpty()
    {   return first == null;   }

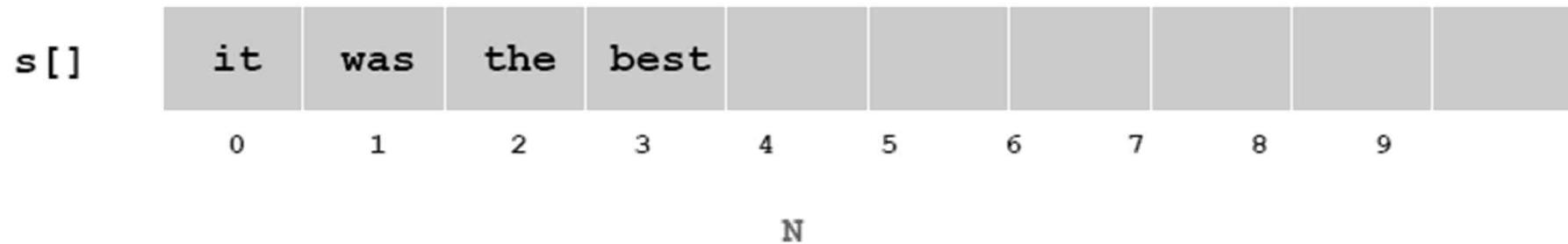
    public void push(String item)
    {
        Node second = first;
        first = new Node();
        first.item = item;
        first.next = second;
    }

    public String pop()
    {
        String item = first.item;
        first = first.next;
        return item;
    }
}
```

- Error conditions?
- Example:
 - ◆ `pop()` an empty stack

Stack: Array implementation

- Array implementation of a stack.
 - ◆ Use array $s[]$ to store N items on stack.
 - ◆ $\text{push}()$ add new item at $s[N]$.
 - ◆ $\text{pop}()$ remove item from $s[N-1]$.



Stack: Array implementation

```
public class StackOfStrings
{
    private String[] s;
    private int N = 0;

    public StringStack(int capacity)
    { s = new String[capacity]; }

    public boolean isEmpty()
    { return N == 0; }

    public void push(String item)
    { s[N++] = item; }

    public String pop()
    {
        String item = s[N-1];
        s[N-1] = null; ←
        N--;
        return item;
    }
}
```

avoid **loitering**
(garbage collector only reclaims memory
if no outstanding references)

2. Stacks & Queues

- stacks
- dynamic resizing
- queues
- generics
- applications

Stack array implementation: Dynamic resizing

- Q. How to grow array when capacity reached?
- Q. How to shrink array (else it stays big even when stack is small)?

- First try:
 - ◆ push(): increase size of `s[]` by 1
 - ◆ pop(): decrease size of `s[]` by 1
- Too expensive
 - ◆ Need to copy all of the elements to a new array.
 - ◆ Inserting N elements: time proportional to $1 + 2 + \dots + N \approx N^2/2$.
 - ↑
infeasible for large N
 - ◆ Need to *guarantee* that array resizing happens *infrequently*

Stack array implementation: Dynamic resizing

Q. How to grow array?

A. Use repeated doubling:

- ◆ if array is full, create a new array of twice the size, and copy items

no-argument
constructor

create new array
copy items to it

```
public StackOfStrings()
{ this(8); }

public void push(String item)
{
    if (N >= s.length) resize(2 * s.length);
    s[N++] = item;
}

private void resize(int max)
{
    String[] dup = new String[max];
    for (int i = 0; i < N; i++)
        dup[i] = s[i];
    s = dup;
}
```

- Consequence. Inserting N items takes time proportional to N
 - ◆ (not N^2).

$$8 + 16 + \dots + N/4 + N/2 + N \approx 2N$$

Stack array implementation: Dynamic resizing

Q. How (and when) to shrink array?

- How: create a new array of half the size, and copy items.
- When (first try): array is half full?
 - ◆ No, causes **thrashing**
 - ◆ **push-pop-push-pop...** sequence: time proportional to N for each op
- When (solution): array is $1/4$ full (then new array is half full).

```
public String pop(String item)
{
    String item = s[--N];
    s[N] = null;
    if (N == s.length/4)
        resize(s.length/2);
    return item;
}
```

□ Consequences.

- ◆ any sequence of N ops takes time proportional to N
- ◆ array is always between 25% and 100% full



Stack Implementations: Array vs. Linked List

- Stack implementation tradeoffs.
 - ◆ Can implement with either array or linked list, and client can use interchangeably. Which is better?
- Array.
 - ◆ Most operations take constant time.
 - ◆ Expensive doubling operation every once in a while.
 - ◆ Any sequence of N operations (starting from empty stack) takes time proportional to N .
- Linked list.
 - ◆ Grows and shrinks gracefully.
 - ◆ Every operation takes constant time.
 - ◆ Every operation uses extra space and time to deal with references.
- Bottom line:
 - ◆ tossup for stacks **but** differences are significant when other operations are added

“amortized” bound



Stack implementations: Array vs. Linked list

□ Which implementation is more convenient?

array?

linked list?

- ◆ return count of elements in stack
- ◆ remove the kth most recently added
- ◆ sample a random element

2. Stacks & Queues

- stacks
- dynamic resizing
- queues
- generics
- applications

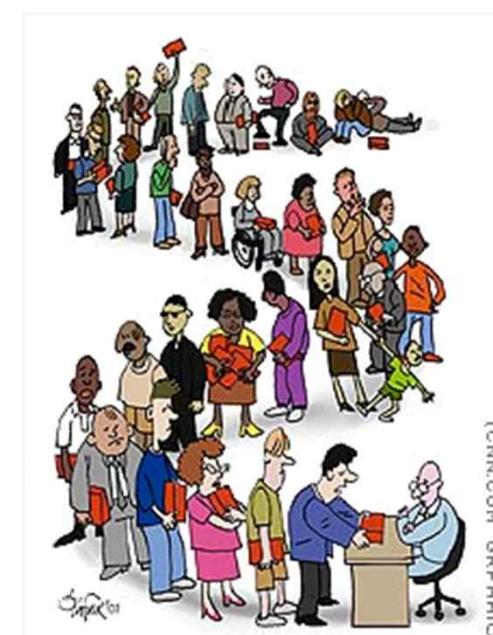
Queues

□ Queue operations.

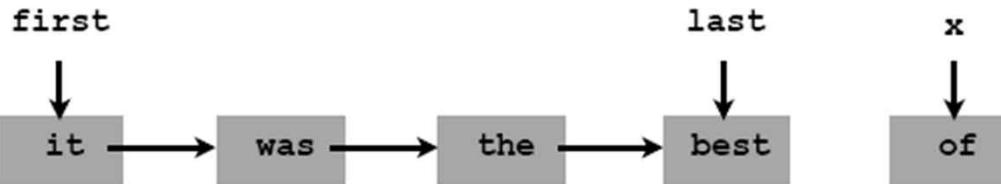
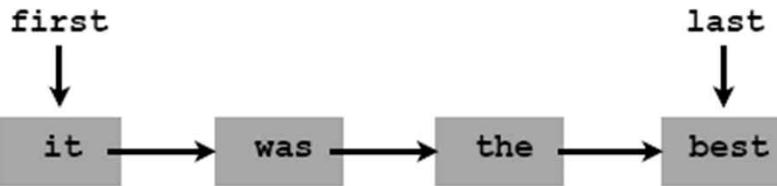
- ◆ enqueue () Insert a new item onto queue.
- ◆ dequeue () Delete and return the item least recently added.
- ◆ isEmpty () Is the queue empty?

```
public static void main(String[] args)
{
    QueueOfStrings q = new QueueOfStrings();
    q.enqueue("Vertigo");
    q.enqueue("Just Lose It");
    q.enqueue("Pieces of Me");
    q.enqueue("Pieces of Me");
    System.out.println(q.dequeue());
    q.enqueue("Drop It Like It's Hot");

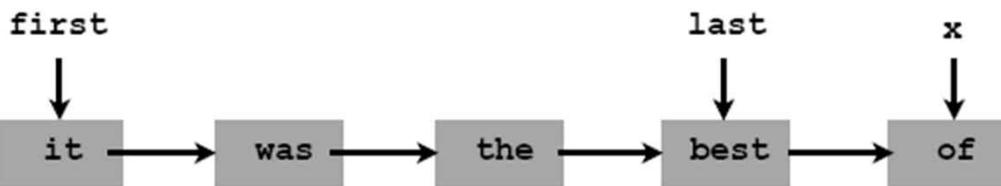
    while(!q.isEmpty())
        System.out.println(q.dequeue());
}
```



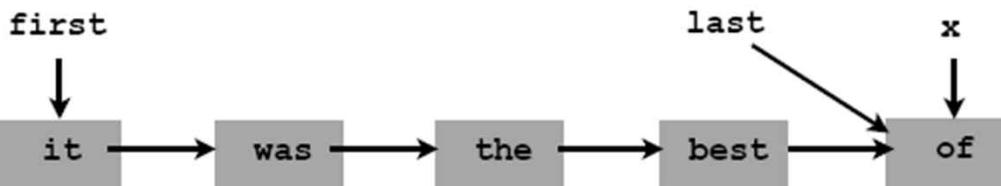
Enqueue: Linked List Implementation



```
x = new Node();
x.item = item;
x.next = null;
```

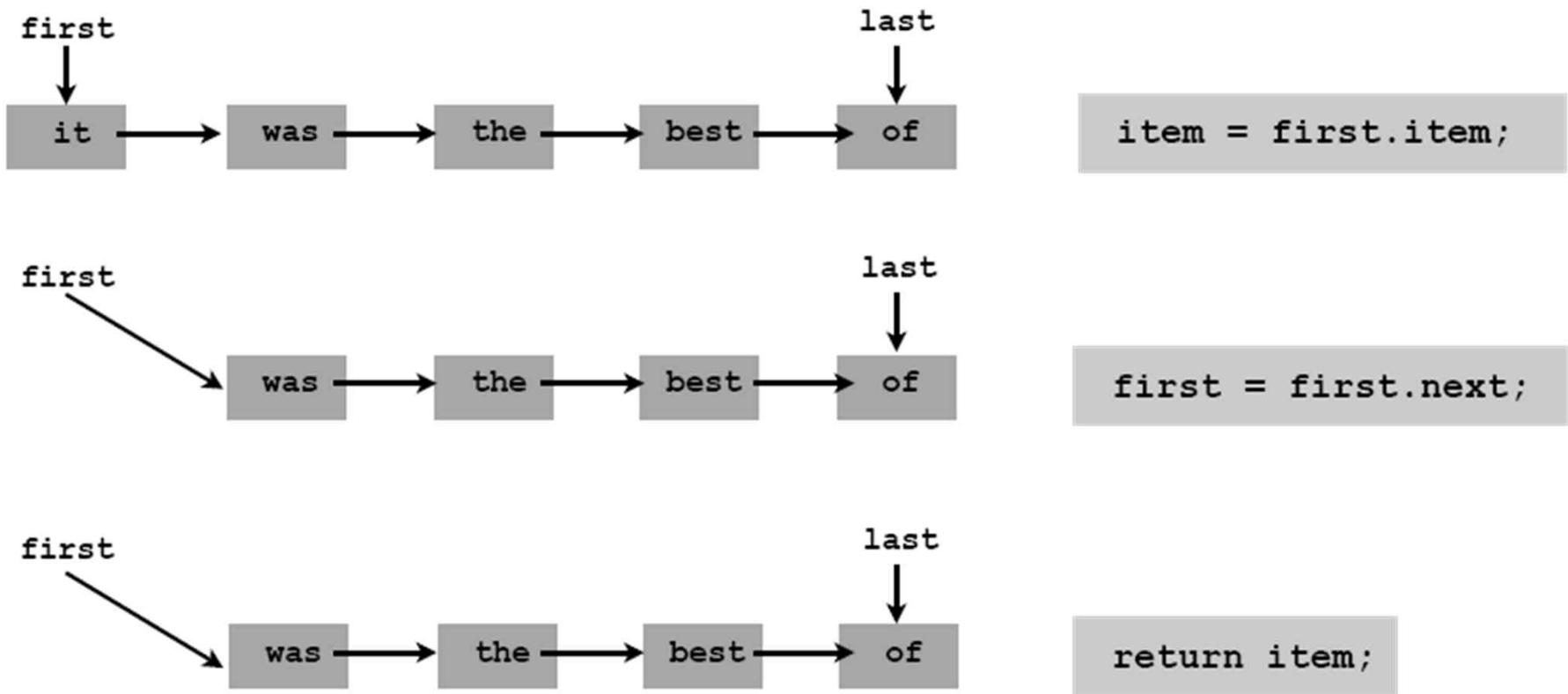


```
last.next = x;
```



```
last = x;
```

Dequeue: Linked List Implementation



❑ Aside:

- ◆ **dequeue** (pronounced "DQ") means "**remove from a queue**"
- ◆ **deque** (pronounced "deck") is a **data structure** (see PA 1)

Queue: Linked List Implementation

```
public class QueueOfStrings
{
    private Node first;
    private Node last;

    private class Node
    { String item; Node next; }

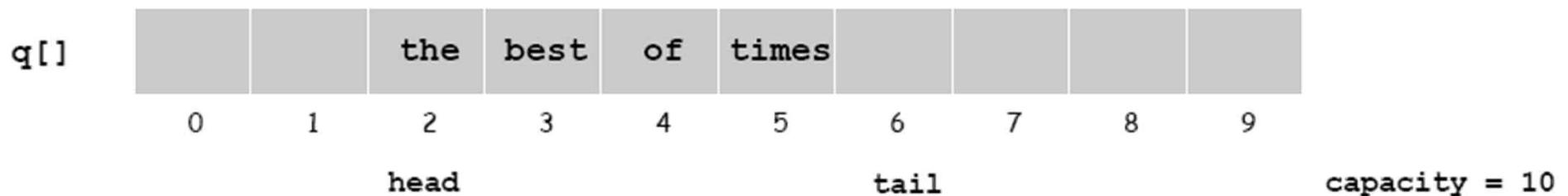
    public boolean isEmpty()
    { return first == null; }

    public void enqueue(String item)
    {
        Node x = new Node();
        x.item = item;
        x.next = null;
        if (isEmpty()) { first = x; last = x; }
        else           { last.next = x; last = x; }
    }

    public String dequeue()
    {
        String item = first.item;
        first = first.next;
        return item;
    }
}
```

Queue: Array implementation

- Array implementation of a queue.
 - ◆ Use array $q[]$ to store items on queue.
 - ◆ $\text{enqueue}()$: add new object at $q[\text{tail}]$.
 - ◆ $\text{dequeue}()$: remove object from $q[\text{head}]$.
 - ◆ Update head and tail modulo the capacity.



2. Stacks & Queues

- stacks
- dynamic resizing
- queues
- *generics*
- applications

Generics (parameterized data types)

- We implemented: StackOfStrings, QueueOfStrings.
- We also want: StackOfURLs, QueueOfCustomers, etc?
- Attempt 1. Implement a separate stack class for each type.
 - ◆ Rewriting code is tedious and error-prone.
 - ◆ Maintaining cut-and-pasted code is tedious and error-prone.
- @#\$*! most reasonable approach until Java 1.5

Stack of Objects

- We implemented: StackOfStrings, QueueOfStrings.
- We also want: StackofURLs, QueueofCustomers, etc?
- Attempt 2. Implement a stack with items of type Object.
 - ◆ Casting is required in client.
 - ◆ Casting is error-prone: run-time error if types mismatch.

```
Stack s = new Stack();
Apple a = new Apple();
Orange b = new Orange();
s.push(a);
s.push(b);
a = (Apple) (s.pop());
```

run-time error

Generics

- Generics. Parameterize stack by a single type.
 - ◆ Avoid casting in both client and implementation.
 - ◆ Discover type mismatch errors at **compile-time** instead of run-time.

```
Stack<Apple> s = new Stack<Apple>();  
Apple a = new Apple();  
Orange b = new Orange();  
s.push(a);  
s.push(b);           compile-time error  
a = s.pop();
```

parameter

no cast needed in client

- Guiding principles.
 - ◆ Welcome compile-time errors
 - ◆ Avoid run-time errors
- Why?



Generic Stack: Linked List Implementation

```
public class StackOfStrings
{
    private Node first = null;

    private class Node
    {
        String item;
        Node next;
    }

    public boolean isEmpty()
    { return first == null; }

    public void push(String item)
    {
        Node second = first;
        first = new Node();
        first.item = item;
        first.next = second;
    }

    public String pop()
    {
        String item = first.item;
        first = first.next;
        return item;
    }
}
```

```
public class Stack<Item>
{
    private Node first = null;

    private class Node
    {
        Item item;
        Node next;
    }

    public boolean isEmpty()
    { return first == null; }

    public void push(Item item)
    {
        Node second = first;
        first = new Node();
        first.item = item;
        first.next = second;
    }

    public Item pop()
    {
        Item item = first.item;
        first = first.next;
        return item;
    }
}
```

Generic type name

Generic stack: array implementation

- The way it should be.

```
public class Stack<Item>
{
    private Item[] s;
    private int N = 0;

    public Stack(int cap)
    { s = new Item[cap]; }

    public boolean isEmpty()
    { return N == 0; }

    public void push(Item item)
    { s[N++] = item; }

    public String pop()
    {
        Item item = s[N-1];
        s[N-1] = null;
        N--;
        return item;
    }
}
```

```
public class StackOfStrings
{
    private String[] s;
    private int N = 0;

    public StackOfStrings(int cap)
    { s = new String[cap]; }

    public boolean isEmpty()
    { return N == 0; }

    public void push(String item)
    { s[N++] = item; }

    public String pop()
    {
        String item = s[N-1];
        s[N-1] = null;
        N--;
        return item;
    }
}
```

@#\$*! generic array creation not allowed in Java

Generic stack: array implementation

- The way it is: an ugly cast in the implementation.

```
public class Stack<Item>
{
    private Item[] s;
    private int N = 0;

    public Stack(int cap)
    { s = (Item[]) new Object[cap]; } ← the ugly cast

    public boolean isEmpty()
    { return N == 0; }

    public void push(Item item)
    { s[N++] = item; }

    public String pop()
    {
        Item item = s[N-1];
        s[N-1] = null;
        N--;
        return item;
    }
}
```

- Number of casts in good code: 0

Generic data types: autoboxing

- Generic stack implementation is object-based.
- What to do about primitive types?
- Wrapper type.
 - ◆ Each primitive type has a **wrapper** object type.
 - ◆ Ex: Integer is wrapper type for int.
- Autoboxing.
 - ◆ Automatic cast between a primitive type and its wrapper.
 - ◆ Syntactic sugar. Behind-the-scenes casting.

```
Stack<Integer> s = new Stack<Integer>();  
s.push(17);           // s.push(new Integer(17));  
int a = s.pop();    // int a = ((int) s.pop()).intValue();
```

- Bottom line: Client code can use generic stack for **any** type of data



2. Stacks & Queues

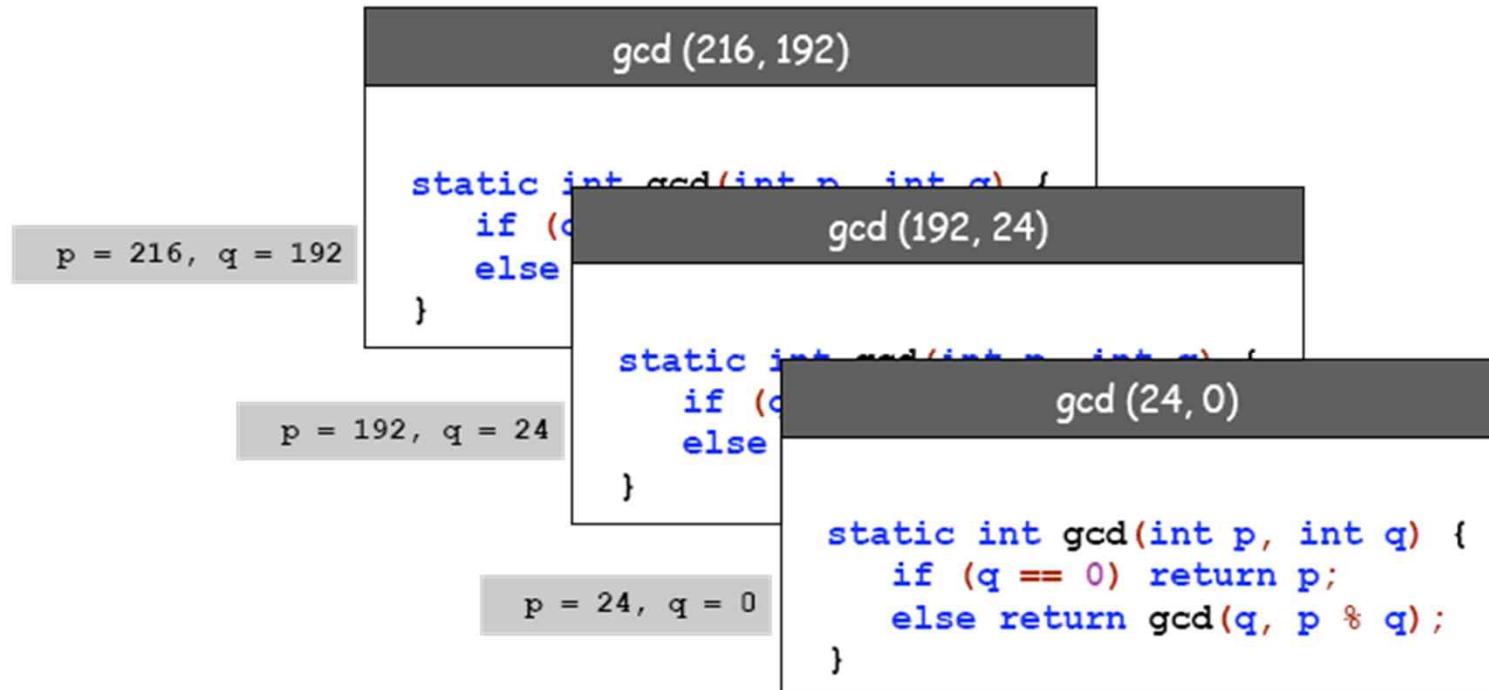
- stacks
- dynamic resizing
- queues
- generics
- applications

Stack Applications

- Real world applications.
 - ◆ Parsing in a compiler.
 - ◆ Java virtual machine.
 - ◆ Undo in a word processor.
 - ◆ Back button in a Web browser.
 - ◆ PostScript language for printers.
 - ◆ Implementing function calls in a compiler.

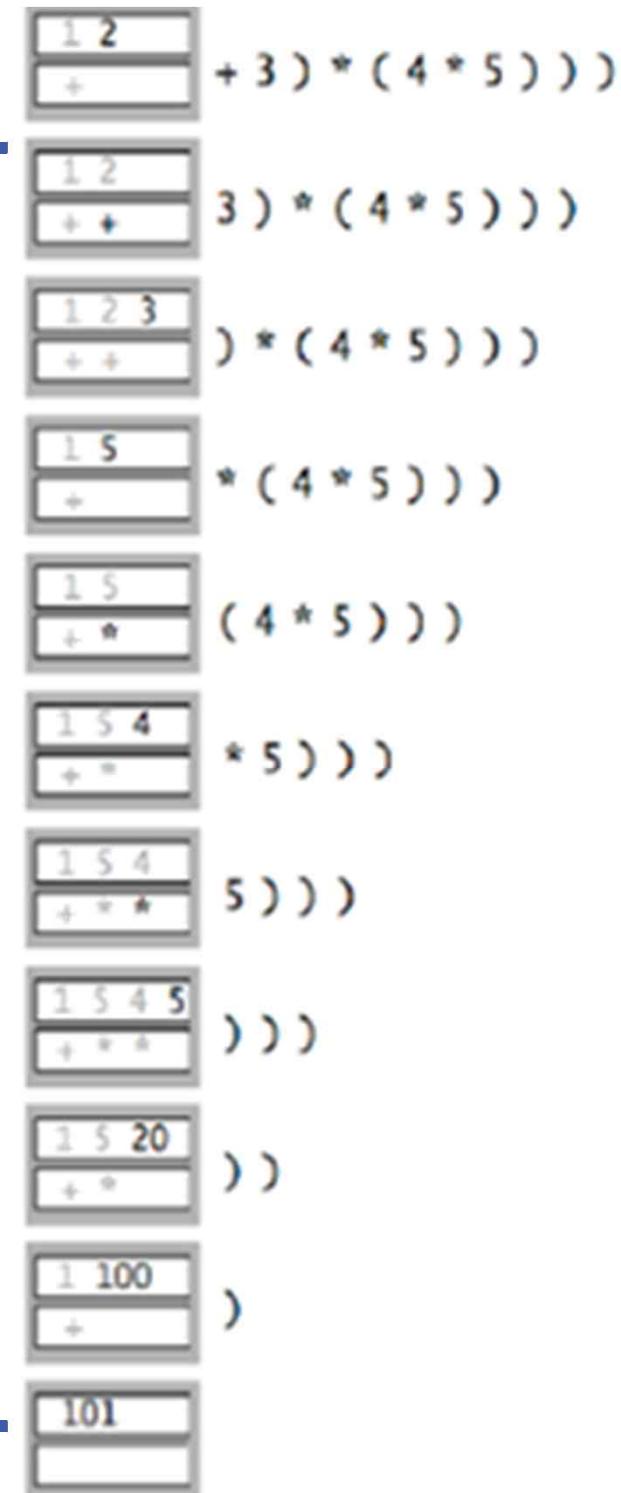
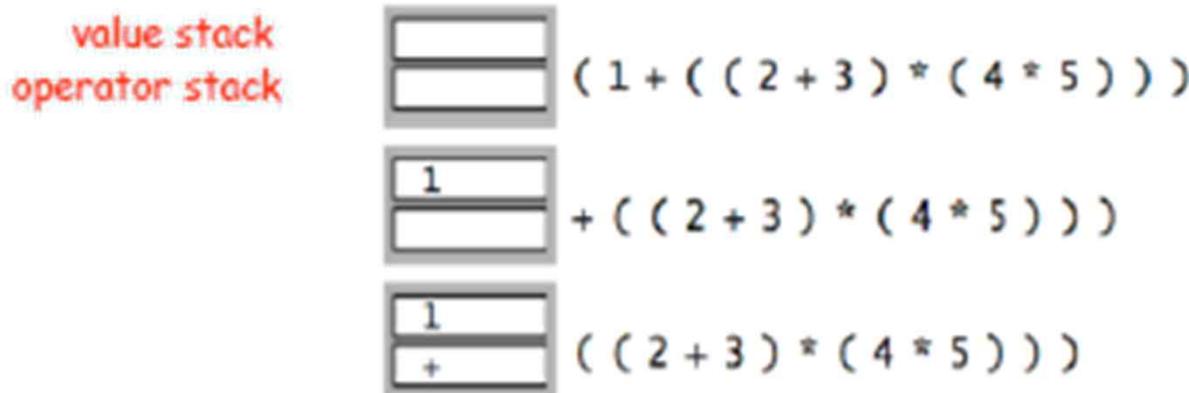
Function Calls

- How a compiler implements functions.
 - ◆ Function call: **push** local environment and return address.
 - ◆ Return: **pop** return address and local environment.
- Recursive function. Function that calls itself.
- Note. Can always use an explicit stack to remove recursion.



Arithmetic Expression Evaluation

- Goal. Evaluate infix expressions.
- Two-stack algorithm. [E. W. Dijkstra]
 - ◆ Value: push onto the value stack.
 - ◆ Operator: push onto the operator stack.
 - ◆ Left parens: ignore.
 - ◆ Right parens: pop operator and two values; push the result of applying that operator to those values onto the operand stack.
- Context. An interpreter!



Arithmetic Expression Evaluation

```
public class Evaluate {  
    public static void main(String[] args) {  
        Stack<String> ops = new Stack<String>();  
        Stack<Double> vals = new Stack<Double>();  
        while (!StdIn.isEmpty()) {  
            String s = StdIn.readString();  
            if (s.equals("(")) ;  
            else if (s.equals("+")) ops.push(s);  
            else if (s.equals("*")) ops.push(s);  
            else if (s.equals(")")) {  
                String op = ops.pop();  
                if (op.equals("+")) vals.push(vals.pop() + vals.pop());  
                else if (op.equals("*")) vals.push(vals.pop() * vals.pop());  
            }  
            else vals.push(Double.parseDouble(s));  
        }  
        StdOut.println(vals.pop());  
    }  
}
```

```
% java Evaluate  
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )  
101.0
```

- ❑ Note: Old books have two-pass algorithm because generics were not available!



Correctness

□ Why correct?

- ◆ When algorithm encounters an operator surrounded by two values within parentheses, it leaves the result on the value stack.

✓ $(1 + ((2 + 3) * (4 * 5)))$

- ◆ as if the original input were:

✓ $(1 + (5 * (4 * 5)))$

- ◆ Repeating the argument:

✓ $(1 + (5 * 20))$

✓ $(1 + 100)$

✓ 101

□ Extensions. More ops, precedence order, associativity.

✓ $1 + (2 - 3 - 4) * 5 * \text{sqrt}(6 + 7)$

Stack-based programming languages

□ Observation 1.

Remarkably, the 2-stack algorithm computes the same value if the **operator** occurs **after** the two **values**.

```
( 1 ( ( 2 3 + ) ( 4 5 * ) * ) + )
```

□ Observation 2.

All of the parentheses are redundant!

```
1 2 3 + 4 5 * * +
```

□ Bottom line. Postfix or "reverse Polish" notation.

□ Applications. Postscript, Forth, calculators, Java virtual machine, ...

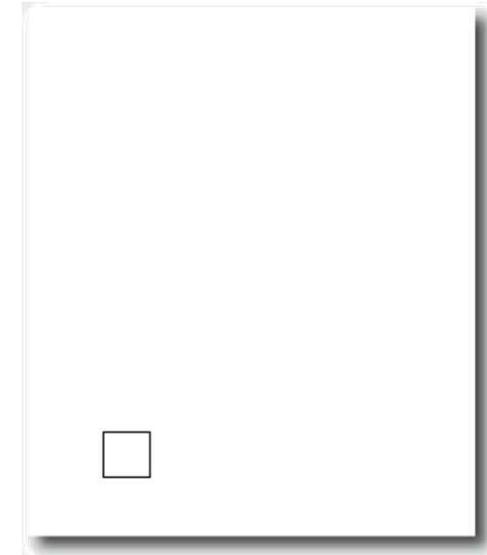
Stack-based programming languages: PostScript

□ Page description language

- ◆ explicit stack
- ◆ full computational model
- ◆ graphics engine

□ Basics

- ◆ %!: "I am a PostScript program"
- ◆ literal: "push me on the stack"
- ◆ function calls take args from stack
- ◆ turtle graphics built in



a PostScript program

```
%!  
72 72 moveto  
0 72 rlineto  
72 0 rlineto  
0 -72 rlineto  
-72 0 rlineto  
2 setlinewidth  
stroke
```

Stack-based programming languages: PostScript

□ Data types

- ◆ basic: integer, floating point, boolean, ...
- ◆ graphics: font, path,
- ◆ full set of built-in operators

□ Text and strings

- ◆ full font support like System.out.print()
- ◆ show (display a string, using current font)
- ◆ cvs (convert anything to a string) like toString()

Square root of 2:
1.4142

```
%!  
/Helvetica-Bold findfont 16 scalefont setfont  
72 168 moveto  
(Square root of 2:) show  
72 144 moveto  
2 sqrt 10 string cvs show
```

Stack-based programming languages: PostScript

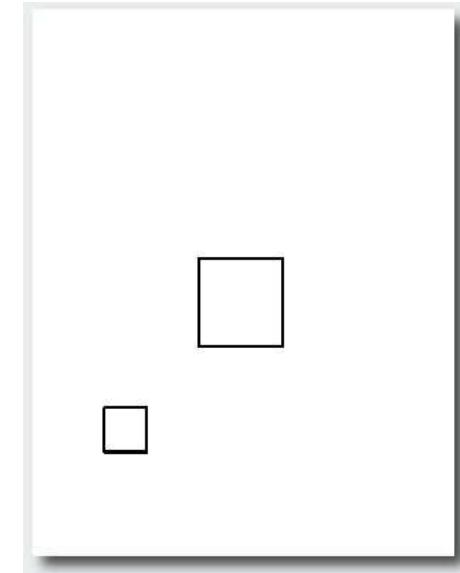
□ Variables (and functions)

- ◆ identifiers start with /
- ◆ def operator associates id with value
- ◆ Braces
- ◆ args on stack

```
%!  
function definition → /box  
{  
    /sz exch def  
    0 sz rlineto  
    sz 0 rlineto  
    0 sz neg rlineto  
    sz neg 0 rlineto  
} def
```

```
72 144 moveto  
72 box  
288 288 moveto  
144 box  
2 setlinewidth  
stroke
```

function calls



Stack-based programming languages: PostScript

□ for loop

- ◆ "from, increment, to" on stack
- ◆ loop body in braces
- ◆ for operator

```
1 1 20
{ 19 mul dup 2 add moveto 72 box }
for
```

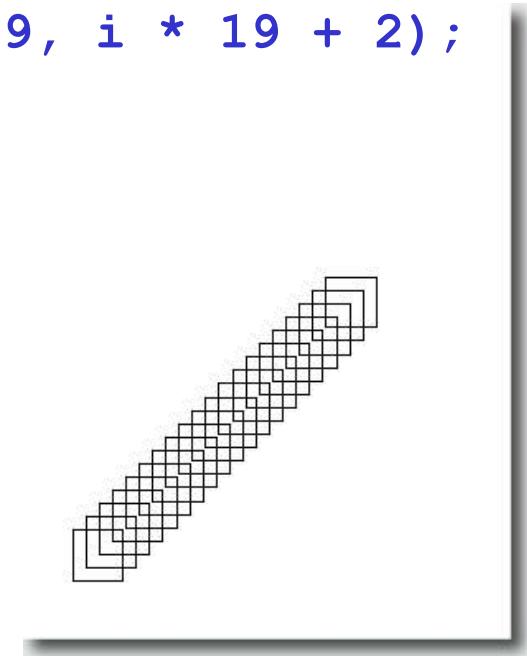
```
for(i = 0; i <= 20; i++) {
    moveto(i * 19, i * 19 + 2);
    box(72);
```

```
}
```

□ if-else

- ◆ boolean on stack
- ◆ alternatives in braces
- ◆ if operator

... (hundreds of operators)



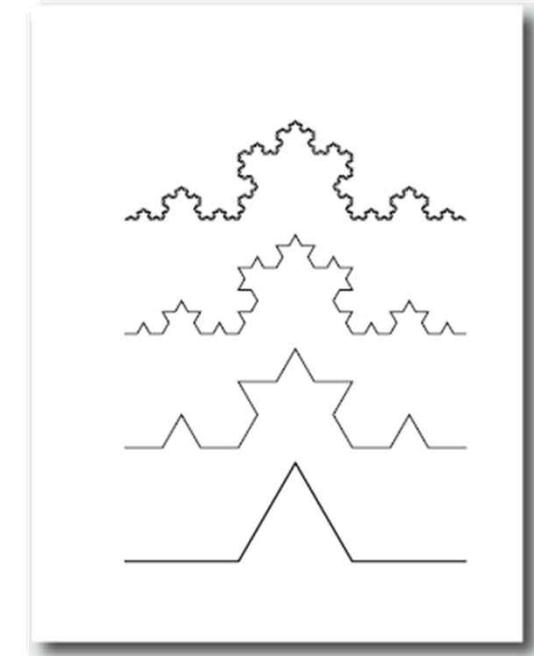
Stack-based programming languages: PostScript

An application: all figures in *Algorithms in Java*

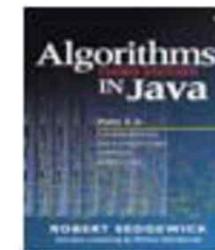
```
%!
72 72 translate

/kochR
{
    2 copy ge { dup 0 rlineto }
    {
        3 div
        2 copy kochR 60 rotate
        2 copy kochR -120 rotate
        2 copy kochR 60 rotate
        2 copy kochR
    } ifelse
    pop pop
} def

0 0 moveto 81 243 kochR
0 81 moveto 27 243 kochR
0 162 moveto 9 243 kochR
0 243 moveto 1 243 kochR
stroke
```



See page 218



Queue applications

- Familiar applications.
 - ◆ iTunes playlist.
 - ◆ Data buffers (iPod, TiVo).
 - ◆ Asynchronous data transfer (file IO, pipes, sockets).
 - ◆ Dispensing requests on a shared resource (printer, processor).
- Simulations of the real world.
 - ◆ Traffic analysis.
 - ◆ Waiting times of customers at call center.
 - ◆ Determining number of cashiers to have at a supermarket.

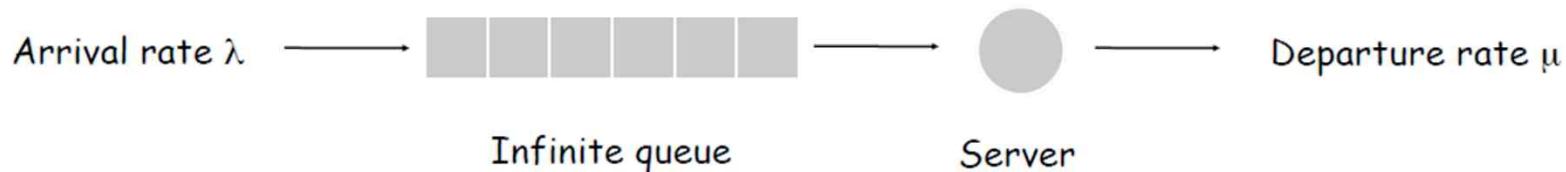
M/D/1 queuing model

□ M/D/1 queue.

- ◆ Customers are serviced at fixed rate of μ per minute.
- ◆ Customers arrive according to Poisson process at rate of λ per minute.

inter-arrival time has exponential distribution

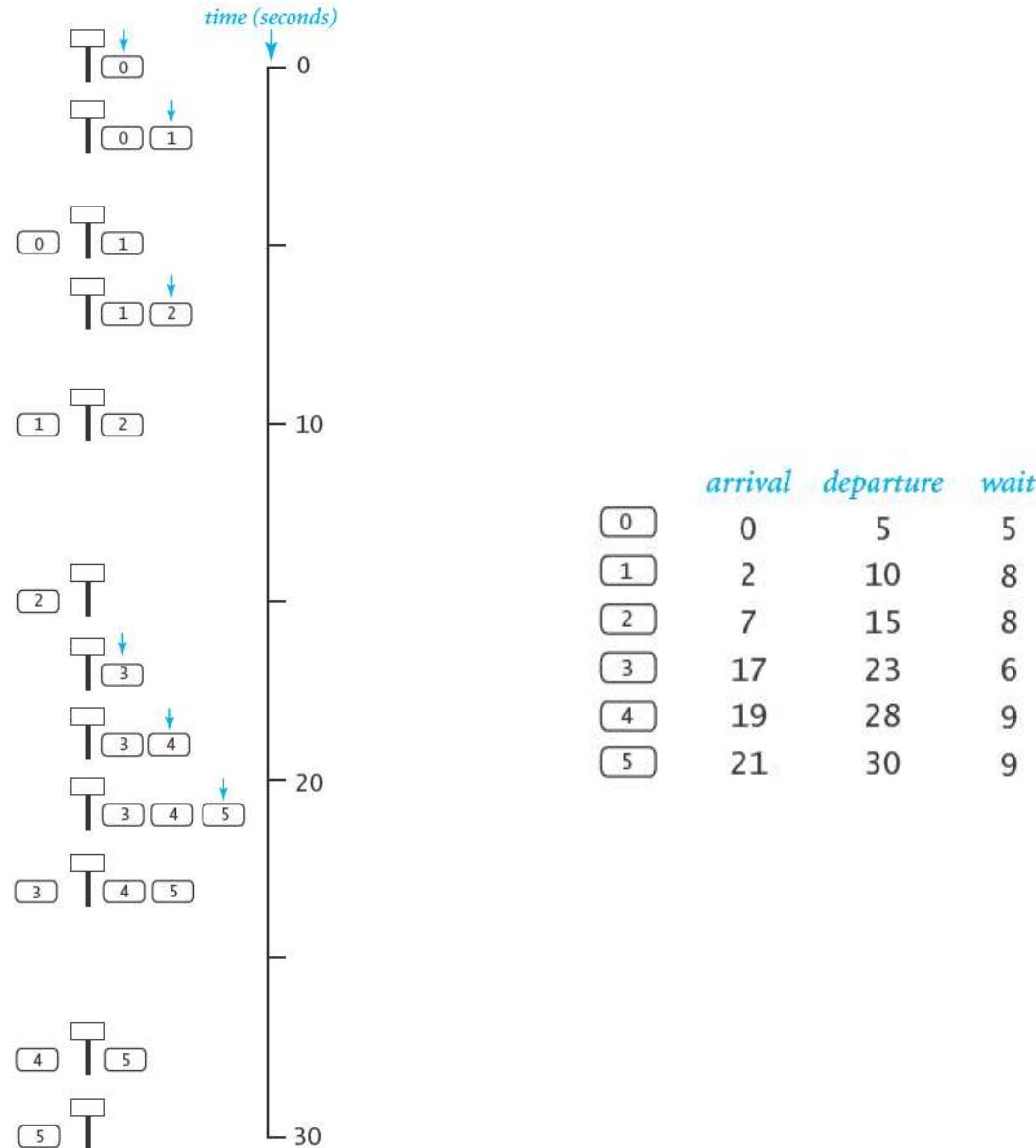
$$\Pr[X \leq x] = 1 - e^{-\lambda x}$$



Q. What is average wait time W of a customer?

Q. What is average number of customers L in system?

M/D/1 queuing model: example

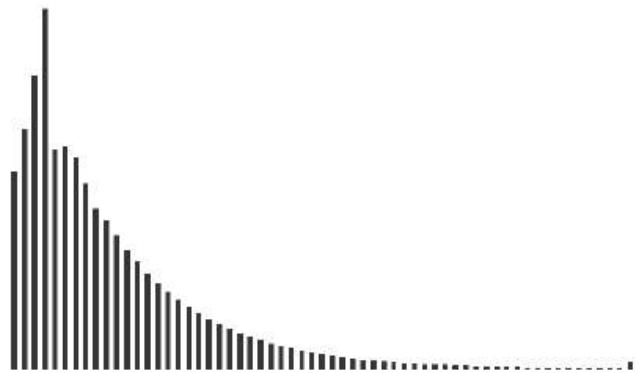


M/D/1 queuing model: experiments and analysis

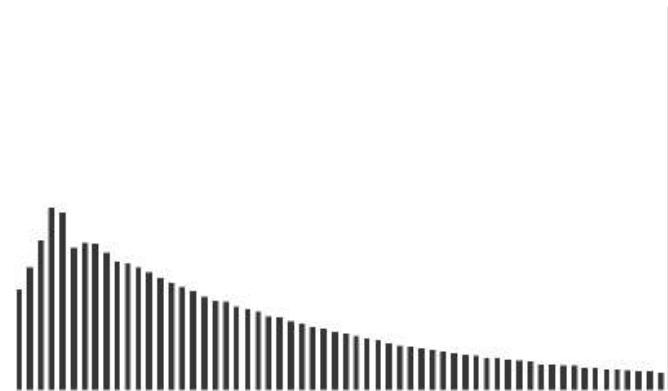
□ Observation.

- ◆ As service rate μ approaches arrival rate λ , service goes to hell***.

```
% java MD1Queue .167 .25
```



```
% java MD1Queue .167 .22
```



□ Queueing theory (see ORFE 309).

$$W = \frac{\lambda}{2\mu(\mu-\lambda)} + \frac{1}{\mu}, \quad L = \lambda W$$

Little's Law

wait time W and queue length L approach infinity as service rate approaches arrival rate

M/D/1 queuing model: event-based simulation

```
public class MD1Queue
{
    public static void main(String[] args)
    {
        double lambda = Double.parseDouble(args[0]);      // arrival rate
        double mu     = Double.parseDouble(args[1]);      // service rate
        Histogram hist = new Histogram(60);
        Queue<Double> q = new Queue<Double>();
        double nextArrival = StdRandom.exp(lambda);
        double nextService = 1/mu;
        while (true)
        {
            while (nextArrival < nextService)
            {
                q.enqueue(nextArrival);
                nextArrival += StdRandom.exp(lambda);
            }
            double wait = nextService - q.dequeue();
            hist.addDataPoint(Math.min(60, (int) (wait)));
            if (!q.isEmpty())
                nextService = nextArrival + 1/mu;
            else
                nextService = nextService + 1/mu;
        }
    }
}
```

Data Structures & Algorithms

3. Sorting Algorithms

- rules of the game
- shellsort
- mergesort
- quicksort
- animations



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

Classic sorting algorithms

- Critical components in the world's computational infrastructure.
 - ◆ Full scientific understanding of their properties has enabled us to develop them into practical system sorts.
 - ◆ Quicksort honored as one of top 10 algorithms of 20th century in science and engineering.
- Shellsort.
 - ◆ Warmup: easy way to break the N^2 barrier.
 - ◆ Embedded systems.
- Mergesort.
 - ◆ Java sort for objects.
 - ◆ Perl, Python stable sort.
- Quicksort.
 - ◆ Java sort for primitive types.
 - ◆ C qsort, Unix, g++, Visual C++, Python.



3. Sorting Algorithms

- rules of the game
- shellsort
- mergesort
- quicksort
- animations

Basic terms

- Ex: student record in a University.

file →

record →

key →

FOX	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

- Sort: rearrange sequence of objects into ascending order.

Aaron	4	A	664-480-0023	097 Little
Andrews	3	A	874-088-1212	121 Whitman
Battle	4	C	991-878-4944	308 Blair
Chen	2	A	884-232-5341	11 Dickinson
FOX	1	A	243-456-9091	101 Brown
Furia	3	A	766-093-9873	22 Brown
Gazsi	4	B	665-303-0266	113 Walker
Kanaga	3	B	898-122-9643	343 Forbes
Rohde	3	A	232-343-5555	115 Holder
Quilici	1	C	343-987-5642	32 McCosh

Sample sort client

- Goal: Sort any type of data
- Example. List the files in the current directory, sorted by file name.

```
import java.io.File;
public class Files
{
    public static void main(String[] args)
    {
        File directory = new File(args[0]);
        File[] files = directory.listFiles();
        Insertion.sort(files);
        for (int i = 0; i < files.length; i++)
            System.out.println(files[i]);
    }
}
```

```
% java Files .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
ShellX.class
ShellX.java
index.html
```

Next: How does sort compare file names?

Callbacks

- Goal. Write robust sorting library method that can sort any type of data using the data type's natural order.
- Callbacks.
 - ◆ Client passes array of objects to sorting routine.
 - ◆ Sorting routine calls back object's comparison function as needed.
- Implementing callbacks.
 - ◆ Java: interfaces.
 - ◆ C: function pointers.
 - ◆ C++: functors (classes used like functions).



Callbacks

client

```
import java.io.File;
public class SortFiles
{
    public static void main(String[] args)
    {
        File directory = new File(args[0]);
        File[] files = directory.listFiles();
        Insertion.sort(files);
        for (int i = 0; i < files.length; i++)
            System.out.println(files[i]);
    }
}
```

interface

```
interface Comparable <Item>
{
    public int compareTo(Item);
}
```

Built in to Java

Key point: no reference to File →

object implementation

```
public class File
implements Comparable<File>
{
    ...
    public int compareTo(File b)
    {
        ...
        return -1;
        ...
        return +1;
        ...
        return 0;
    }
}
```

sort implementation

```
public static void sort(Comparable[] a)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0; j--)
            if (a[j].compareTo(a[j-1]) < 0)
                exch(a, j, j-1);
            else break;
}
```

Callbacks

- Goal: Write robust sorting library that can sort any type of data into sorted order using the client's comparison routine.

Callbacks

- ◆ Client passes array of objects to sorting routine
- ◆ Sorting routine calls back object's comparison function as needed

Implementing callbacks

- ◆ Java interfaces
- ◆ C++ function pointers
- ◆ C# delegates

- Plus: Code reuse for all types of data
- Minus: Significant overhead in inner loop

This course:

- ◆ enables focus on algorithm implementation
- ◆ use **same code** for experiments, real-world data



Interface specification for sorting

- Comparable interface.
- Must implement method `compareTo()` so that `v.compareTo(w)` returns:
 - ◆ a negative integer if `v` is less than `w`
 - ◆ a positive integer if `v` is greater than `w`
 - ◆ zero if `v` is equal to `w`
- Consistency.
- Implementation must ensure a total order.
 - ◆ if $(a < b)$ and $(b < c)$, then $(a < c)$.
 - ◆ either $(a < b)$ or $(b < a)$ or $(a = b)$.
- Built-in comparable types. String, Double, Integer, Date, File.
- User-defined comparable types. Implement the Comparable interface.

Implementing the Comparable interface: example 1

Date data type (simplified version of built-in Java code)

```
public class Date implements Comparable<Date>
{
    private int month, day, year;

    public Date(int m, int d, int y)
    {
        month = m;
        day   = d;
        year  = y;
    }

    public int compareTo(Date b)
    {
        Date a = this;
        if (a.year < b.year) return -1;
        if (a.year > b.year) return +1;
        if (a.month < b.month) return -1;
        if (a.month > b.month) return +1;
        if (a.day < b.day) return -1;
        if (a.day > b.day) return +1;
        return 0;
    }
}
```

only compare dates to other dates

Implementing the Comparable interface: example 2

□ Domain names

- ◆ Subdomain: sdtlab.inu.ac.kr.
- ◆ Reverse subdomain: kr.ac.inu.sdtlab.
- ◆ Sort by reverse subdomain to **group by category**.

```
public class Domain implements Comparable<Domain>
{
    private String[] fields;
    private int N;
    public Domain(String name)
    {
        fields = name.split("\\.");
        N = fields.length;
    }
    public int compareTo(Domain b)
    {
        Domain a = this;
        for (int i = 0; i < Math.min(a.N, b.N); i++)
        {
            int c = a.fields[i].compareTo(b.fields[i]);
            if (c < 0) return -1;
            else if (c > 0) return +1;
        }
        return a.N - b.N;
    }
}
```

details included for the bored...

unsorted
ee.princeton.edu
cs.princeton.edu
princeton.edu
cnn.com
google.com
apple.com
<u>www.cs.princeton.edu</u>
bolle.cs.princeton.edu

sorted
com.apple
com.cnn
com.google
edu.princeton
edu.princeton.cs
edu.princeton.cs.bolle
edu.princeton.cs.www
edu.princeton.ee

Sample sort clients

File names

```
import java.io.File;
public class Files
{
    public static void main(String[] args)
    {
        File directory = new File(args[0]);
        File[] files = directory.listFiles()
        Insertion.sort(files);
        for (int i = 0; i < files.length; i++)
            System.out.println(files[i]);
    }
}
```

```
% java Files .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
```

Several Java library data types implement Comparable
You can implement Comparable for your own types

Random numbers

```
public class Experiment
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        Double[] a = new Double[N];
        for (int i = 0; i < N; i++)
            a[i] = Math.random();
        Selection.sort(a);
        for (int i = 0; i < N; i++)
            System.out.println(a[i]);
    }
}
```

```
% java Experiment 10
0.08614716385210452
0.09054270895414829
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

Two useful abstractions

- Helper functions. Refer to data only through two operations.

- ◆ **less**. Is v less than w ?

```
private static boolean less(Comparable v, Comparable w)
{
    return (v.compareTo(w) < 0);
}
```

- ◆ **exchange**. Swap object in array at index i with the one at index j .

```
private static void exch(Comparable[] a, int i, int j)
{
    Comparable t = a[i];
    a[i] = a[j];
    a[j] = t;
}
```

Sample sort implementations

```
selection sort    public class Selection
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
        {
            int min = i;
            for (int j = i+1; j < N; j++)
                if (less(a, j, min)) min = j;
            exch(a, i, min);
        }
    ...
}
```



```
insertion sort   public class Insertion
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 1; i < N; i++)
        for (int j = i; j > 0; j--)
            if (less(a[j], a[j-1]))
                exch(a, j, j-1);
            else break;
    ...
}
```



Why use less() and exch() ?

- Switch to faster implementation for primitive types

```
private static boolean less(double v, double w)
{
    return v < w;
}
```

- Instrument for experimentation and animation

```
private static boolean less(double v, double w)
{
    cnt++;
    return v < w;
}
```

- Translate to other languages

```
...
for (int i = 1; i < a.length; i++)
    if (less(a[i], a[i-1]))
        return false;
return true;
```

Good code in C, C++,
JavaScript, Ruby....

Properties of elementary sorts (review)

Selection sort

		a[i]										
i	min	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
0	6	S	O	R	T	E	X	A	M	P	L	E
1	4	A	O	R	T	E	X	S	M	P	L	E
2	10	A	E	R	T	O	X	S	M	P	L	E
3	9	A	E	E	T	O	X	S	M	P	L	R
4	7	A	E	E	L	O	X	S	M	P	T	R
5	7	A	E	E	L	M	X	S	O	P	T	R
6	8	A	E	E	L	M	O	S	X	P	T	R
7	10	A	E	E	L	M	O	P	X	S	T	R
8	8	A	E	E	L	M	O	P	R	S	T	X
9	9	A	E	E	L	M	O	P	R	S	T	X
10	10	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

Running time: Quadratic ($\sim c N^2$)

Exception: expensive exchanges
(could be linear)

Insertion sort

		a[i]										
i	j	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
1	0	O	S	R	T	E	X	A	M	P	L	E
2	1	O	R	S	T	E	X	A	M	P	L	E
3	3	O	R	S	T	E	X	A	M	P	L	E
4	0	E	O	R	S	T	X	A	M	P	L	E
5	5	E	O	R	S	T	X	A	M	P	L	E
6	0	A	E	O	R	S	T	X	M	P	L	E
7	2	A	E	M	O	R	S	T	X	P	L	E
8	4	A	E	M	O	P	R	S	T	X	L	E
9	2	A	E	L	M	O	P	R	S	T	X	E
10	2	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

Running time: Quadratic ($\sim c N^2$)

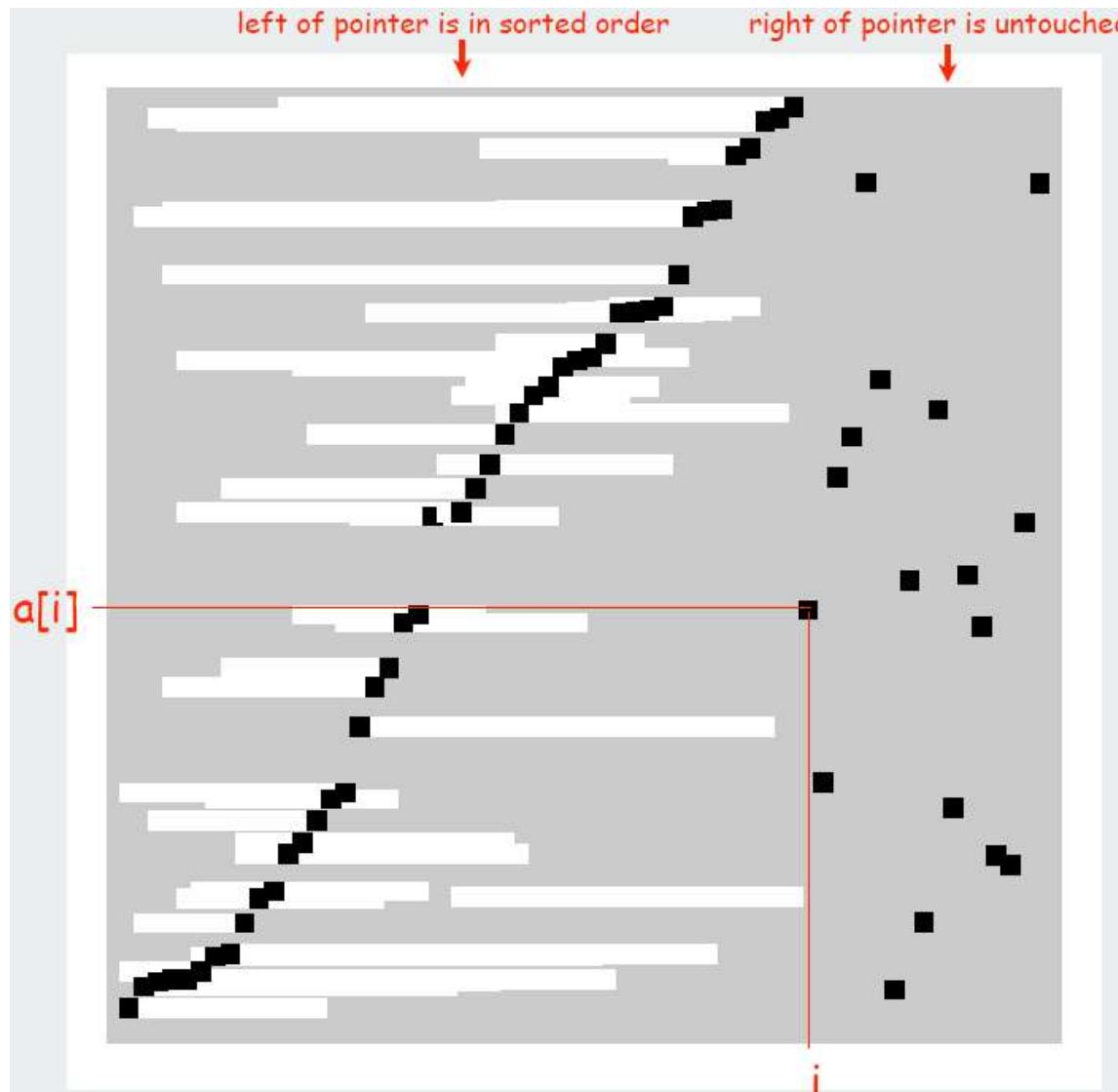
Exception: input nearly in order
(could be linear)

- Bottom line: both are quadratic (too slow) for large randomly ordered files

3. Sorting Algorithms

- rules of the game
- shellsort
- mergesort
- quicksort
- animations

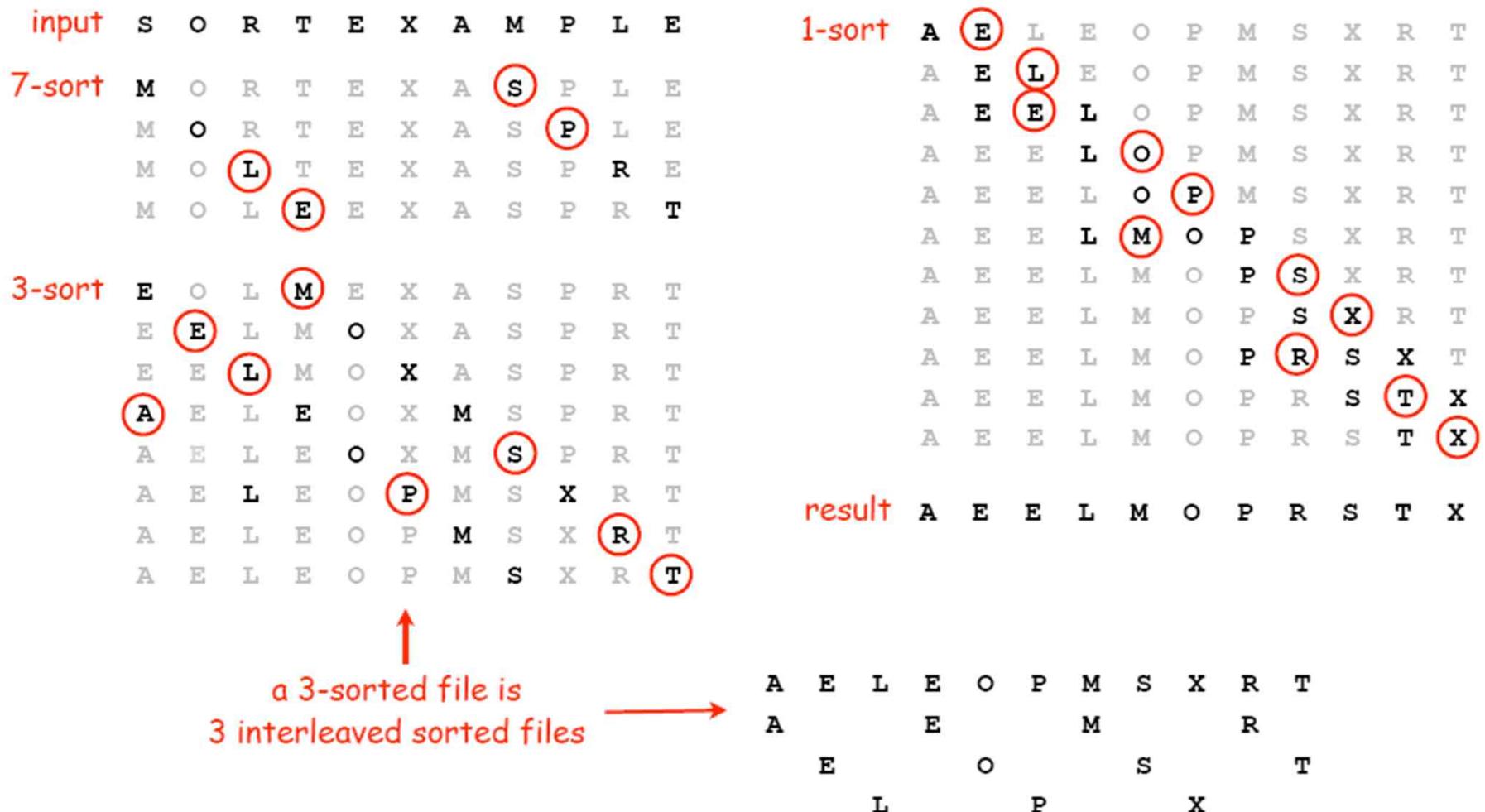
Visual representation of insertion sort



Reason it is slow: data movement

Shellsort (Donald Shell, 1959)

- Idea: move elements more than one position at a time
- by h-sorting the file for a decreasing sequence of values of h



Shellsort

- Idea: move elements more than one position at a time
- by h-sorting the file for a decreasing sequence of values of h
- Use insertion sort, modified to h-sort

big increments:
small subfiles

small increments:
subfiles nearly in order

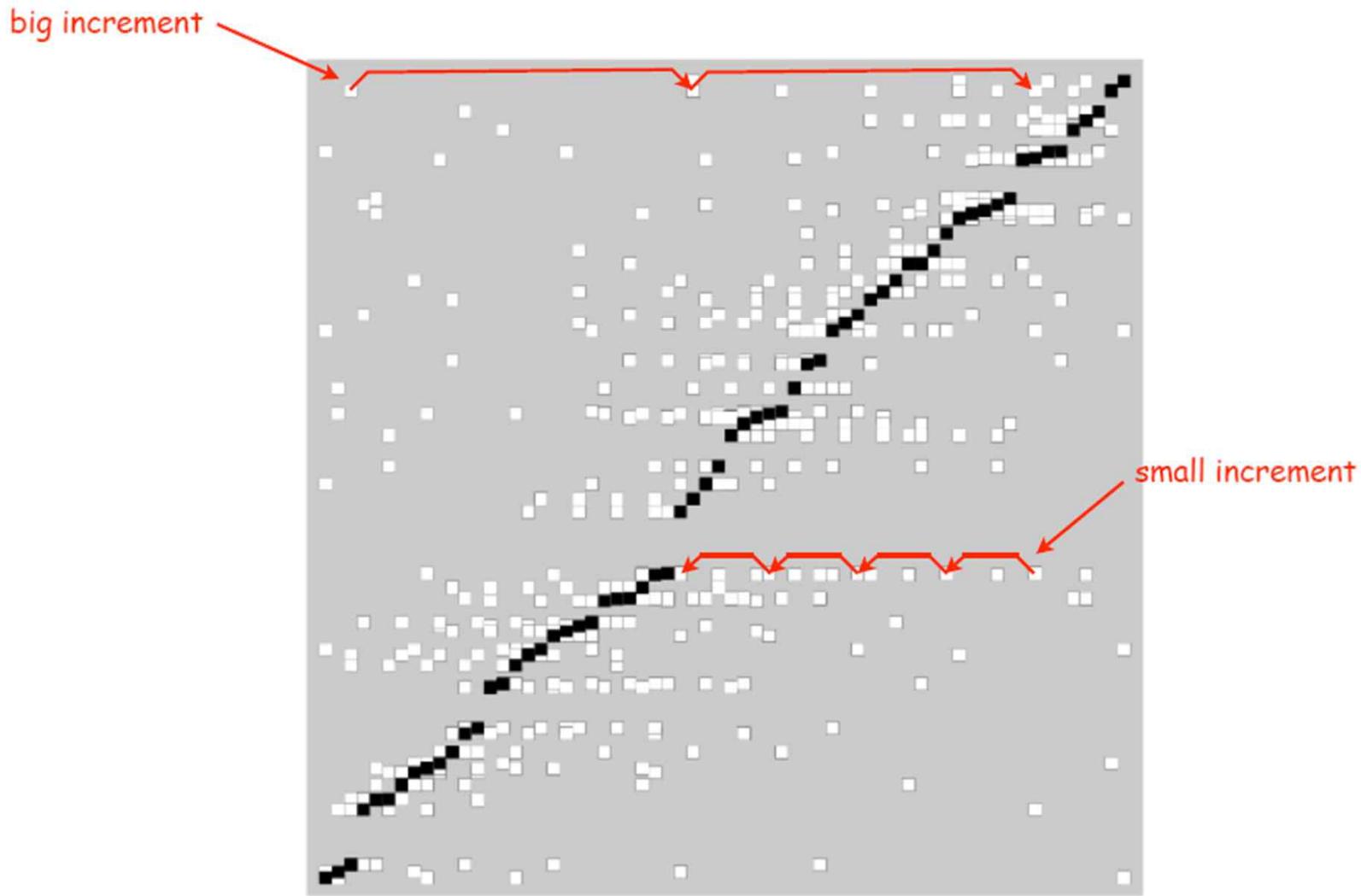
method of choice for both
small subfiles
subfiles nearly in order

insertion sort!

```
public static void sort(double[] a)
{
    int N = a.length;
    int[] incs = { 1391376, 463792, 198768, 86961,
                  33936, 13776, 4592, 1968, 861,
                  336, 112, 48, 21, 7, 3, 1 };
    for (int k = 0; k < incs.length; k++)
    {
        int h = incs[k];
        for (int i = h; i < N; i++)
            for (int j = i; j >= h; j-= h)
                if (less(a[j], a[j-h]))
                    exch(a, j, j-h);
                else break;
    }
}
```

magic increment
sequence

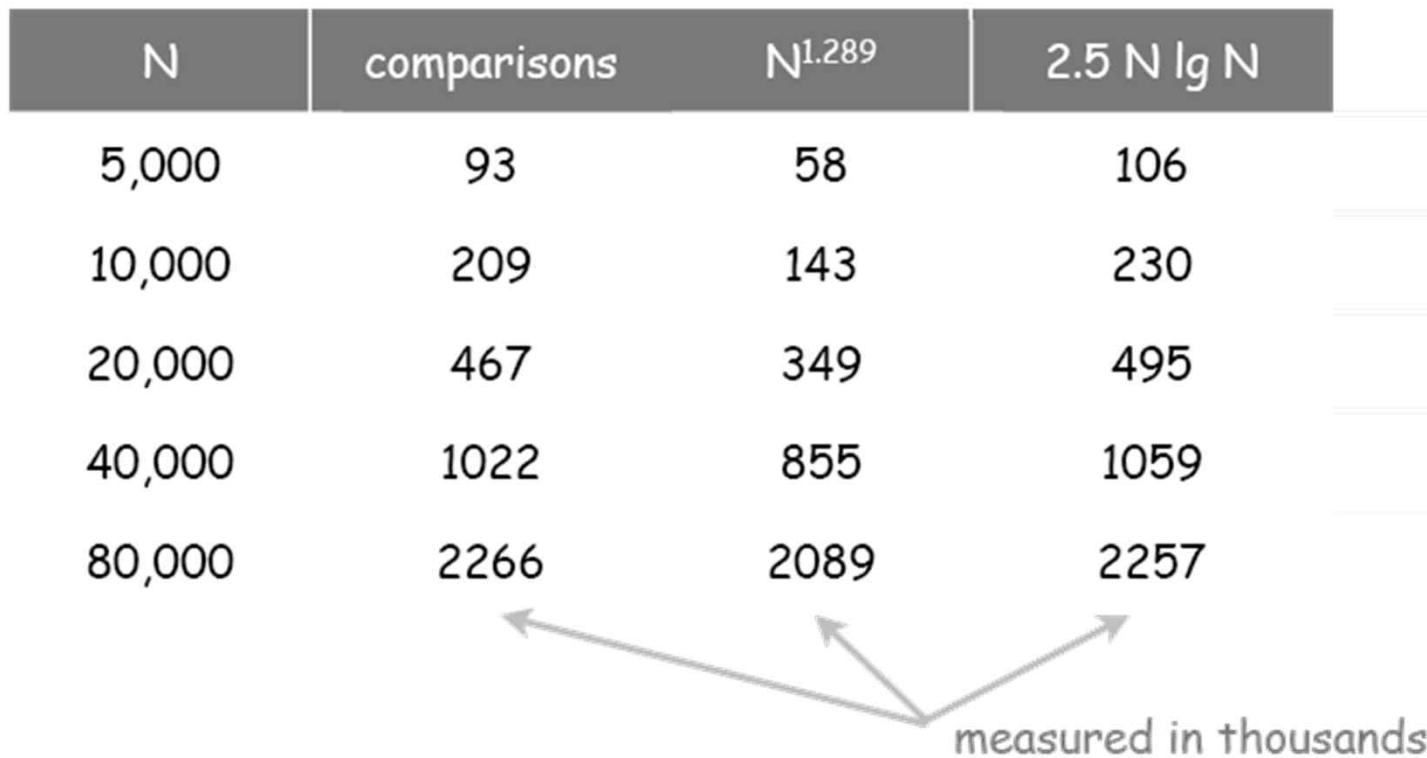
Visual representation of shellsort



- Bottom line: substantially faster!

Analysis of shellsort

- Model has not yet been discovered (!)



Why are we interested in shellsort?

- Example of simple idea leading to substantial performance gains
- Useful in practice
 - ◆ fast unless file size is huge
 - ◆ tiny, fixed footprint for code (used in embedded systems)
 - ◆ hardware sort prototype
- Simple algorithm, nontrivial performance, interesting questions
 - ◆ asymptotic growth rate?
 - ◆ best sequence of increments?
 - ◆ average case performance?
- Your first open problem in algorithmics (see Section 6.8):
Find a better increment sequence
- Lesson: some good algorithms are still waiting discovery

3. Sorting Algorithms

- rules of the game
- shellsort
- mergesort
- quicksort
- animations

Mergesort (von Neumann, 1945(!))

□ Basic plan:

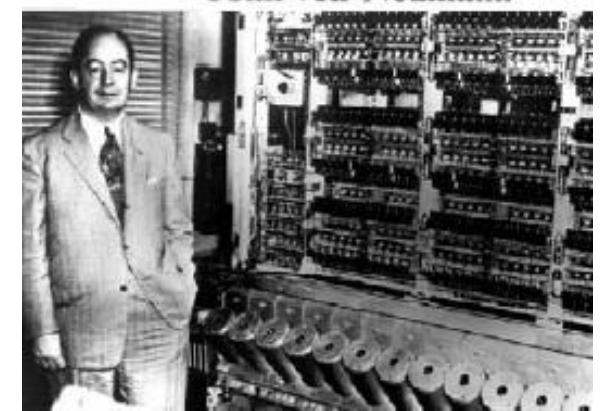
- ◆ Divide array into two halves.
- ◆ Recursively sort each half.
- ◆ Merge two halves.

		trace															a[i]														
		lo	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15												
				M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E												
0	1	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E														
2	3	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E														
0	3	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E														
4	5	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E														
6	7	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E														
4	7	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E														
0	7	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E														
8	9	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E														
10	11	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E														
8	11	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E														
12	13	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E														
14	15	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L														
12	15	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P														
8	15	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X														
0	15	A	E	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X													

plan

M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E	
E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E	
E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X	
A	E	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

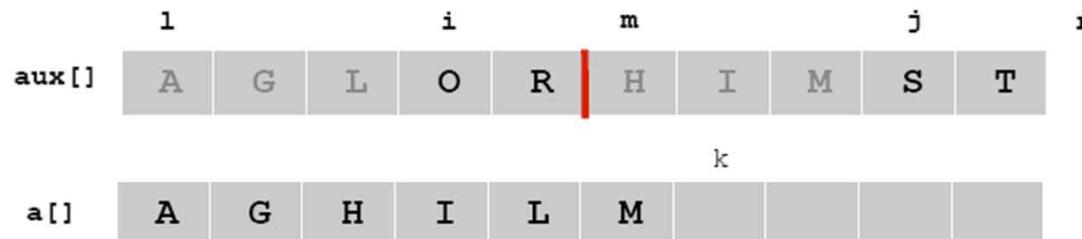
First Draft of a Report on the EDVAC



John von Neumann

Merging

- Merging. Combine two pre-sorted lists into a sorted whole.
- How to merge efficiently? Use an auxiliary array.



```
private static void merge(Comparable[] a,
                        Comparable[] aux, int l, int m, int r)
{
    copy → for (int k = l; k < r; k++) aux[k] = a[k];
    int i = l, j = m;
    for (int k = l; k < r; k++)
        if      (i >= m)                  a[k] = aux[j++];
        else if (j >= r)                  a[k] = aux[i++];
        else if (less(aux[j], aux[i]))    a[k] = aux[j++];
        else                                a[k] = aux[i++];

    merge →
}
```

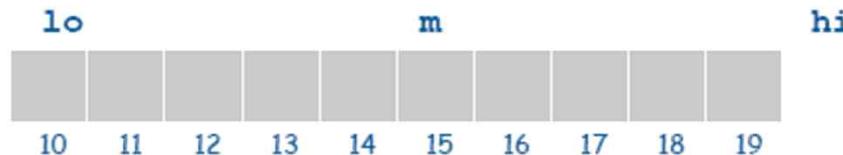
copy → for (int k = l; k < r; k++) aux[k] = a[k];
merge →

see book for a trick
to eliminate these

Mergesort: Java implementation of recursive sort

```
public class Merge
{
    private static void sort(Comparable[] a,
                           Comparable[] aux, int lo, int hi)
    {
        if (hi <= lo + 1) return;
        int m = lo + (hi - lo) / 2;
        sort(a, aux, lo, m);
        sort(a, aux, m, hi);
        merge(a, aux, lo, m, hi);
    }

    public static void sort(Comparable[] a)
    {
        Comparable[] aux = new Comparable[a.length];
        sort(a, aux, 0, a.length);
    }
}
```



Mergesort analysis: Memory

- Q. How much memory does mergesort require?
- A. Too much!
 - ◆ Original input array = N .
 - ◆ Auxiliary array for merging = N .
 - ◆ Local variables: constant.
 - ◆ Function call stack: $\log_2 N$ [stay tuned].
 - ◆ Total = $2N + O(\log N)$.

cannot “fill the memory and sort”
- Q. How much memory do other sorting algorithms require?
 - ◆ $N + O(1)$ for insertion sort and selection sort.
 - ◆ In-place = $N + O(\log N)$.
- Challenge for the bored. In-place merge. [Kronrud, 1969]

Mergesort analysis: Performance

□ Def. $T(N) \equiv$ number of array stores to mergesort an input of size N
 $= T(N/2) + T(N/2) + N$
 ↑ ↑ ↑
 left half right half merge

Mergesort recurrence

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

for $N > 1$, with $T(1) = 0$

- ◆ not quite right for odd N
- ◆ same recurrence holds for many algorithms
- ◆ same for any input of size N
- ◆ comparison count slightly smaller because of array ends

Solution of Mergesort recurrence

$$T(N) \sim N \lg N$$

$\lg N \equiv \log_2 N$

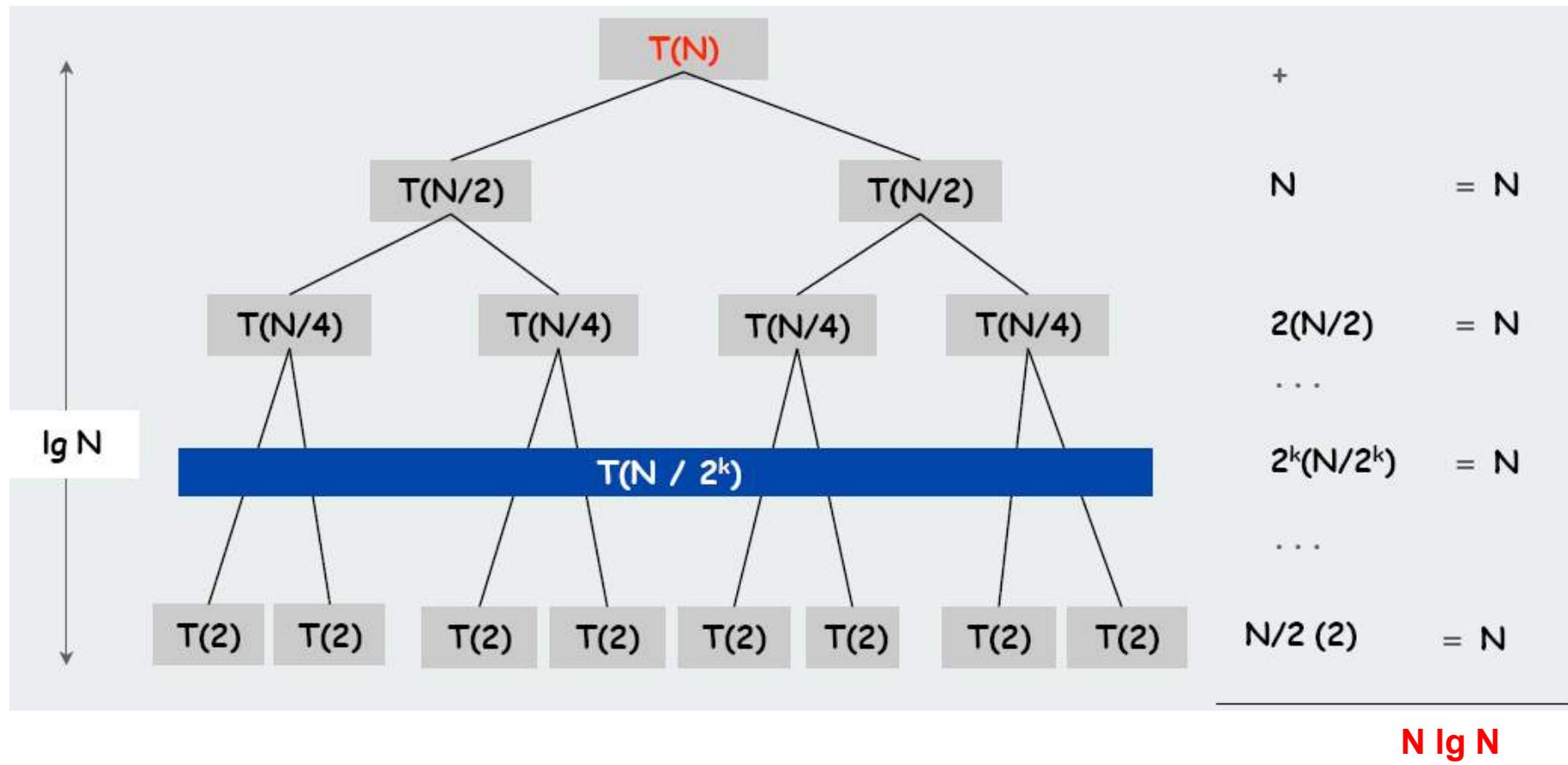
- ◆ true for all N
- ◆ easy to prove when N is a power of 2

Mergesort recurrence: Proof 1 (by recursion tree)

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

for $N > 1$, with $T(1) = 0$

(assume that N is a power of 2)



$$T(N) = N \lg N$$

Mergesort recurrence: Proof 2 (by telescoping)

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

for $N > 1$, with $T(1) = 0$

(assume that N is a power of 2)

Pf.

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

given

$$T(N)/N = 2 T(N/2)/N + 1$$

divide both sides by N

$$= T(N/2)/(N/2) + 1$$

algebra

$$= T(N/4)/(N/4) + 1 + 1$$

telescope (apply to first term)

$$= T(N/8)/(N/8) + 1 + 1 + 1$$

telescope again

...

$$= T(N/N)/(N/N) + 1 + 1 + \dots + 1$$

stop telescoping, $T(1) = 0$

$$= \lg N$$

$$T(N) = N \lg N$$



Mergesort recurrence: Proof 3 (by induction)

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

(assume that N is a power of 2)

for $N > 1$, with $T(1) = 0$

- **Claim.** If $T(N)$ satisfies this recurrence, then $T(N) = N \lg N$.
- **Pf.** [by induction on N]

- ◆ Base case: $N = 1$.
- ◆ Inductive hypothesis: $T(N) = N \lg N$
- ◆ Goal: show that $T(2N) = 2N \lg (2N)$.

$$\begin{aligned} T(2N) &= 2 T(N) + 2N && \text{give} \\ &= 2 N \lg N + 2 N && \text{inductive hypothesis} \\ &= 2 N (\lg (2N) - 1) + 2N && \text{algebra} \\ &= 2 N \lg (2N) && \text{QED} \end{aligned}$$

- Ex. Extend to show that $T(N) = N \lg N$ for general N



Bottom-up mergesort

□ Basic plan:

- ◆ Pass through file, merging to double size of sorted subarrays.
- ◆ Do so for subarray sizes 1, 2, 4, 8, ..., N/2, N.

		a[i]															
lo	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
0	1	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
2	3	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
4	5	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
6	7	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
8	9	E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
10	11	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
12	13	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
14	15	E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
		E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
0	3	E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
4	7	E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L
8	11	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
12	15	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
0	7	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
8	15	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
0	15	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

No recursion needed!

Algorithms

Bottom-up Mergesort: Java implementation

```
public class Merge
{
    private static void merge(Comparable[] a, Comparable[] aux,
                             int l, int m, int r)
    {
        for (int i = l; i < m; i++) aux[i] = a[i];
        for (int j = m; j < r; j++) aux[j] = a[m + r - j - 1];
        int i = l, j = r - 1;
        for (int k = l; k < r; k++)
            if (less(aux[j], aux[i])) a[k] = aux[j--];
            else                         a[k] = aux[i++];

    }

    public static void sort(Comparable[] a)
    {
        int N = a.length;
        Comparable[] aux = new Comparable[N];
        for (int m = 1; m < N; m = m+m)
            for (int i = 0; i < N-m; i += m+m)
                merge(a, aux, i, i+m, Math.min(i+m+m, N));
    }
}
```

tricky merge
that uses sentinel
(see Program 8.2)

Concise industrial-strength code if you have the space

Algorithms

Mergesort: Practical Improvements

- Use sentinel.
 - ◆ Two statements in inner loop are array-bounds checking.
 - ◆ Reverse one subarray so that **largest** element is sentinel (Program 8.2)
- Use insertion sort on small subarrays.
 - ◆ Mergesort has too much overhead for tiny subarrays.
 - ◆ Cutoff to insertion sort for ≈ 7 elements.
- Stop if already sorted.
 - ◆ Is biggest element in first half \leq smallest element in second half?
 - ◆ Helps for nearly ordered lists.
- Eliminate the copy to the auxiliary array. Save time (but not space) by switching the role of the input and auxiliary array in each recursive call.

See Program 8.4 (or Java system sort)



Sorting Analysis Summary

□ Running time estimates:

- ◆ Home pc executes 10^8 comparisons/second.
- ◆ Supercomputer executes 10^{12} comparisons/second.

Insertion Sort(N^2)				Mergesort($N \log N$)		
computer	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 sec	18 min
super	instant	1 second	1.6 weeks	instant	instant	instant

□ Lesson. Good algorithms are better than supercomputers.

Good enough?

18 minutes might be too long for some applications

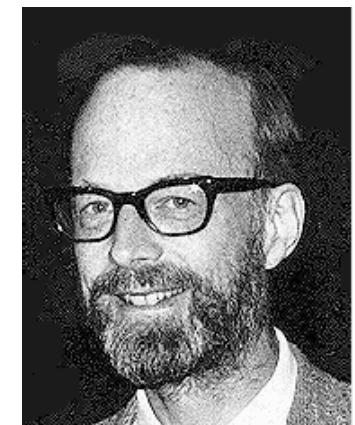
3. Sorting Algorithms

- rules of the game
- shellsort
- mergesort
- **quicksort**
- animations

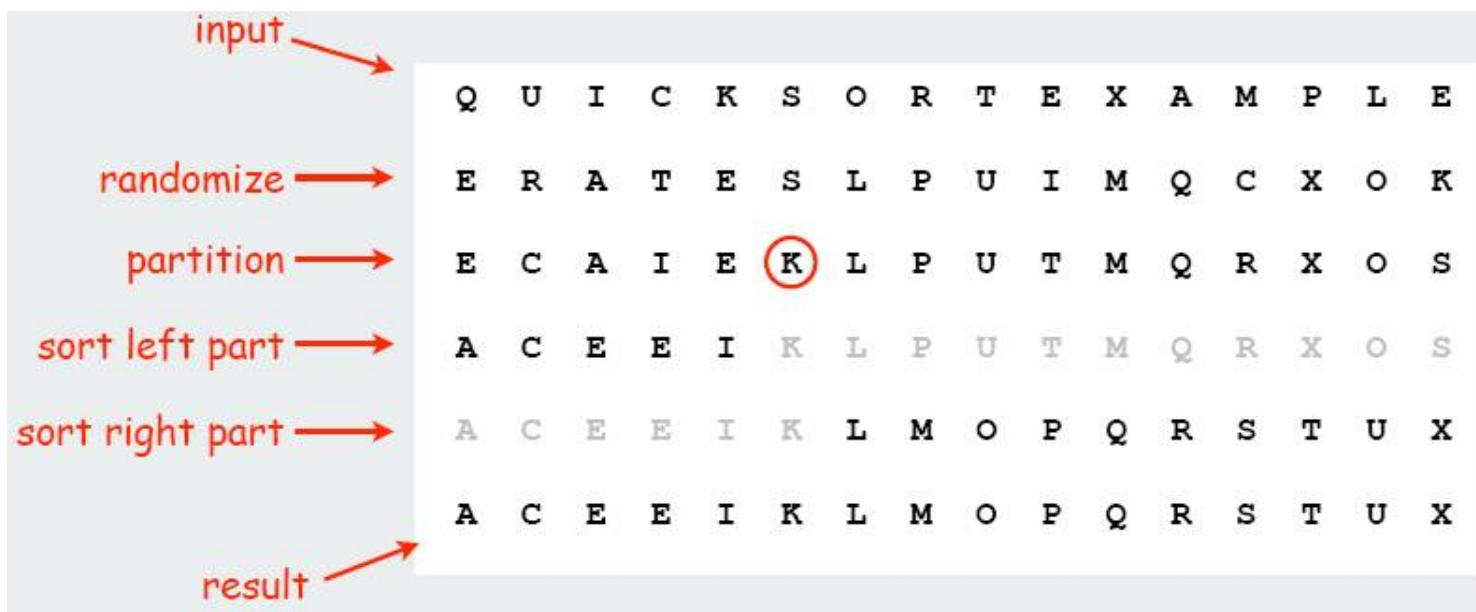
Quicksort (Hoare, 1959)

□ Basic plan.

- ◆ **Shuffle** the array.
- ◆ **Partition** so that, for some i
 - ✓ element $a[i]$ is in place
 - ✓ no larger element to the left of i
 - ✓ no smaller element to the right of i
- ◆ **Sort** each piece recursively.



Sir Charles Antony Richard Hoare
1980 Turing Award



Quicksort: Java Code for Recursive Sort

```
public class Quick
{
    public static void sort(Comparable[] a)
    {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1);
    }

    private static void sort(Comparable[] a, int l, int r)
    {
        if (r <= l) return;
        int m = partition(a, l, r);
        sort(a, l, m-1);
        sort(a, m+1, r);
    }
}
```

Quicksort Trace

a[i]																		
1	r	i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			Q	U	I	C	K	S	O	R	T	E	X	A	M	P	L	E
			E	R	A	T	E	S	L	P	U	I	M	Q	C	X	O	K
0	15	5	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S
0	4	2	A	C	E	I	E	K	L	P	U	T	M	Q	R	X	O	S
0	1	1	A	C	E	I	E	K	L	P	U	T	M	Q	R	X	O	S
0	0		A	C	E	I	E	K	L	P	U	T	M	Q	R	X	O	S
3	4	3	A	C	E	I	E	K	L	P	U	T	M	Q	R	X	O	S
4	4		A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
6	15	12	A	C	E	E	I	K	L	P	O	R	M	Q	S	X	U	T
6	11	10	A	C	E	E	I	K	L	P	O	M	Q	R	S	X	U	T
6	9	7	A	C	E	E	I	K	L	P	O	P	Q	R	S	X	U	T
6	6		A	C	E	E	I	K	L	M	O	P	Q	R	S	X	U	T
8	9	9	A	C	E	E	I	K	L	M	O	P	Q	R	S	X	U	T
8	8		A	C	E	E	I	K	L	M	O	P	Q	R	S	X	U	T
11	11		A	C	E	E	I	K	L	M	O	P	Q	R	S	X	U	T
13	15	13	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
14	15	15	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
14	14		A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X

array contents after each recursive sort

no partition for
subfiles of size 1

Quicksort Partitioning

□ Basic plan:

- ◆ scan from left for an item that belongs on the right
- ◆ scan from right for an item that belongs on the left
- ◆ Exchange
- ◆ continue until pointers cross

a[i]																					
i	j	r	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
-1	15	15	E	R	A	T	E	S	L	P	U	I	M	Q	C	X	O	K			
scans →			1	12	15	E	R	A	T	E	S	L	P	U	I	M	Q	C	X	O	K
exchange →			1	12	15	E	C	A	T	E	S	L	P	U	I	M	Q	R	X	O	K
3	9	15	E	C	A	T	E	S	L	P	U	I	M	Q	R	X	O	K			
3	9	15	E	C	A	I	E	S	L	P	U	T	M	Q	R	X	O	K			
5	5	15	E	C	A	I	E	S	L	P	U	T	M	Q	R	X	O	K			
5	5	15	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S			
array contents before and after each exchange																					

array contents before and after each exchange

Algorithms

Quicksort: Java Code for Partitioning

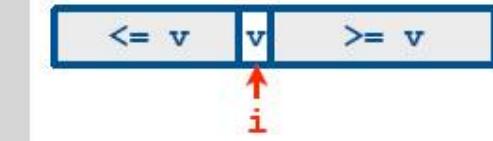
```
private static int partition(Comparable[] a, int l, int r)
{
    int i = l - 1;
    int j = r;
    while(true)
    {
        while (less(a[++i], a[r]))           find item on left to swap
            if (i == r) break;

        while (less(a[r], a[--j]))           find item on right to swap
            if (j == l) break;

        if (i >= j) break;                  check if pointers cross

        exch(a, i, j);                   swap
    }

    exch(a, i, r);                   swap with partitioning item
    return i;                        return index of item now known to be in place
}
```



Quicksort Implementation Details

- **Partitioning in-place.** Using a spare array makes partitioning easier, but is not worth the cost.
- **Terminating the loop.** Testing whether the pointers cross is a bit trickier than it might seem.
- **Staying in bounds.** The $(i == r)$ test is redundant, but the $(j == l)$ test is not.
- **Preserving randomness.** Shuffling is key for performance guarantee.
- **Equal keys.** When duplicates are present, it is (counter-intuitively) best to stop on elements equal to partitioning element.



Quicksort: Average Case Analysis

Theorem. The average number of comparisons C_N to quicksort a random file of N elements is about $2N \ln N$.

- ◆ The precise recurrence satisfies $C_0 = C_1 = 0$ and for $N \geq 2$:

$$\begin{aligned} C_N &= N + 1 + ((C_0 + C_{N-1}) + \dots + (C_{k-1} + C_{N-k}) + \dots + (C_{N-1} + C_1)) / N \\ &\quad \uparrow \qquad \qquad \qquad \uparrow \qquad \qquad \uparrow \\ &\quad \text{partition} \qquad \qquad \text{left} \qquad \qquad \text{right} \\ &= N + 1 + 2(C_0 + \dots + C_{k-1} + \dots + C_{N-1}) / N \end{aligned}$$

↑
partitioning probability

- ◆ Multiply both sides by N

$$NC_N = N(N + 1) + 2(C_0 + \dots + C_{k-1} + \dots + C_{N-1})$$

- ◆ Subtract the same formula for $N-1$:

$$NC_N - (N - 1)C_{N-1} = N(N + 1) - (N - 1)N + 2C_{N-1}$$

- ◆ Simplify:

$$NC_N = (N + 1)C_{N-1} + 2N$$

Quicksort: Average Case

$$NC_N = (N + 1)C_{N-1} + 2N$$

- ◆ Divide both sides by $N(N+1)$ to get a telescoping sum:

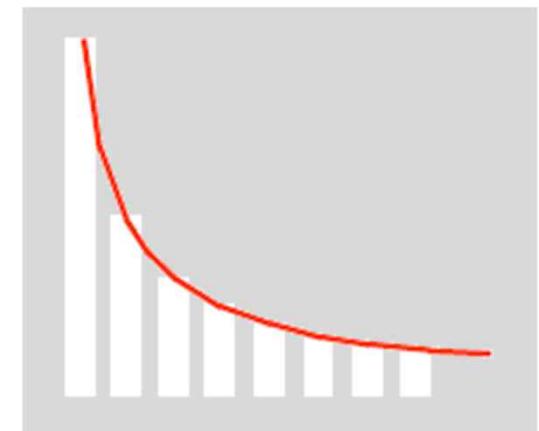
$$\begin{aligned} C_N / (N + 1) &= C_{N-1} / N + 2 / (N + 1) \\ &= C_{N-2} / (N - 1) + 2/N + 2/(N + 1) \\ &= C_{N-3} / (N - 2) + 2/(N - 1) + 2/N + 2/(N + 1) \\ &= 2 \left(1 + 1/2 + 1/3 + \dots + 1/N + 1/(N + 1) \right) \end{aligned}$$

- ◆ Approximate the exact answer by an integral:

$$\begin{aligned} C_N &\approx 2(N + 1)(1 + 1/2 + 1/3 + \dots + 1/N) \\ &= 2(N + 1) H_N \approx 2(N + 1) \int_1^n dx/x \end{aligned}$$

- ◆ Finally, the desired result:

$$C_N \approx 2(N + 1) \ln N \approx 1.39 N \lg N$$



Quicksort: Summary of Performance Characteristics

- Worst case. Number of comparisons is quadratic.
 - ◆ $N + (N-1) + (N-2) + \dots + 1 \approx N^2 / 2$.
 - ◆ More likely that your computer is struck by lightning.
- Average case. Number of comparisons is $\sim 1.39 N \lg N$.
 - ◆ 39% more comparisons than mergesort.
 - ◆ but faster than mergesort in practice because of lower cost of other high-frequency operations.
- Random shuffle
 - ◆ probabilistic guarantee against worst case
 - ◆ basis for math model that can be validated with experiments
- Caveat emptor. Many textbook implementations go quadratic if input:
 - ◆ Is sorted.
 - ◆ Is reverse sorted.
 - ◆ Has many duplicates (even if randomized)! [stay tuned]

Sorting Analysis Summary

□ Running time estimates:

- ◆ Home pc executes 10^8 comparisons/second.
- ◆ Supercomputer executes 10^{12} comparisons/second.

Insertion Sort (N^2)

computer	thousand	million	billion
home	instant	2.8 hours	317 years
super	instant	1 second	1.6 weeks

Mergesort($N \log N$)

thousand	million	billion
instant	1 sec	18 min
instant	instant	instant

Quicksort ($N \log N$)

thousand	million	billion
instant	0.3 sec	6 min
instant	instant	instant

- Lesson 1. Good algorithms are better than supercomputers.
- Lesson 2. Great algorithms are better than good ones.



Quicksort: Practical Improvements

- Median of sample.
 - ◆ Best choice of pivot element = median.
 - ◆ But how to compute the median?
 - ◆ Estimate true median by taking median of sample.
- Insertion sort small files.
 - ◆ Even quicksort has too much overhead for tiny files.
 - ◆ Can delay insertion sort until end.
- Optimize parameters.
 - ◆ Median-of-3 random elements. $\approx 12/7 N \log N$ comparisons
 - ◆ Cutoff to insertion sort for ≈ 10 elements.
- Non-recursive version.
 - ◆ Use explicit stack. guarantees $O(\log N)$ stack size
 - ◆ Always sort smaller half first.
- All validated with refined math models and experiments

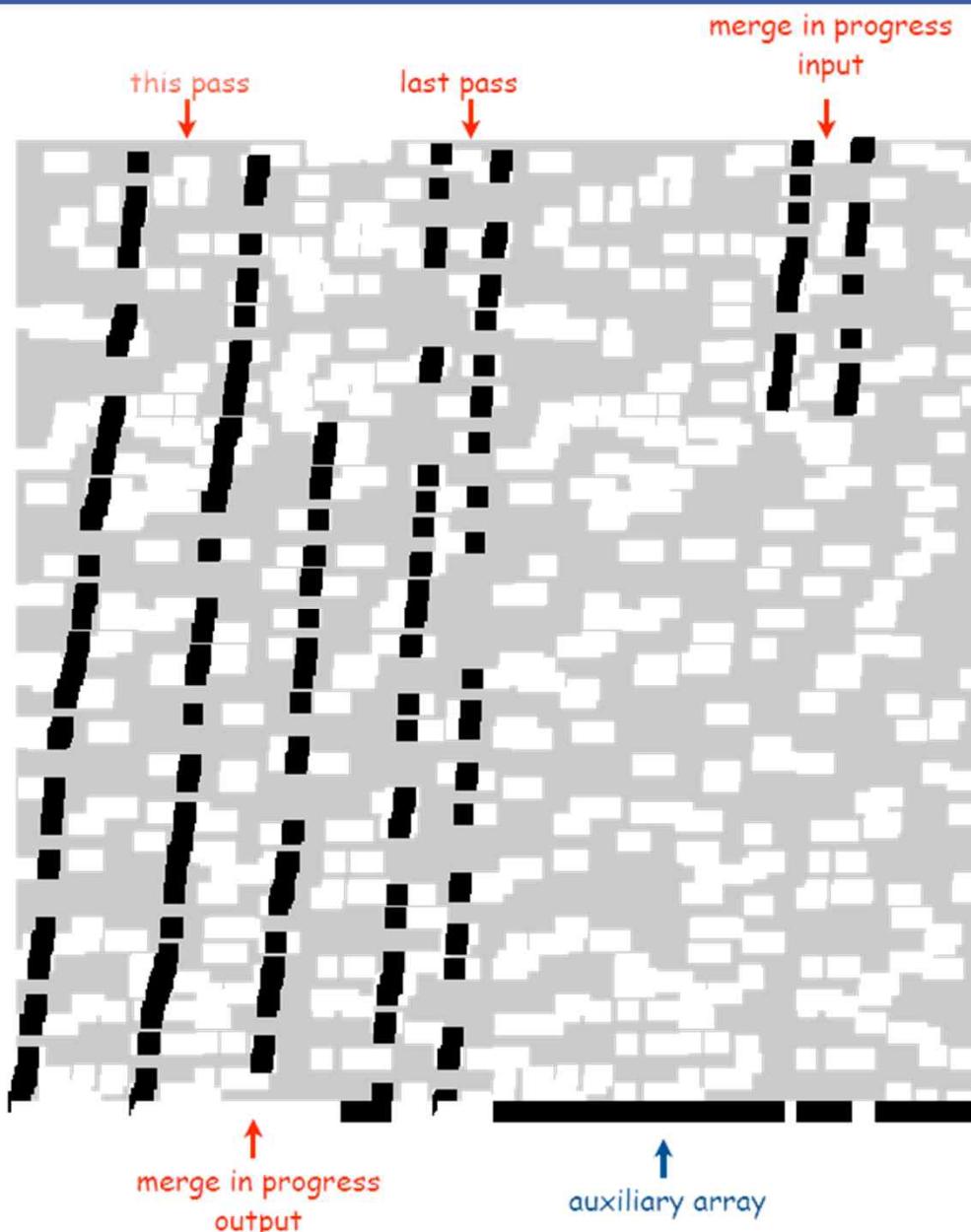
3. Sorting Algorithms

- rules of the game
- shellsort
- mergesort
- quicksort
- **animations**

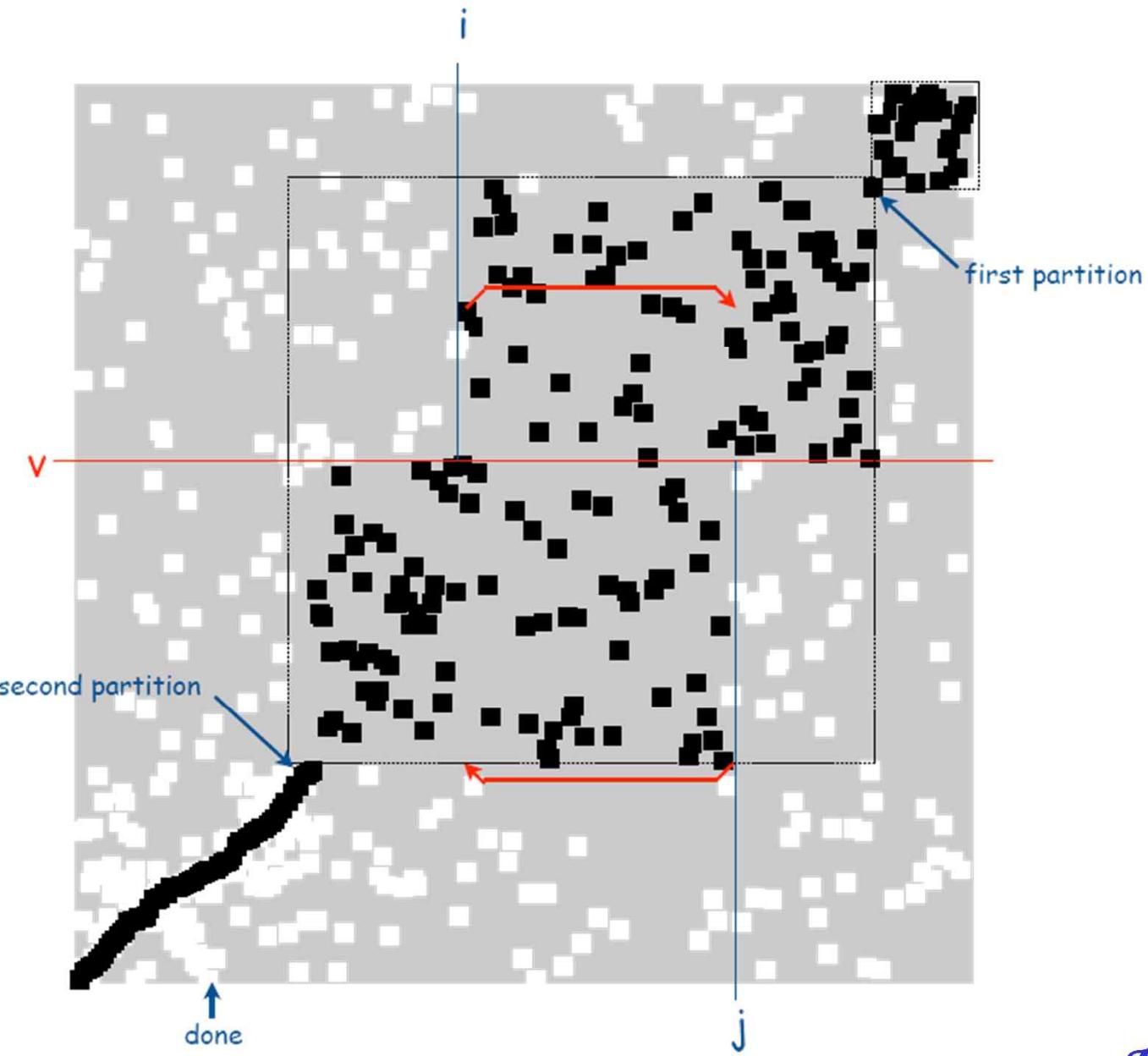
Mergesort Animation



Bottom-up mergesort Animation



Quicksort Animation



Data Structures & Algorithms

4. Advanced Topics in Sorting

- complexity
- system sorts
- duplicate keys
- comparators



Slides are Reformatted From Lecture Note of Algorithms Course,
by Robert Sedgewick, Princeton University, Fall, 2008.

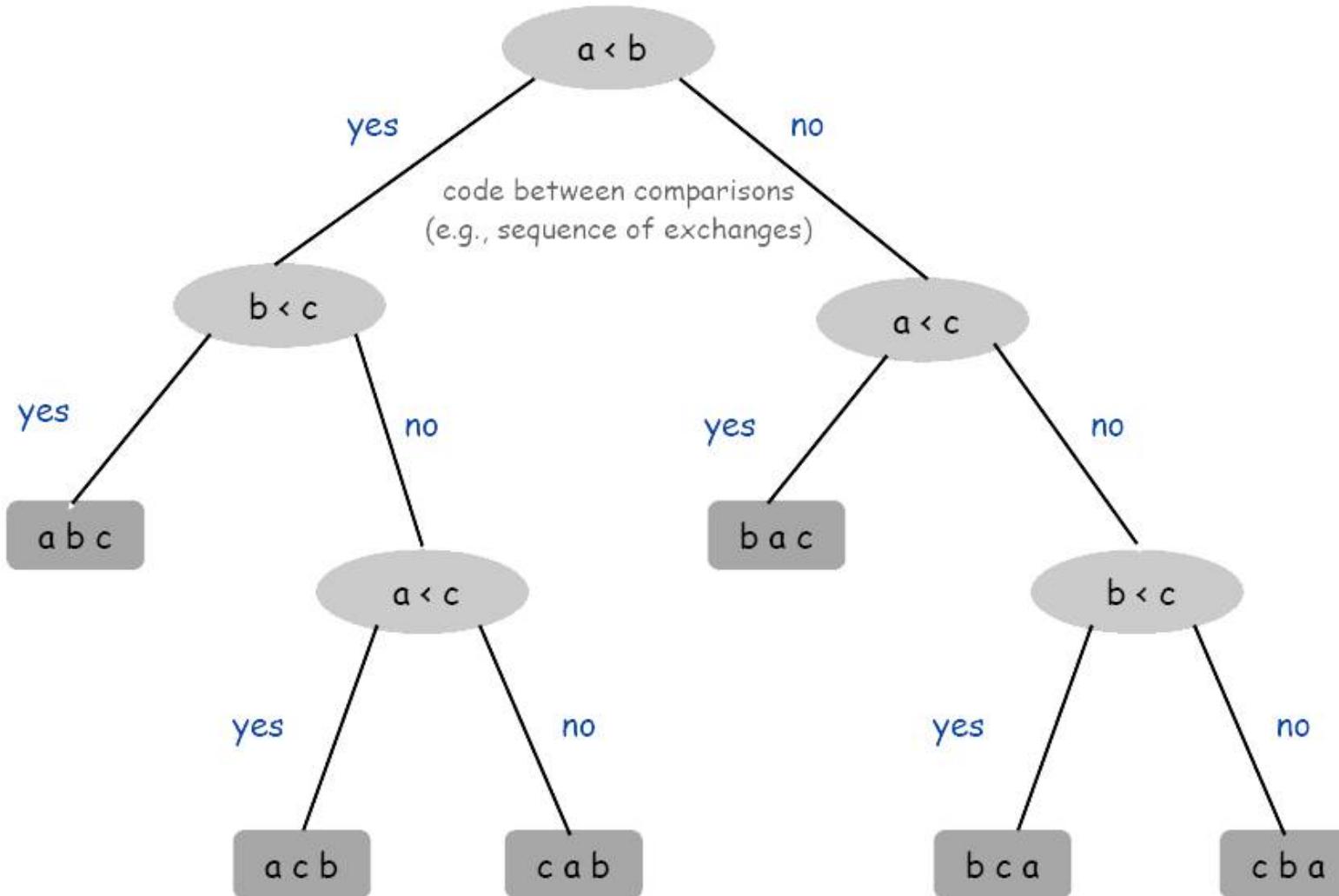
4. Advanced Topics in Sorting

- complexity
- system sorts
- duplicate keys
- comparators

Complexity of Sorting

- Computational complexity. Framework to study efficiency of algorithms for solving a particular problem X.
 - Machine model. Focus on fundamental operations.
 - Upper bound. Cost guarantee provided by some algorithm for X.
 - Lower bound. Proven limit on cost guarantee of any algorithm for X.
 - Optimal algorithm. Algorithm with best cost guarantee for X.
 - Example: sorting.
 - ◆ Machine model = # comparisons
 - ◆ Upper bound = $N \lg N$ from mergesort.
 - ◆ Lower bound ?
- lower bound ~ upper bound
- ↑
access information only
through compares

Decision Tree



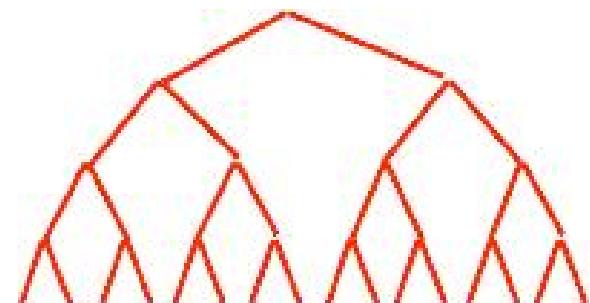
Comparison-Based Lower Bound for Sorting

Theorem. Any comparison based sorting algorithm must use more than $N \lg N - 1.44 N$ comparisons in the worst-case.

□ Pf.

- ◆ Assume input consists of N distinct values a_1 through a_N .
- ◆ Worst case dictated by tree height h .
- ◆ $N!$ different orderings.
- ◆ (At least) one leaf corresponds to each ordering.
- ◆ Binary tree with $N!$ leaves cannot have height less than $\lg(N!)$

$$\begin{aligned} & \checkmark h \geq \lg N! \\ & \checkmark \quad \geq \lg(N/e)^N \quad \leftarrow \text{Stirling's formula} \\ & \checkmark \quad = N \lg N - N \lg e \\ & \checkmark \quad \geq N \lg N - 1.44 N \end{aligned}$$



Complexity of Sorting

- **Upper bound.** Cost *guarantee* provided by some algorithm for X.
 - **Lower bound.** Proven limit on cost *guarantee* of any algorithm for X.
 - **Optimal algorithm.** Algorithm with best cost guarantee for X.
-
- Example: sorting.
 - ◆ Machine model = # comparisons
 - ◆ Upper bound = $N \lg N$ (mergesort)
 - ◆ Lower bound = $N \lg N - 1.44 N$

Mergesort is **optimal** (to within a small additive factor)
lower bound \approx upper bound

- First goal of algorithm design: optimal algorithms

Complexity Results in Context

□ Other operations?

- ◆ statement is only about number of compares
- ◆ quicksort is faster than mergesort (lower use of other operations)

□ Space?

- ◆ mergesort is not optimal with respect to space usage
- ◆ insertion sort, selection sort, shellsort, quicksort are space-optimal
- ◆ is there an algorithm that is both time- and space-optimal?

stay tuned for heapsort

□ Nonoptimal algorithms may be better in practice

- ◆ statement is only about guaranteed worst-case performance
- ◆ quicksort's probabilistic guarantee is just as good in practice

Lessons

- use theory as a guide
- know your algorithms

don't try to design an algorithm that uses half as many compares as mergesort

use quicksort when time and space are critical



Example: Selection

- Find the k th largest element.
 - ◆ Max: $k = 1$.
 - ◆ Min: $k = N$.
 - ◆ Median: $k = N/2$.
- Applications.
 - ◆ Order statistics.
 - ◆ Find the “top k ”
- Use theory as a guide
 - ◆ easy $O(N \log N)$ upper bound: sort, return $a[k]$
 - ◆ easy $O(N)$ upper bound for some k : min, max
 - ◆ easy $\Omega(N)$ lower bound: must examine every element
- Which is true?
 - ◆ $\Omega(N \log N)$ lower bound? [is selection as hard as sorting?]
 - ◆ $O(N)$ upper bound? [linear algorithm for all k]

Complexity Results in Context (Cont'd)

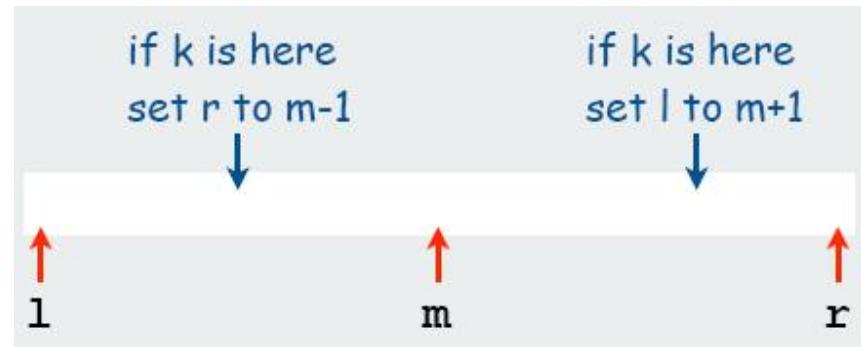
- Lower bound may not hold if the algorithm has information about
 - ◆ the key values
 - ◆ their initial arrangement
- Partially ordered arrays. Depending on the initial order of the input, we may not need $N \lg N$ compares.
 - insertion sort requires $O(N)$ compares on an already sorted array
- Duplicate keys. Depending on the input distribution of duplicates, we may not need $N \lg N$ compares.
 - stay tuned for 3-way quicksort
- Digital properties of keys. We can use digit/character comparisons instead of key comparisons for numbers and strings.
 - stay tuned for radix sorts

Selection: quick-select Algorithm

□ Partition array so that:

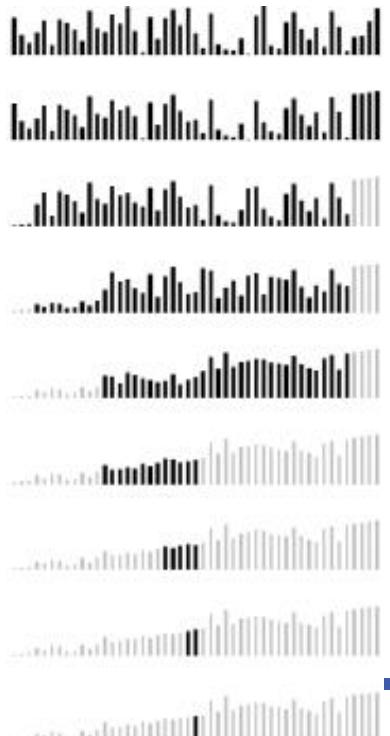
- ◆ element $a[m]$ is in place
- ◆ no larger element to the left of m
- ◆ no smaller element to the right of m

□ Repeat in one subarray, depending on m .



□ Finished when $m = k$ ← $a[k]$ is in place, no larger element to the left, no smaller element to the right

```
public static void select(Comparable[] a, int k)
{
    StdRandom.shuffle(a);
    int l = 0;
    int r = a.length - 1;
    while (r > l)
    {
        int m = partition(a, l, r);
        if (m > k) r = m - 1;
        else if (m < k) l = m + 1;
        else return;
    }
}
```



Quick-Select Analysis

Theorem. Quick-select takes linear time on average.

□ Pf.

- ◆ Intuitively, each partitioning step roughly splits array in half.
- ◆ $N + N/2 + N/4 + \dots + 1 \approx 2N$ comparisons.
- ◆ Formal analysis similar to quicksort analysis:

$$C_N = 2N + k \ln(N/k) + (N-k) \ln(N/(N-k))$$

↗
Ex: $(2 + 2 \ln 2)N$ comparisons to find the median

Note. Might use $\sim N^2/2$ comparisons, but as with quicksort, the random shuffle provides a probabilistic guarantee.

□ Theorem. [Blum, Floyd, Pratt, Rivest, Tarjan, 1973] There exists a selection algorithm that take linear time in the worst case.

Note. Algorithm is far too complicated to be useful in practice.

Use theory as a guide

- still worthwhile to seek practical linear-time (worst-case) algorithm
- until one is discovered, use quick-select if you don't need a full sort

4. Advanced Topics in Sorting

- complexity
- system sorts
- duplicate keys
- comparators

Sorting Challenge 1

- Problem: sort a file of huge records with tiny keys.
- Ex: reorganizing your MP3 files.

- Which sorting method to use?
 1. merge sort
 2. insertion sort
 3. selection sort

file →

record →

key →

FOX	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Puria	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazai	4	B	665-303-0266	113 Walker

Sorting Challenge 1

- Problem: sort a file of huge records with tiny keys.
- Ex: reorganizing your MP3 files.

- Which sorting method to use?
 1. merge sort probably no, selection sort simpler and faster
 2. insertion sort no, too many exchanges
 3. selection sort YES, linear time under reasonable assumptions

- Ex: 5,000 records, each 2 million bytes with 100-byte keys.
 - ◆ Cost of comparisons: $100 \times 5000^2 / 2 = 1.25$ billion
 - ◆ Cost of exchanges: $2,000,000 \times 5,000 = 10$ trillion
 - ◆ Merge sort might be a factor of $\log(5000)$ slower.

Sorting Challenge 2

- Problem: sort a huge randomly-ordered file of small records.
- Ex: process transaction records for a phone company.

- Which sorting method to use?
 1. quick sort
 2. insertion sort
 3. selection sort

file →

record →

key →

FOX	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazai	4	B	665-303-0266	113 Walker

Sorting Challenge 2

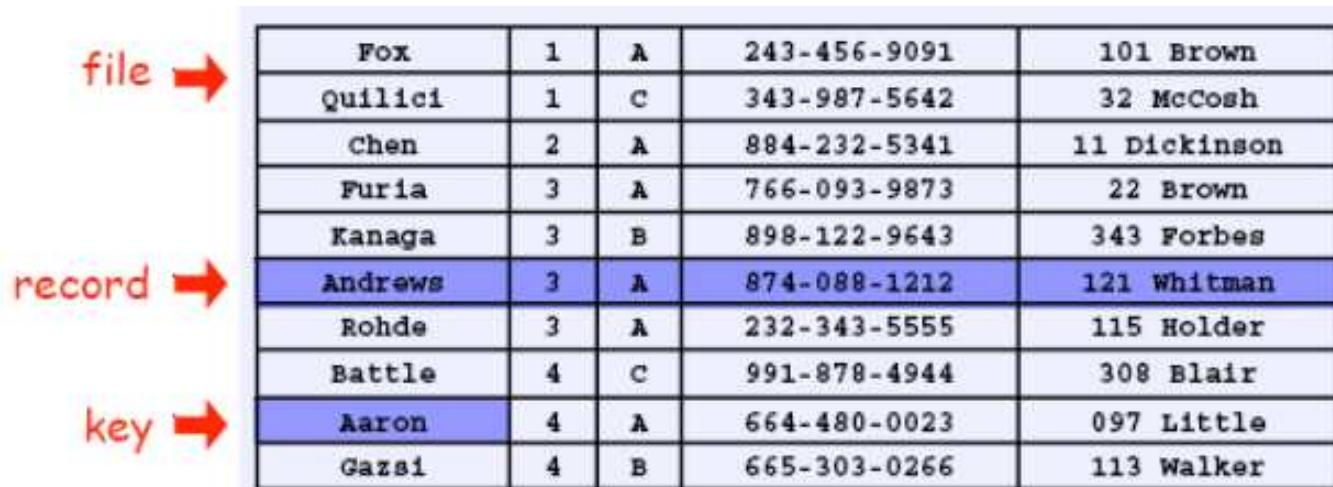
- Problem: sort a huge randomly-ordered file of small records.
- Ex: process transaction records for a phone company.

- Which sorting method to use?
 1. quick sort YES, it's designed for this problem
 2. insertion sort no, quadratic time for randomly-ordered files
 3. selection sort no, always takes quadratic time

Sorting Challenge 3

- Problem: sort a huge number of tiny files (each file is independent)
- Ex: daily customer transaction records.

- Which sorting method to use?
 1. quick sort
 2. insertion sort
 3. selection sort



The diagram illustrates the hierarchical structure of a file. On the left, three red arrows point from the words "file", "record", and "key" to specific columns in a table. The "file" arrow points to the first column, "record" to the second, and "key" to the third. The table itself consists of 10 rows of data, each representing a record. The columns are labeled: Name, ID, Category, Phone Number, and Address. The last two rows of the table are highlighted with a blue background.

FOX	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Puria	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Sorting Challenge 3

- Problem: sort a huge number of tiny files (each file is independent)
- Ex: daily customer transaction records.

- Which sorting method to use?
 1. quick sort no, too much overhead
 2. insertion sort YES, much less overhead than system sort
 3. selection sort YES, much less overhead than system sort

- Ex: 4 record file.
 - ◆ $4 N \log N + 35 = 70$
 - ◆ $2N^2 = 32$

Sorting Challenge 4

- Problem: sort a huge file that is already almost in order.
- Ex: re-sort a huge database after a few changes.

- Which sorting method to use?
 1. quick sort
 2. insertion sort
 3. selection sort

file →

record →

key →

FOX	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Puria	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Sorting Challenge 4

- Problem: sort a huge file that is already almost in order.
- Ex: re-sort a huge database after a few changes.

- Which sorting method to use?
 1. quick sort probably no, insertion simpler and faster
 2. insertion sort YES, linear time for most definitions of "in order"
 3. selection sort no, always takes quadratic time

- Ex:

A B C D E F H I J G P K L M N O Q R S T U V W X Y Z

Z A B C D E F G H I J K L M N O P Q R S T U V W X Y

Sorting Applications

- Sorting algorithms are essential in a broad variety of applications
 - ◆ Sort a list of names.
 - ◆ Organize an MP3 library.
 - ◆ Display Google PageRank results.
 - ◆ List RSS (Really Simple Syndication) news items in reverse chronological order.

 - ◆ Find the median.
 - ◆ Find the closest pair.
 - ◆ Binary search in a database.
 - ◆ Identify statistical outliers.
 - ◆ Find duplicates in a mailing list.
- obvious applications
-
- ◆ Data compression.
 - ◆ Computer graphics.
 - ◆ Computational biology.
 - ◆ Supply chain management.
 - ◆ Load balancing on a parallel computer.
- problems become easy once
items are in sorted order
-
- ◆ Every system needs (and has) a system sort!
- non-obvious applications



System Sort: Which Algorithm to Use?

- Many sorting algorithms to choose from
- internal sorts.
 - ◆ insertion sort, selection sort, bubblesort, shaker sort.
 - ◆ quicksort, mergesort, heapsort, samplesort, shellsort.
 - ◆ solitaire sort, red-black sort, splaysort, Dobosiewicz sort, psort, ...
- external sorts. Poly-phase mergesort, cascade-merge, oscillating sort.
- radix sorts.
 - ◆ Distribution, MSD, LSD.
 - ◆ 3-way radix quicksort.
- parallel sorts.
 - ◆ bitonic sort, Batcher even-odd sort.
 - ◆ smooth sort, cube sort, column sort.
 - ◆ GPUsort.



System Sort: Which Algorithm to Use?

- Applications have diverse attributes

- ◆ Stable?
- ◆ Multiple keys?
- ◆ Deterministic?
- ◆ Keys all distinct?
- ◆ Multiple key types?
- ◆ Linked list or arrays?
- ◆ Large or small records?
- ◆ Is your file randomly ordered?
- ◆ Need guaranteed performance?

	attributes									
algorithm	1	2	3	4	.	.	.	M		
A	•			•						
B			•	•						•
C	•		•							
D							•			
E				•						
F	•			•			•			
G	•									•
·		·	·		·	·	·			
·		·	·		·	·	·			
K	•			•						

many more combinations of attributes than algorithms

- Elementary sort may be method of choice for some combination.
- Cannot cover **all** combinations of attributes.

- Q. Is the system sort good enough?
- A. Maybe (no matter which algorithm it uses).

4. Advanced Topics in Sorting

- complexity
- system sorts
- duplicate keys
- comparators

Duplicate Keys

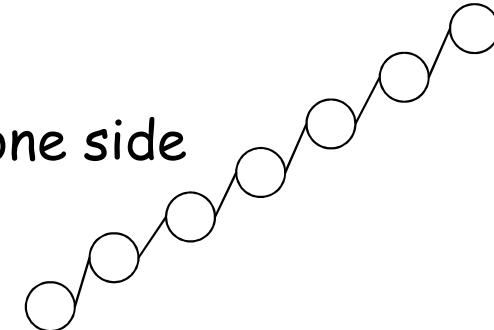
- Often, purpose of sort is to bring records with duplicate keys together.
 - ◆ Sort population by age.
 - ◆ Finding collinear points.
 - ◆ Remove duplicates from mailing list.
 - ◆ Sort job applicants by college attended.
- Typical characteristics of such applications.
 - Huge file.
 - Small number of key values.
- Mergesort with duplicate keys: always $\sim N \lg N$ compares
- Quicksort with duplicate keys
 - ◆ algorithm goes **quadratic** unless partitioning stops on equal keys!
 - ◆ [many textbook and system implementations have this problem]
 - ◆ 1990s Unix user found this problem in `qsort()`

Duplicate Keys: the Problem

- Assume all keys are **equal**.
- Recursive code guarantees that case will predominate!
- **Mistake:** Put all keys equal to the partitioning element on one side
 - ◆ easy to code
 - ◆ guarantees N^2 running time when all keys equal

B A A B A B B B C C C

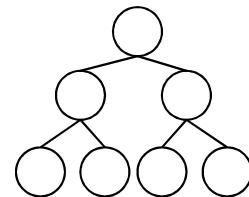
A A A A A A A A A A A A A A A



- **Recommended:** Stop scans on keys equal to the partitioning element
 - ◆ easy to code
 - ◆ guarantees $N \lg N$ compares when all keys equal

B A A B A B C C B C B

A A A A A A A A A A A A A A A



- **Desirable:** Put all keys equal to the partitioning element in place

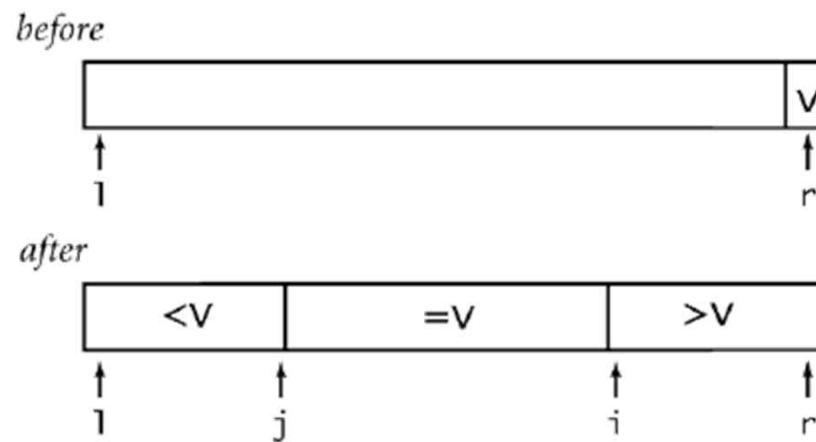
A A A B B B B B C C C

A A A A A A A A A A A A A A A

- Common wisdom to 1990s: not worth adding code to inner loop

3-Way Partitioning

- 3-Way Partitioning. Partition elements into 3 parts:
 - ◆ Elements **between i and j** equal to partition element v.
 - ◆ No larger elements to **left of j**.
 - ◆ No smaller elements to **right of i**.



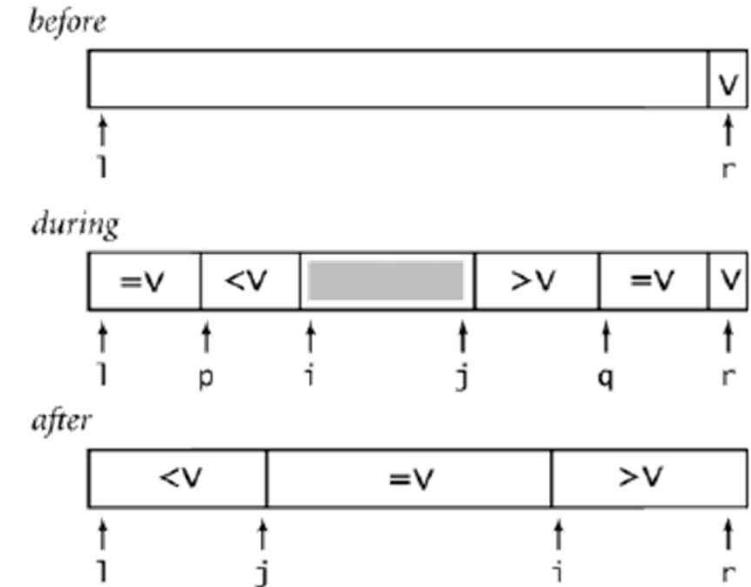
Dutch National Flag Problem.

- ◆ not done in practical sorts before mid-1990s.
- ◆ new approach discovered when fixing mistake in Unix qsort()
- ◆ now incorporated into Java system sort



Solution to Dutch National Flag Problem.

- 3-way partitioning (Bentley-McIlroy).
 - ◆ Partition elements into 4 parts:
 - ✓ no larger elements to left of i
 - ✓ no smaller elements to right of j
 - ✓ equal elements to left of p
 - ✓ equal elements to right of q
 - ◆ Afterwards, swap equal keys into center.



- All the right properties.
 - ◆ in-place.
 - ◆ not much code.
 - ◆ linear if keys are all equal.
 - ◆ small overhead if no equal keys.

3-Way Quicksort: Java Implementation

```
private static void sort(Comparable[] a, int l, int r)
{
    if (r <= l) return;
    int i = l-1, j = r;
    int p = l-1, q = r;

    while(true)                                4-way partitioning
    {
        while (less(a[++i], a[r])) ;
        while (less(a[r], a[--j])) if (j == l) break;
        if (i >= j) break;
        exch(a, i, j);
        if (eq(a[i], a[r])) exch(a, ++p, i);  swap equal keys to left or right
        if (eq(a[j], a[r])) exch(a, --q, j);
    }
    exch(a, i, r);

    j = i - 1;                                swap equal keys back to middle
    i = i + 1;
    for (int k = l ; k <= p; k++) exch(a, k, j--);
    for (int k = r-1; k >= q; k--) exch(a, k, i++);

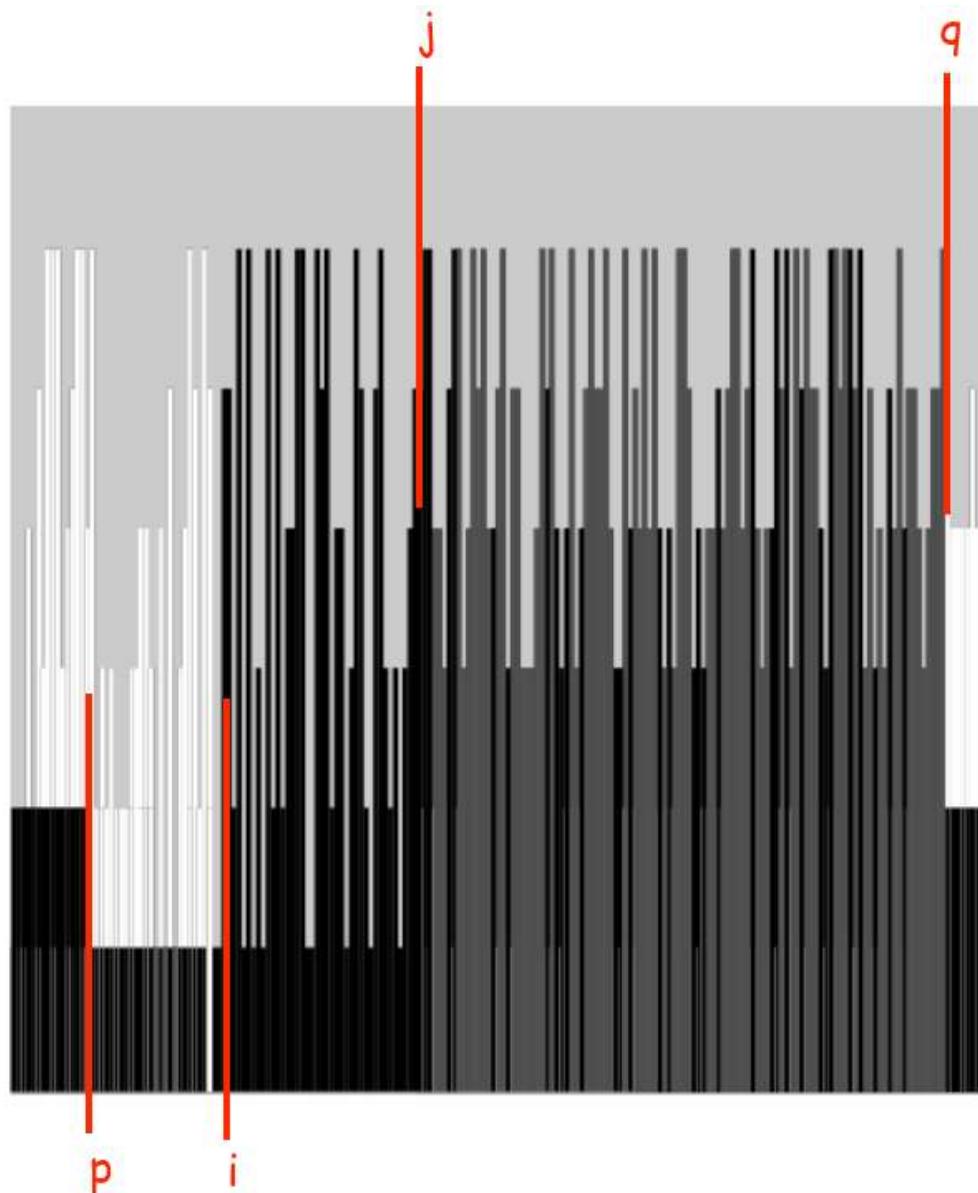
    sort(a, l, j);                            recursively sort left and right
    sort(a, i, r);
}
```

Duplicate Keys: Lower Bound

- Theorem. [Sedgewick-Bentley] Quicksort with 3-way partitioning is **optimal** for random keys with duplicates.
- Proof (beyond scope of the course).
 - ◆ generalize decision tree
 - ◆ tie cost to entropy
 - ◆ note: cost is **linear** when number of key values is $O(1)$

Bottom line: Randomized Quicksort with 3-way partitioning reduces cost from **linearithmic** to **linear** (!) in broad class of applications

3-Way Partitioning Animation



4. Advanced Topics in Sorting

- complexity
- system sorts
- duplicate keys
- comparators

Generalized Compare

- Comparable interface: sort uses type's compareTo() function:

```
public class Date implements Comparable<Date>
{
    private int month, day, year;

    public Date(int m, int d, int y)
    {
        month = m;
        day   = d;
        year  = y;
    }

    public int compareTo(Date b)
    {
        Date a = this;
        if (a.year < b.year) return -1;
        if (a.year > b.year) return +1;
        if (a.month < b.month) return -1;
        if (a.month > b.month) return +1;
        if (a.day   < b.day ) return -1;
        if (a.day   > b.day ) return +1;
        return 0;
    }
}
```



Generalized Compare

- Comparable interface: sort uses type's compareTo() function:

Problem 1: Not type-safe

Problem 2: May want to use a different order.

Problem 3: Some types may have no "natural" order.

- Ex. Sort strings by:

- ◆ Natural order.
- ◆ Case insensitive.
- ◆ French.
- ◆ Spanish.

Now is the time
is Now the time
real réal rico
café cuidado champiñón dulce

ch and rr are single letters

```
String[] a;  
...  
Arrays.sort(a);  
Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);  
Arrays.sort(a, Collator.getInstance(Locale.FRENCH));  
Arrays.sort(a, Collator.getInstance(Locale.SPANISH));
```

import java.text.Collator;



Generalized Compare

- Comparable interface: sort uses type's compareTo() function:

Problem 1: Not type-safe

Problem 2: May want to use a different order.

Problem 3: Some types may have no "natural" order.

- A bad client

```
public class BadClient
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        Comparable[] a = new Comparable[N];
        ...
        a[i] = 1;
        ...
        a[j] = 2.0;
        ...
        Insertion.sort(a);
    }
}
Exception ... java.lang.ClassCastException: java.lang.Double
at java.lang.Integer.compareTo(Integer.java:35)
```

autoboxed to Integer → a[i] = 1;

autoboxed to Double → a[j] = 2.0;

Generalized Compare

- Comparable interface: sort uses type's compareTo() function:

Problem 1: Not type-safe

Problem 2: May want to use a different order.

Problem 3: Some types may have no "natural" order.

- General key

```
public class Insertion
{
    public static <Key extends Comparable<Key>>
        void sort(Key[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
            for (int j = i; j > 0; j--)
                if (less(a[j], a[j-1])) exch(a, j, j-1);
                else break;
    }
}
```

Client can sort array of any Comparable type: Double[], File[], Date[], ...

Necessary in system library code; not in this course (for brevity)



Generalized Compare

- Comparable interface: sort uses type's compareTo() function:

Problem 1: Not type-safe

Problem 2: May want to use a different order.

Problem 3: Some types may have no "natural" order.

Solution: Use Comparator interface

- Comparator interface. Require a method compare() so that compare(v, w) is a total order that behaves like compareTo().
- Advantage. Separates the definition of the data type from definition of what it means to compare two objects of that type.
 - ◆ add any number of new orders to a data type.
 - ◆ add an order to a library data type with no natural order.



Generalized Compare

- Comparable interface: sort uses type's compareTo() function:

Problem 2: May want to use a different order.

Problem 3: Some types may have no "natural" order.

- Solution: Use Comparator interface

- Example:

```
public class ReverseOrder implements Comparator<String>
{
    public int compare(String a, String b)
    {
        return - a.compareTo(b);    }
}
```

reverse sense of comparison

```
...
Arrays.sort(a, new ReverseOrder());
...
```

Generalized Compare

- Easy modification to support comparators in our sort implementations
 - ◆ pass comparator to sort(), less()
 - ◆ use it in less()
- Example: (insertion sort)

```
public static void sort(Object[] a, Comparator comparator)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0; j--)
            if (less(comparator, a[j], a[j-1]))
                exch(a, j, j-1);
            else break;
}

private static boolean less(Comparator c, Object v, Object w)
{   return c.compare(v, w) < 0; }

private static void exch(Object[] a, int i, int j)
{   Object t = a[i]; a[i] = a[j]; a[j] = t; }
```

Generalized Compare

- Comparators enable multiple sorts of single file (different keys)
- Example. Enable sorting students by name or by section.

```
Arrays.sort(students, Student.BY_NAME);  
Arrays.sort(students, Student.BY_SECT);
```

sort by name



Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	2	A	991-878-4944	308 Blair
Fox	1	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	101 Brown
Gazsi	4	B	665-303-0266	22 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	3	A	232-343-5555	343 Forbes

then sort by section



Fox	1	A	884-232-5341	11 Dickinson
Chen	2	A	991-878-4944	308 Blair
Andrews	3	A	664-480-0023	097 Little
Furia	3	A	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	3	A	232-343-5555	343 Forbes
Battle	4	C	874-088-1212	121 Whitman
Gazsi	4	B	665-303-0266	22 Brown

Generalized Compare

- Comparators enable multiple sorts of single file (different keys)
- Example. Enable sorting students by name or by section.

```
public class Student
{
    public static final Comparator<Student> BY_NAME = new ByName();
    public static final Comparator<Student> BY_SECT = new BySect();

    private String name;
    private int section;
    ...

    private static class ByName implements Comparator<Student>
    {
        public int compare(Student a, Student b)
        { return a.name.compareTo(b.name); }
    }

    private static class BySect implements Comparator<Student>
    {
        public int compare(Student a, Student b)
        { return a.section - b.section; }
    }
}
```

only use this trick if no danger of overflow

Generalized Compare problem

- A typical application
 - ◆ first, sort by name
 - ◆ then, sort by section

```
Arrays.sort(students, Student.BY_NAME);
```



Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	2	A	991-878-4944	308 Blair
Fox	1	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	101 Brown
Gazsi	4	B	665-303-0266	22 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	3	A	232-343-5555	343 Forbes

```
Arrays.sort(students, Student.BY_SECT);
```



Fox	1	A	884-232-5341	11 Dickinson
Chen	2	A	991-878-4944	308 Blair
Kanaga	3	B	898-122-9643	22 Brown
Andrews	3	A	664-480-0023	097 Little
Furia	3	A	766-093-9873	101 Brown
Rohde	3	A	232-343-5555	343 Forbes
Battle	4	C	874-088-1212	121 Whitman
Gazsi	4	B	665-303-0266	22 Brown

- @#%&@!! Students in section 3 no longer in order by name.
- A **stable** sort preserves the relative order of records with equal keys.
Is the system sort stable?

Stability

- Q. Which sorts are stable?
 - ◆ Selection sort?
 - ◆ Insertion sort?
 - ◆ Shellsort?
 - ◆ Quicksort?
 - ◆ Mergesort?

- A. Careful look at code required.

- Annoying fact. Many useful sorting algorithms are unstable.

- Easy solutions.
 - ◆ add an integer rank to the key
 - ◆ careful implementation of mergesort

Open: Stable, inplace, optimal, practical sort??



Java System Sorts

- Use theory as a guide: Java uses both mergesort and quicksort.
 - ◆ Can sort array of type Comparable or any primitive type.
 - ◆ Uses quicksort for primitive types.
 - ◆ Uses mergesort for objects.

```
import java.util.Arrays;
public class IntegerSort
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        int[] a = new int[N];
        for (int i = 0; i < N; i++)
            a[i] = StdIn.readInt();
        Arrays.sort(a);
        for (int i = 0; i < N; i++)
            System.out.println(a[i]);
    }
}
```

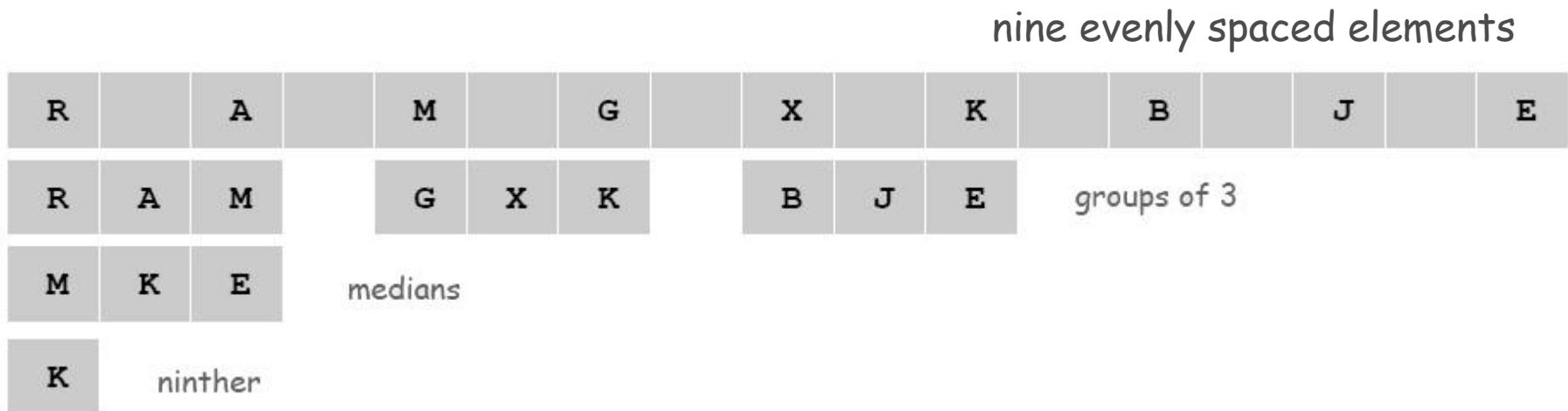
Q. Why use two different sorts?

- A. Use of primitive types indicates time and space are critical
- A. Use of objects indicates time and space not so critical



Arrays.sort() for Primitive Types

- Bentley-McIlroy. [Engineering a Sort Function]
 - ◆ Original motivation: improve qsort() function in C.
 - ◆ Basic algorithm = 3-way quicksort with cutoff to insertion sort.
 - ◆ Partition on Tukey's ninther: median-of-3 elements, each of which is a median-of-3 elements.
 - ↑ approximate median-of-9



□ Why use ninther?

- ◆ better partitioning than sampling
- ◆ quick and easy to implement with macros
- ◆ less costly than random

← Good idea? Stay tuned.

Achilles Heel in Bentley-McIlroy Implementation (Java System Sort)

- Based on all this research, Java's system sort is solid, **right?**
- McIlroy's devious idea. [A Killer Adversary for Quicksort]
 - ◆ Construct malicious input **while** running system quicksort, in response to elements compared.
 - ◆ If p is pivot, commit to $(x < p)$ and $(y < p)$, but don't commit to $(x < y)$ or $(x > y)$ until x and y are compared.
- Consequences.
 - ◆ Confirms theoretical possibility.
 - ◆ Algorithmic complexity attack: you enter linear amount of data; server performs quadratic amount of work.

Achilles Heel in Bentley-McIlroy Implementation (Java System Sort)

□ A killer input:

- ◆ blows function call stack in Java and crashes program
- ◆ would take quadratic time if it didn't crash first

more disastrous possibilities in C

```
% more 250000.txt
0
218750
222662
11
166672
247070
83339
156253
...
```

```
% java IntegerSort < 250000.txt
Exception in thread "main" java.lang.StackOverflowError
    at java.util.Arrays.sort1(Objects.java:562)
    at java.util.Arrays.sort1(Objects.java:606)
    at java.util.Arrays.sort1(Objects.java:608)
    at java.util.Arrays.sort1(Objects.java:608)
    at java.util.Arrays.sort1(Objects.java:608)
    ...
    . . .
```

250,000 integers between
0 and 250,000

Java's sorting library crashes, even if
you give it as much stack space as Windows allows.

Attack is **not** effective if file is randomly ordered before sort



System Sort: Which Algorithm to Use?

❑ Applications have diverse attributes

- ◆ Stable?
- ◆ Multiple keys?
- ◆ Deterministic?
- ◆ Keys all distinct?
- ◆ Multiple key types?
- ◆ Linked list or arrays?
- ◆ Large or small records?
- ◆ Is your file randomly ordered?
- ◆ Need guaranteed performance?

algorithm	attributes							
	1	2	3	4	.	.	.	M
A	•			•				
B		•		•	•			•
C		•	•					
D						•		
E			•					
F		•		•	•			
G	•							•
.		•	•	•	•			
.		•	•	•	•			
.						•		
K	•			•				

many more combinations of attributes than algorithms

❑ Elementary sort may be method of choice for some combination. Cannot cover all combinations of attributes.

Q. Is the system sort good enough?

A. Maybe (no matter which algorithm it uses).

Data Structures & Algorithms

5. Priority Queues

- API
- elementary implementations
- binary heaps
- heapsort
- event-driven simulation



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

5. Priority Queues

- API
- elementary implementations
- binary heaps
- heapsort
- event-driven simulation

Priority Queues

- Data. Items that can be compared.
- Basic operations.

- ◆ Defining Ops

- ✓ Insert.
 - ✓ Remove largest.

- ◆ Generic Ops

- ✓ Copy.
 - ✓ Create.
 - ✓ Destroy.
 - ✓ Test if empty.

insert E →

insert X →

insert A →

insert M →

insert P →

insert L →

insert E →

E

E

E

E

E

E

E

A

X

X

A

M

P

L

E

E

remove largest → X

remove largest → M

remove largest → P

remove largest → L

remove largest → E

remove largest → E

remove largest → A

Priority Queue Applications

- Event-driven simulation. [customers in a line, colliding particles]
 - Numerical computation. [reducing roundoff error]
 - Data compression. [Huffman codes]
 - Graph searching. [Dijkstra's algorithm, Prim's algorithm]
 - Computational number theory. [sum of powers]
 - Artificial intelligence. [A* search]
 - Statistics. [maintain largest M values in a sequence]
 - Operating systems. [load balancing, interrupt handling]
 - Discrete optimization. [bin packing, scheduling]
 - Spam filtering. [Bayesian spam filter]
-
- Generalizes: stack, queue, randomized queue.

Priority queue client example

- Problem: Find the largest M of a stream of N elements.
 - ◆ Fraud detection: isolate \$\$ transactions.
 - ◆ File maintenance: find biggest files or directories.
- Constraint. Not enough memory to store N elements.
- Solution. Use a priority queue.

Operation	time	space
sort	$N \lg N$	N
elementary PQ	$M N$	M
binary heap	$N \lg M$	M
best in theory	N	M

```
MinPQ<Transaction> pq
    = new MinPQ<Transaction>();

while(!StdIn.isEmpty())
{
    String s = StdIn.readLine();
    t = new Transaction(s);
    pq.insert(t);
    if (pq.size() > M)
        pq.delMin();
}

while (!pq.isEmpty())
    System.out.println(pq.delMin());
```

5. Priority Queues

- API
- elementary implementations
- binary heaps
- heapsort
- event-driven simulation

Priority queue: unordered array implementation

```
public class UnorderedPQ<Item extends Comparable>
{
    private Item[] pq; // pq[i] = ith element on PQ
    private int N; // number of elements on PQ

    public UnorderedPQ(int maxN)
    { pq = (Item[]) new Comparable[maxN]; }

    public boolean isEmpty()
    { return N == 0; }

    public void insert(Item x)
    { pq[N++] = x; }

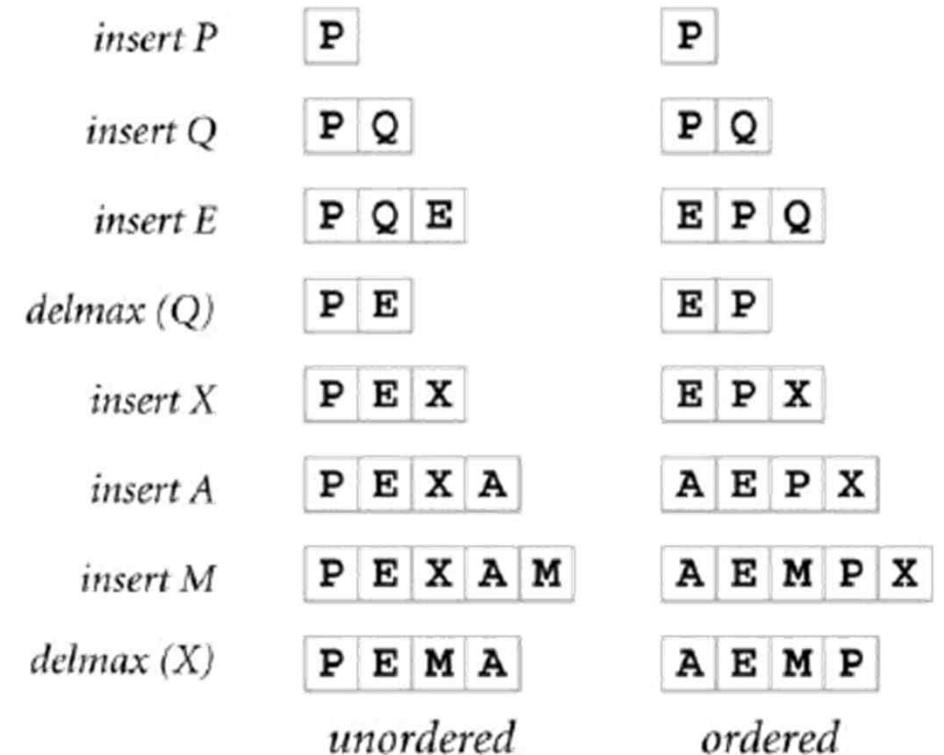
    public Item delMax()
    {
        int max = 0;
        for (int i = 1; i < N; i++)
            if (less(max, i)) max = i;
        exch(max, N-1);
        return pq[--N];
    }
}
```

no generic array creation

Priority queue elementary implementations

Implementation	Insert	Del Max
unordered array	1	N
ordered array	N	1

worst-case asymptotic costs for PQ with N items



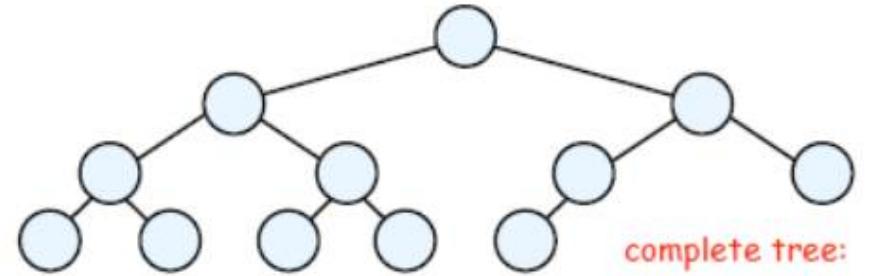
Challenge. Implement **both** operations efficiently.

5. Priority Queues

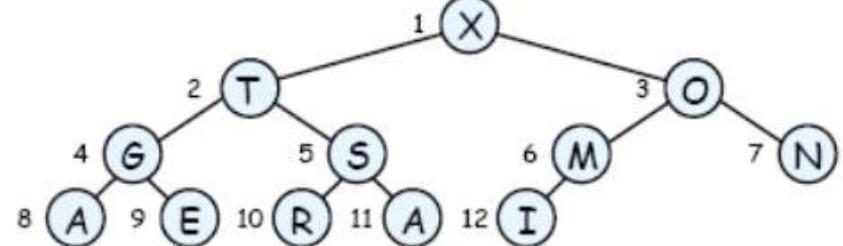
- API
- elementary implementations
- **binary heaps**
- **heapsort**
- event-driven simulation

Binary Heap

- **Heap:** Array representation of a heap-ordered complete binary tree.
- **Binary tree.**
 - ◆ Empty or
 - ◆ Node with links to left and right trees.
- **Heap-ordered binary tree.**
 - ◆ Keys in nodes.
 - ◆ No smaller than children's keys.
- **Array representation.**
 - ◆ Take nodes in level order.
 - ◆ No explicit links needed since tree is complete.



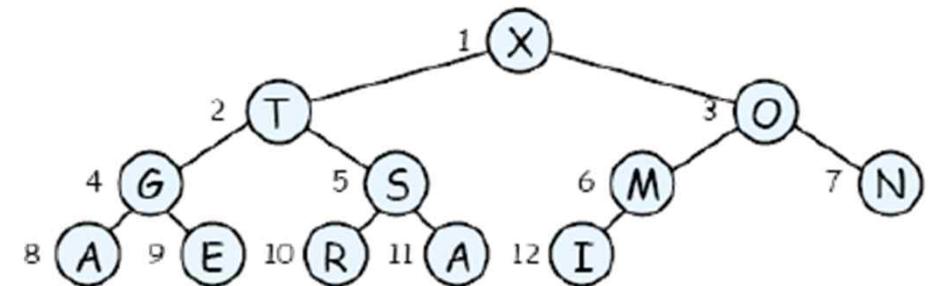
complete tree:
balanced except
for bottom level



1	2	3	4	5	6	7	8	9	10	11	12
X	T	O	G	S	M	N	A	E	R	A	I

Binary Heap Properties

- Property A. Largest key is at root.



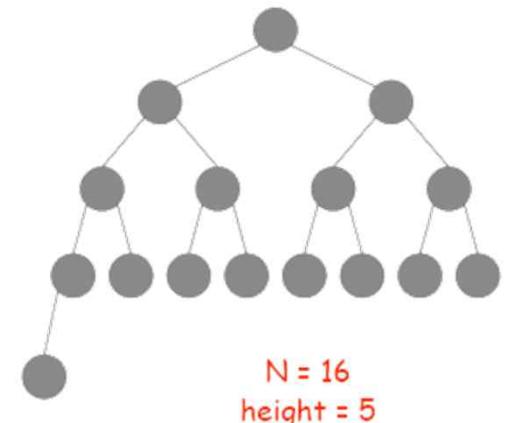
- Property B. Can use array indices to move through tree.

- ◆ Note: indices start at 1.
- ◆ Parent of node at k is at $k/2$.
- ◆ Children of node at k are at $2k$ and $2k+1$.

1	2	3	4	5	6	7	8	9	10	11	12
X	T	O	G	S	M	N	A	E	R	A	I

- Property C. Height of N node heap is $1 + \lfloor \lg N \rfloor$.

height increases only when
N is a power of 2



N = 16
height = 4

Promotion In a Heap

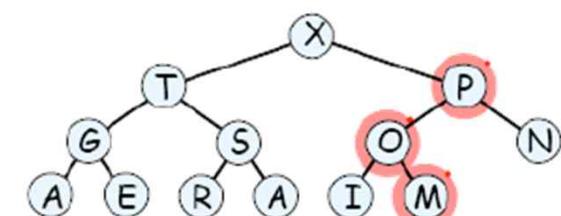
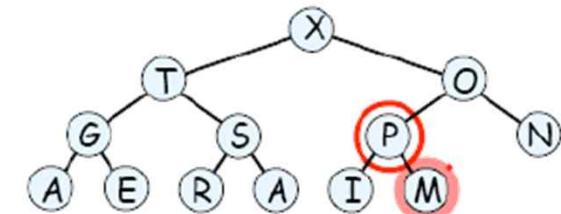
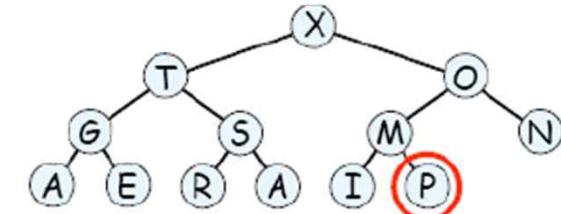
- Scenario. Exactly one node has a **larger** key than its parent.

- To eliminate the violation:

- ◆ Exchange with its **parent**.
- ◆ Repeat until **heap order** restored.

```
private void swim(int k)
{
    while (k > 1 && less(k/2, k))
    {
        exch(k, k/2);
        k = k/2;
    }
}
```

parent of node at k is at k/2

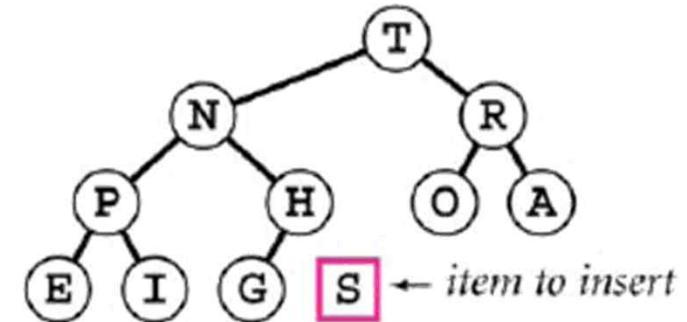


- Peter principle: node promoted to level of incompetence.

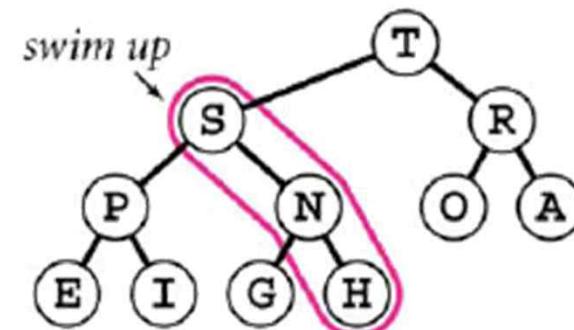
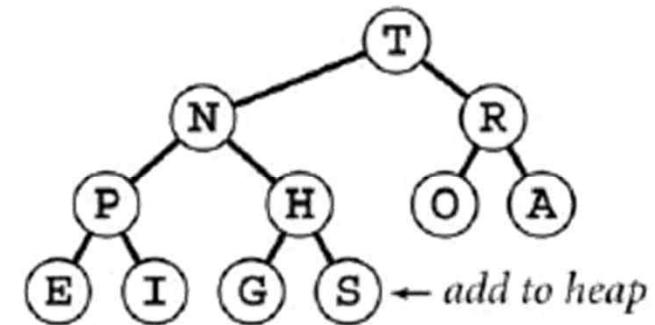
1	2	3	4	5	6	7	8	9	10	11	12	13
X	T	O	G	S	M	N	A	E	R	A	I	P
X	T	P	G	S	O	N	A	E	R	A	I	M

Insert

- Insert. Add node at end, then promote.



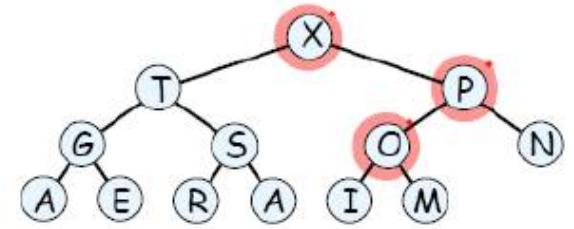
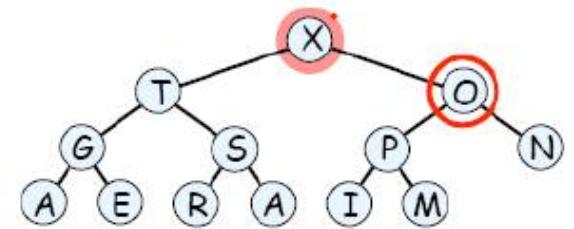
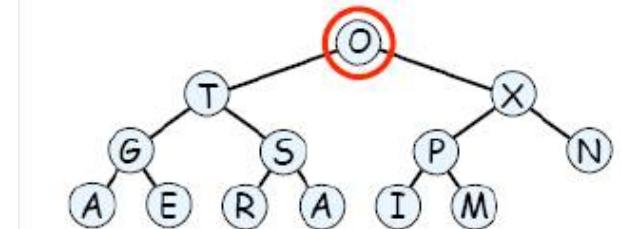
```
public void insert(Item x)
{
    pq[++N] = x;
    swim(N);
}
```



Demotion In a Heap

- Scenario. Exactly one node has a **smaller** key than does a child.
- To eliminate the violation:
 - ◆ Exchange with **larger child**.
 - ◆ Repeat until heap order restored.

```
private void sink(int k)
{
    while (2*k <= N)      children of node
    {                      at k are 2k and 2k+1
        int j = 2*k;
        if (j < N && less(j, j+1)) j++;
        if (!less(k, j)) break;
        exch(k, j);
        k = j;
    }
}
```



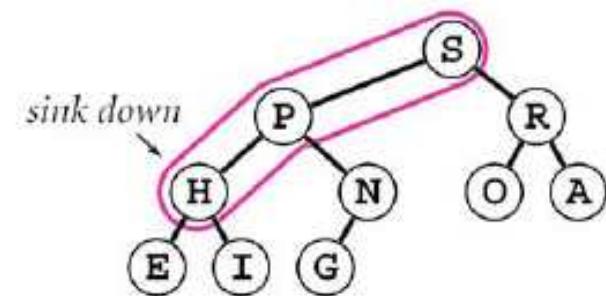
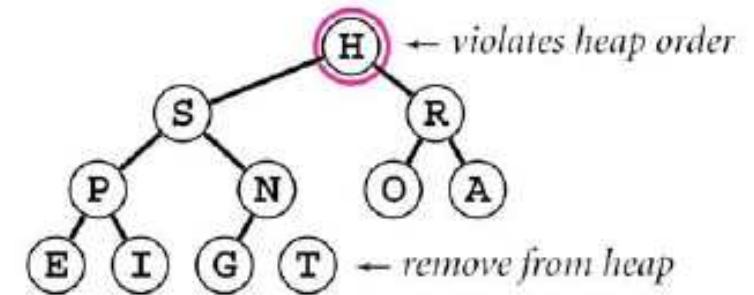
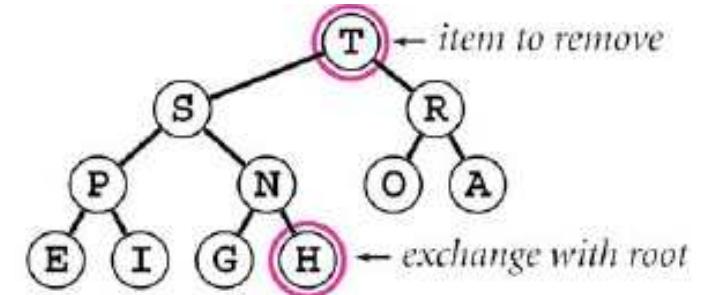
1	2	3	4	5	6	7	8	9	10	11	12	13
O	T	X	G	S	P	N	A	E	R	A	I	M
X	T	P	G	S	O	N	A	E	R	A	I	M

- Power struggle: better subordinate promoted.

Remove the Maximum

- Remove max. Exchange root with node at end, then demote.

```
public Item delMax()
{
    Item max = pq[1];
    exch(1, N--);
    sink(1);
    pq[N+1] = null; ← prevent loitering
    return max;
}
```



Binary heap implementation summary

```
public class MaxPQ<Item extends Comparable>
{
    private Item[] pq;
    private int N;

    public MaxPQ(int maxN)
    { . . . }
    public boolean isEmpty()
    { . . . }

    public void insert(Item x)
    { . . . }
    public Item delMax()
    { . . . }

    private void swim(int k)
    { . . . }
    private void sink(int k)
    { . . . }

    private boolean less(int i, int j)
    { . . . }
    private void exch(int i, int j)
    { . . . }
}
```

same as array-based PQ,
but allocate one extra element

PQ ops

heap helper functions

array helper functions



Binary heap considerations

- Minimum oriented priority queue
 - ◆ replace less() with greater()
 - ◆ implement greater().
- Array resizing
 - ◆ add no-arg constructor
 - ◆ apply repeated doubling. ← leads to $O(\log N)$ amortized time per op
- Immutability of keys.
 - ◆ assumption: client does not change keys while they're on the PQ
 - ◆ best practice: use immutable keys
- Other operations.
 - ◆ remove an arbitrary item. ← easy to implement with sink() and swim()
 - ◆ change the priority of an item. ← [stay tuned]

Priority Queues Implementation Cost Summary

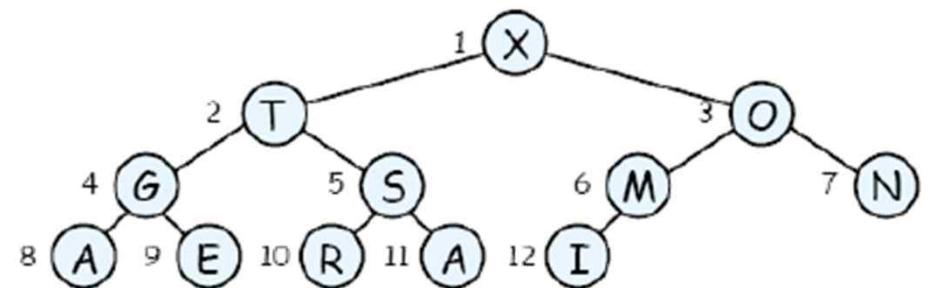
Operation	Insert	Remove Max	Find Max
ordered array	N	1	1
ordered list	N	1	1
unordered array	1	N	N
unordered list	1	N	N
binary heap	$\lg N$	$\lg N$	1

worst-case asymptotic costs for PQ with N items

- Hopeless challenge. Make all ops $O(1)$.
- Why hopeless?

5. Priority Queues

- API
- elementary implementations
- binary heaps
- **heapsort**
- event-driven simulation



1	2	3	4	5	6	7	8	9	10	11	12
X	T	O	G	S	M	N	A	E	R	A	I

Digression: Heapsort

- ❑ First pass: build heap.
 - ◆ Insert items into heap, one at at time.
 - ◆ Or can use faster bottom-up method; see book.

```
for (int k = N/2; k >= 1; k--)  
    sink(a, k, N);
```

- ❑ Second pass: sort.
 - ◆ Remove maximum items, one at a time.
 - ◆ Leave in array, instead of nulling out.

```

while (N > 1)
{
    exch(a, 1, N--);
    sink(a, 1, N);
}

```

- Property D. At most $2 N \lg N$ comparisons.



Significance of Heapsort

Q. Sort in $O(N \log N)$ worst-case without using extra memory?

A. Yes. Heapsort.

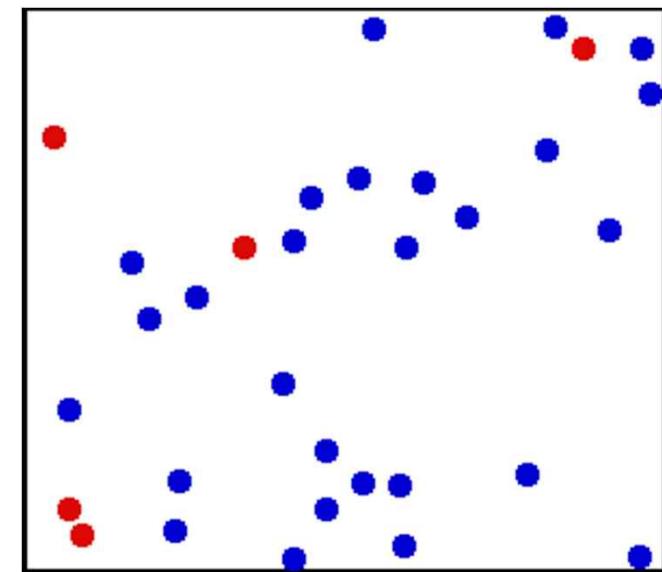
- Not mergesort? Linear extra space. ← in-place merge possible,
not practical
- Not quicksort? Quadratic time in worst case. ← $O(N \log N)$ worst-case
quicksort possible, not
practical.
- Heapsort is **optimal** for both time and space, **but**:
 - ◆ inner loop longer than quicksort's.
 - ◆ makes poor use of cache memory.

Sorting algorithms: summary

	inplace	stable	worst	average	best	remarks
selection	x		$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
insertion	x	x	$N^2 / 2$	$N^2 / 4$	N	use for small N or partly ordered
shell	x				N	tight code
quick	x		$N^2 / 2$	$2N \ln N$	$N \lg N$	$N \log N$ probabilistic guarantee fastest in practice
merge		x	$N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee, stable
heap	x		$2N \lg N$	$2N \lg N$	$N \lg N$	$N \log N$ guarantee, in-place

5. Priority Queues

- API
- elementary implementations
- binary heaps
- heapsort
- event-driven simulation



Basic Framework

□ Bouncing balls

```
public class BouncingBalls
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        Ball balls[] = new Ball[N];
        for (int i = 0; i < N; i++)
            balls[i] = new Ball();
        while(true)
        {
            StdDraw.clear();
            for (int i = 0; i < N; i++)
            {
                balls[i].move();
                balls[i].draw();
            }
            StdDraw.show(50);
        }
    }
}
```

Basic Framework

□ Bouncing balls

```
public class Ball
{
    private double rx, ry;      // position
    private double vx, vy;      // velocity
    private double radius;      // radius
    public Ball()
    { ... initialize position and velocity ... }
    public void move()
    {
        if ((rx + vx < radius) || (rx + vx > 1.0 - radius)) { vx = -vx; }
        if ((ry + vy < radius) || (ry + vy > 1.0 - radius)) { vy = -vy; }
        rx = rx + vx;
        ry = ry + vy;
    }
    public void draw()
    { StdDraw.filledCircle(rx, ry, radius); }
}
```

checks for
colliding with
walls



□ Missing: check for balls colliding with each other

- ◆ physics problems: when? what effect?
- ◆ CS problems: what object does the checks? too many checks?

Molecular dynamics simulation of hard spheres

Goal. Simulate the motion of N moving particles that behave according to the laws of elastic collision.

□ Hard sphere model.

- ◆ Moving particles interact via elastic collisions with each other, and with fixed walls.
- ◆ Each particle is a sphere with known position, velocity, mass, and radius.
- ◆ No other forces are exerted.

temperature, pressure,
diffusion constant

motion of individual
atoms and molecules

□ Significance. Relates **macroscopic** observables to **microscopic** dynamics.

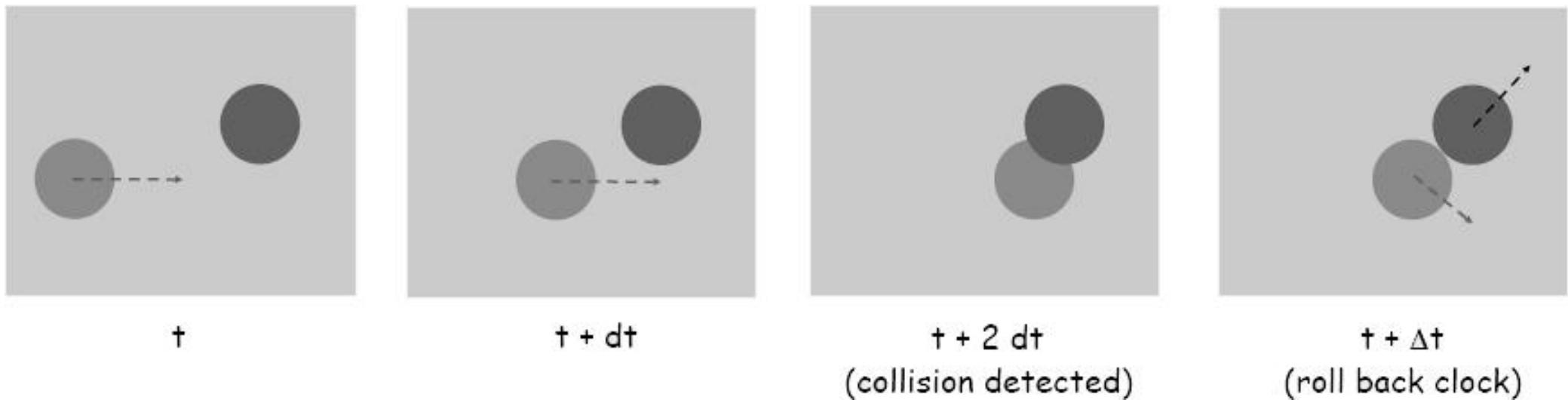
- ◆ Maxwell and Boltzmann: derive distribution of speeds of interacting molecules as a function of temperature.
- ◆ Einstein: explain Brownian motion of pollen grains.

kinetic theory of
gases



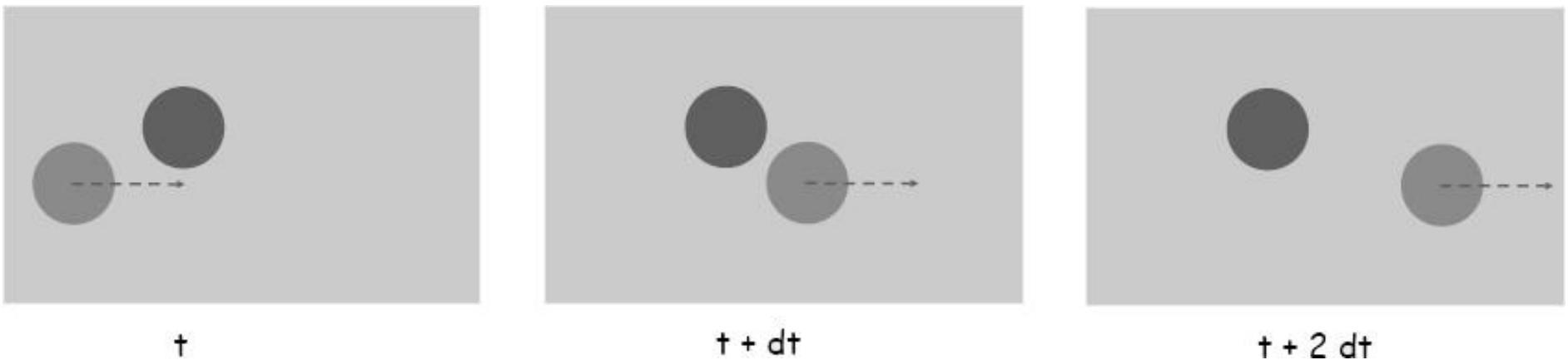
Time-driven simulation

- Time-driven simulation.
 - ◆ Discretize **time** in quanta of size dt .
 - ◆ Update the **position** of each particle after every dt units of time, and check for **overlaps**.
 - ◆ If overlap, roll back the clock to the time of the **collision**, update the velocities of the colliding particles, and continue the simulation.



Time-driven simulation

- Main drawbacks.
 - ◆ N^2 overlap checks per time quantum.
 - ◆ May miss collisions if dt is too large and colliding particles fail to overlap when we are looking.
 - ◆ Simulation is too slow if dt is very small.



Event-driven simulation

- Change state only when something happens.
 - ◆ Between collisions, particles move in straight-line trajectories.
 - ◆ Focus **only** on times when collisions occur.
 - ◆ Maintain **priority queue** of collision events, prioritized by time.
 - ◆ Remove the minimum = get next collision.
- Collision prediction. Given position, velocity, and radius of a particle, when will it collide next with a wall or another particle?
- Collision resolution. If collision occurs, update colliding particle(s) according to laws of elastic collisions.
- Note: Same approach works for a broad variety of systems

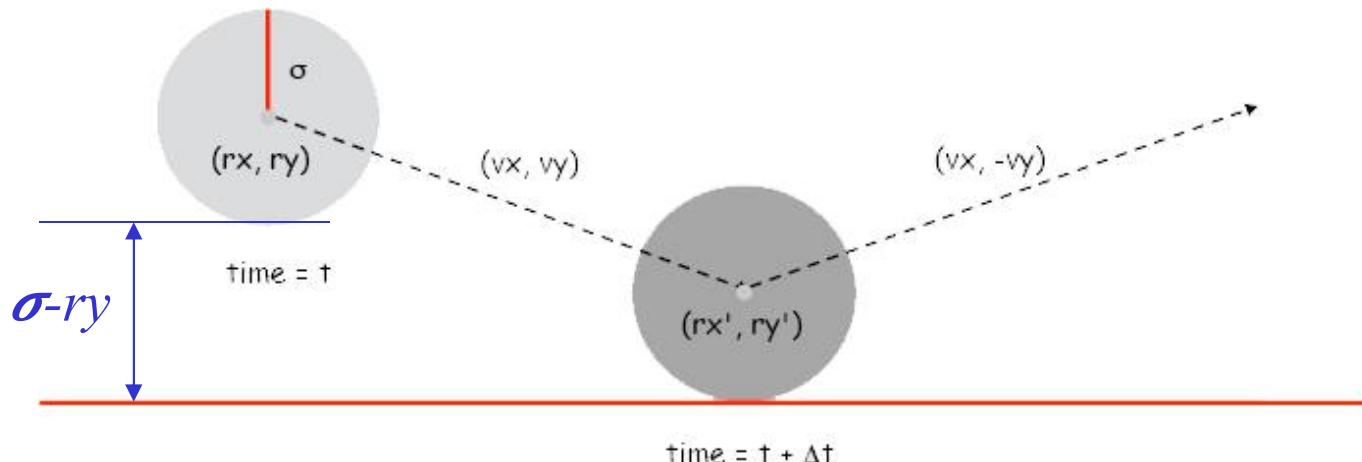
Particle-wall collision

□ Collision prediction.

- ◆ Particle of radius σ at position (rx, ry) .
- ◆ Particle moving in unit box with velocity (vx, vy) .
- ◆ Will it collide with a horizontal wall? If so, when?

$$\Delta t = \begin{cases} \infty & \text{if } vy = 0 \\ (\sigma - ry)/vy & \text{if } vy < 0 \\ (1 - \sigma - ry)/vy & \text{if } vy > 0 \end{cases}$$

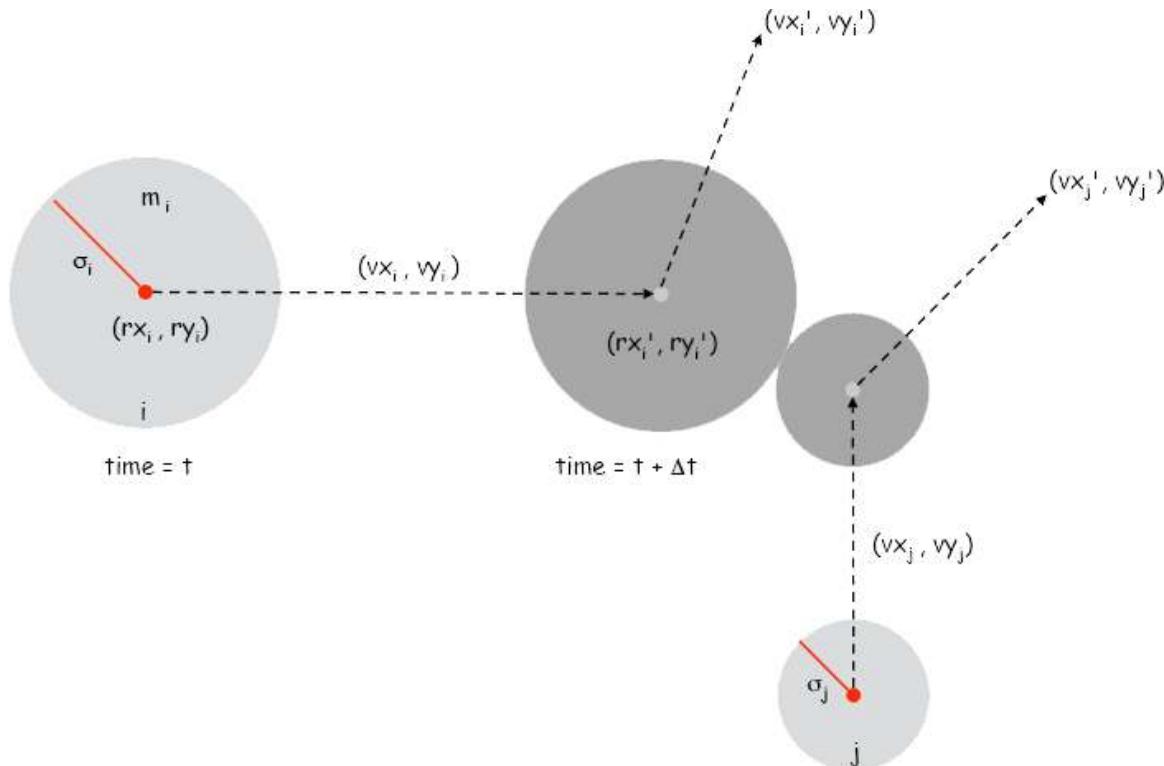
□ Collision resolution. $(vx', vy') = (vx, -vy)$.



Particle-particle collision prediction

□ Collision prediction.

- ◆ Particle i: radius σ_i , position (rx_i, ry_i) , velocity (vx_i, vy_i) .
- ◆ Particle j: radius σ_j , position (rx_j, ry_j) , velocity (vx_j, vy_j) .
- ◆ Will particles i and j collide? If so, when?



Particle-particle collision prediction

□ Collision prediction.

- ◆ Particle i: radius σ_i , position (rx_i, ry_i) , velocity (vx_i, vy_i) .
- ◆ Particle j: radius σ_j , position (rx_j, ry_j) , velocity (vx_j, vy_j) .
- ◆ Will particles i and j collide? If so, when?

$$\Delta t = \begin{cases} \infty & \text{if } \Delta v \cdot \Delta r \geq 0 \\ \infty & \text{if } d < 0 \\ -\frac{\Delta v \cdot \Delta r + \sqrt{d}}{\Delta v \cdot \Delta v} & \text{otherwise} \end{cases}$$

$$d = (\Delta v \cdot \Delta r)^2 - (\Delta v \cdot \Delta v) (\Delta r \cdot \Delta r - \sigma^2) \quad \sigma = \sigma_i + \sigma_j$$

$$\Delta v = (\Delta vx, \Delta vy) = (vx_i - vx_j, vy_i - vy_j)$$

$$\Delta r = (\Delta rx, \Delta ry) = (rx_i - rx_j, ry_i - ry_j)$$

$$\Delta v \cdot \Delta v = (\Delta vx)^2 + (\Delta vy)^2$$

$$\Delta r \cdot \Delta r = (\Delta rx)^2 + (\Delta ry)^2$$

$$\Delta v \cdot \Delta r = (\Delta vx)(\Delta rx) + (\Delta vy)(\Delta ry)$$

Particle-particle collision prediction implementation

- Particle has method to predict collision with another particle

```
public double dt(Particle b)
{
    Particle a = this;
    if (a == b) return INFINITY;
    double dx = b.rx - a.rx;
    double dy = b.ry - a.ry;
    double dvx = b.vx - a.vx;
    double dvy = b.vy - a.vy;
    double dvdr = dx*dvx + dy*dvy;
    if(dvdr > 0) return INFINITY;
    double dvdv = dvx*dvx + dvy*dvy;
    double drdr = dx*dx + dy*dy;
    double sigma = a.radius + b.radius;
    double d = (dvdr*dvdr) - dvdv * (drdr - sigma*sigma);
    if (d < 0) return INFINITY;
    return -(dvdr + Math.sqrt(d)) / dvdv;
}
```

and methods `dtX()` and `dtY()` to predict collisions with walls

Particle-particle collision prediction implementation

- ❑ CollisionSystem has method to predict all collisions

```
private void predict(Particle a, double limit)
{
    if (a == null) return;
    for(int i = 0; i < N; i++)
    {
        double dt = a.dt(particles[i]);
        if(t + dt <= limit)
            pq.insert(new Event(t + dt, a, particles[i]));
    }
    double dtX = a.dtX();
    double dtY = a.dtY();
    if (t + dtX <= limit)
        pq.insert(new Event(t + dtX, a, null));
    if (t + dtY <= limit)
        pq.insert(new Event(t + dtY, null, a));
}
```

Particle-particle collision resolution

- Collision resolution. When two particles collide, how does velocity change?

$$vx_i' = vx_i + Jx / m_i$$

$$vy_i' = vy_i + Jy / m_i$$

$$vx_j' = vx_j - Jx / m_j$$

$$vy_j' = vy_j - Jy / m_j$$

Newton's second law
(momentum form)

$$Jx = \frac{J \Delta rx}{\sigma}, \quad Jy = \frac{J \Delta ry}{\sigma}, \quad J = \frac{2m_i m_j (\Delta v \cdot \Delta r)}{\sigma(m_i + m_j)}$$

impulse due to normal force
(conservation of energy, conservation of momentum)

Particle-particle collision resolution implementation

- Particle has method to **resolve** collision with another particle

```
public void bounce(Particle b)
{
    Particle a = this;
    double dx  = b.rx - a.rx;
    double dy  = b.ry - a.ry;
    double dvx = b.vx - a.vx;
    double dvy = b.vy - a.vy;
    double dvdr = dx*dvx + dy*dvy;
    double dist = a.radius + b.radius;
    double J = 2 * a.mass * b.mass * dvdr / ((a.mass + b.mass) * dist);
    double Jx = J * dx / dist;
    double Jy = J * dy / dist;
    a.vx += Jx / a.mass;
    a.vy += Jy / a.mass;
    b.vx -= Jx / b.mass;
    b.vy -= Jy / b.mass;
    a.count++;
    b.count++;
}
```

and methods **bounceX()** and **bounceY()** to resolve collisions with walls

Collision system: event-driven simulation main loop

□ Initialization.

- ◆ Fill PQ with all potential particle-wall collisions
- ◆ Fill PQ with all potential particle-particle collisions.



"potential" since collision may not happen if some other collision intervenes

□ Main loop.

- ◆ Delete the impending event from PQ (min priority = t).
- ◆ If the event is no longer valid, ignore it.
- ◆ Advance all particles to time t , on a straight-line trajectory.
- ◆ Update the **velocities** of the colliding particle(s).
- ◆ Predict **future** particle-wall and particle-particle **collisions** involving the colliding particle(s) and insert events onto PQ.

Collision system: main event-driven simulation loop implementation

```
public void simulate(double limit)
{
    pq = new MinPQ<Event>();
    for(int i = 0; i < N; i++)
        predict(particles[i], limit);
    pq.insert(new Event(0, null, null));
    while(!pq.isEmpty())
    {
        Event e = pq.delMin();
        if(!e.isValid()) continue;
        Particle a = e.a();
        Particle b = e.b();

        for(int i = 0; i < N; i++)
            particles[i].move(e.time() - t);
        t = e.time();

        if      (a != null && b != null) a.bounce(b);
        else if (a != null && b == null) a.bounceX();
        else if (a == null && b != null) b.bounceY();
        else if (a == null && b == null)
        {
            StdDraw.clear(StdDraw.WHITE);
            for(int i = 0; i < N; i++) particles[i].draw();
            StdDraw.show(20);
            if (t < limit)
                pq.insert(new Event(t + 1.0 / Hz, null, null));
        }
        predict(a, limit);
        predict(b, limit);
    }
}
```

initialize PQ with collision events and redraw event

main event-driven simulation loop

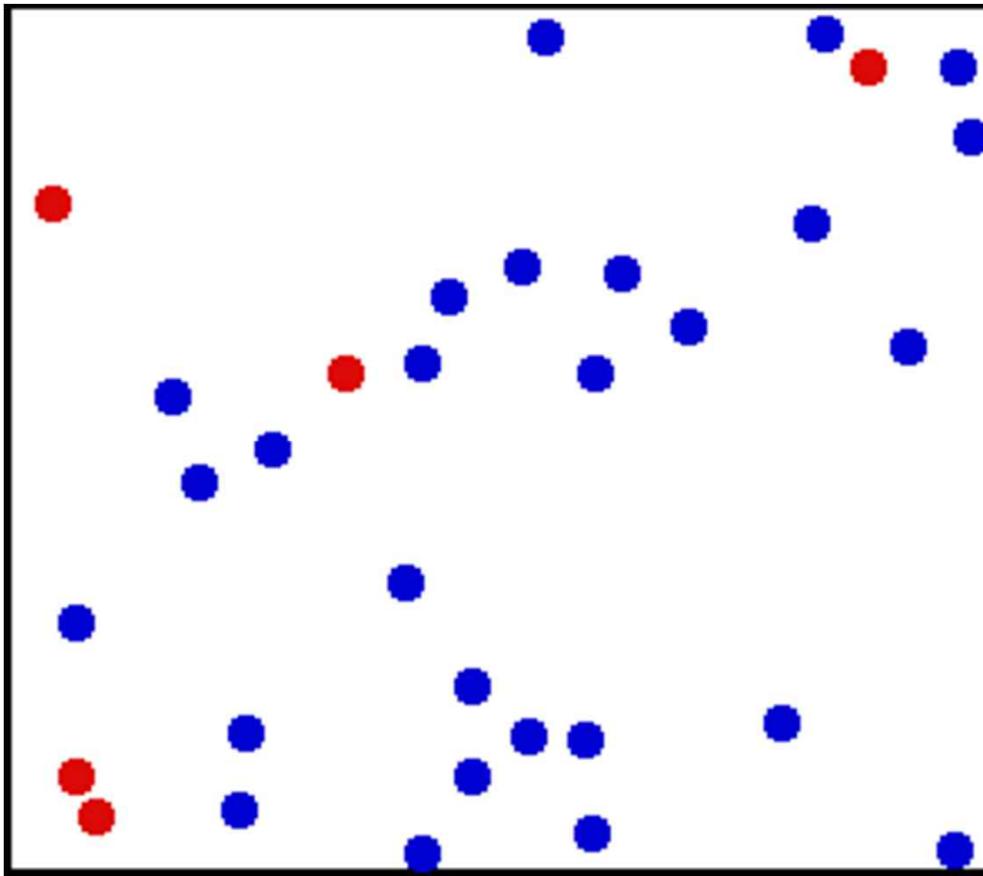
update positions and time

process event

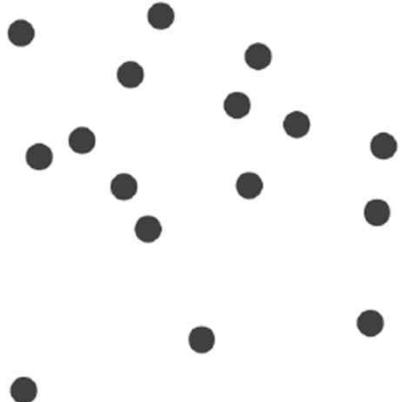
predict new events based on changes

Random collisions of particles in a gas

- <http://algs4.cs.princeton.edu/61event/>
 - ◆ MinPQ.java, Particle.java, CollisionSystem.java



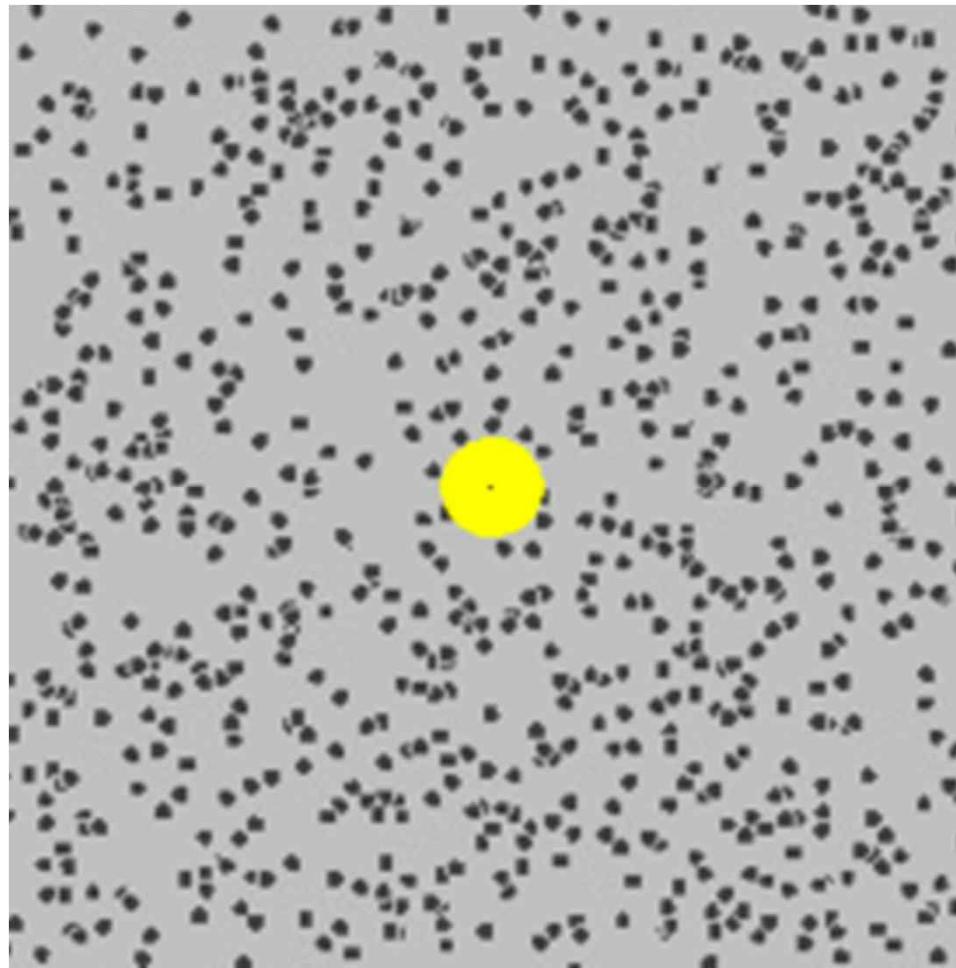
```
java CollisionSystem < billiards5.txt
```



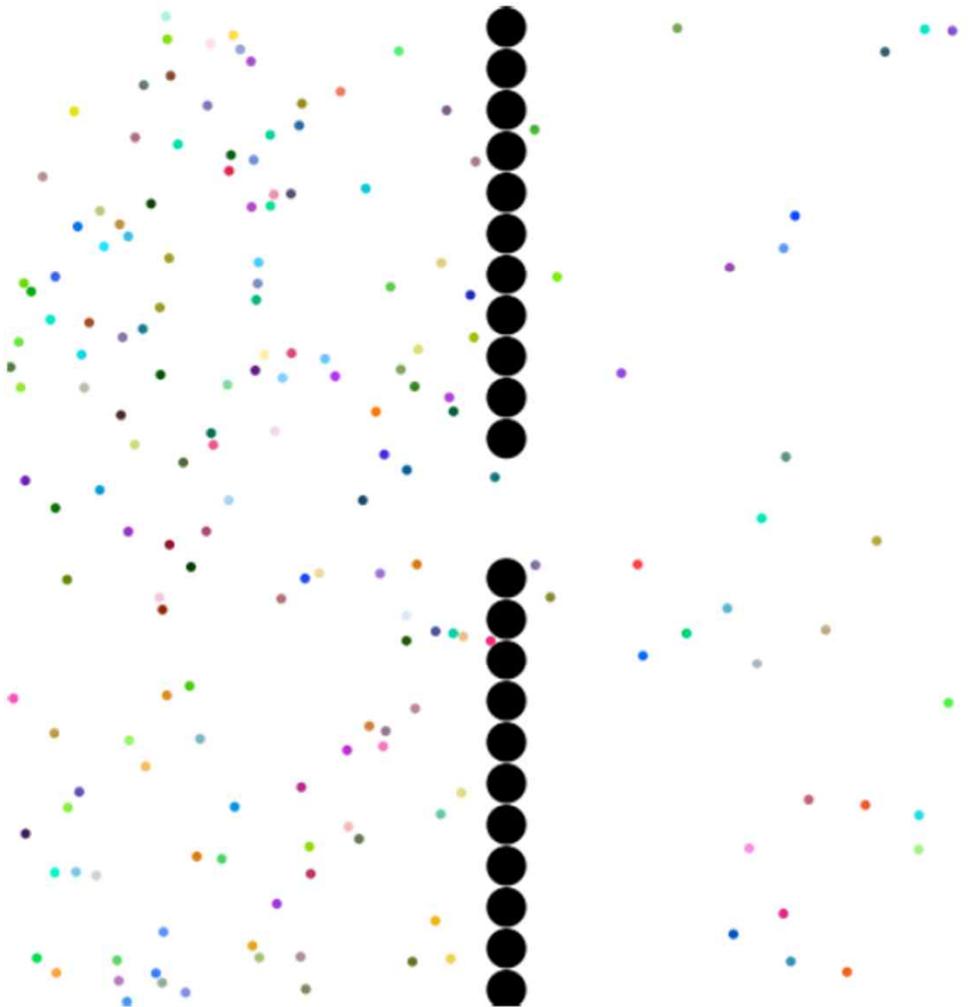
```
java CollisionSystem < squeeze2.txt
```



The Brownian motion of a big particle



```
java CollisionSystem < diffusion.txt
```



Data Structures & Algorithms

6. Symbol Table

- API
- basic implementations
- iterators
- Comparable keys
- challenges



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

6. Symbol Table

- API
- basic implementations
- iterators
- Comparable keys
- challenges

Symbol Tables

- Key-value pair abstraction.
 - ◆ Insert a value with specified key.
 - ◆ Given a key, search for the corresponding value.
- Example: DNS lookup.
 - ◆ Insert URL with specified IP address.
 - ◆ Given URL, find corresponding IP address

Domain Name System
Uniform Resource Locator
Internet Protocol

URL	IP address
www.cs.princeton.edu	128.112.136.11
www.princeton.edu	128.112.128.15
www.yale.edu	130.132.143.21
www.harvard.edu	128.103.060.55
www.simpsons.com	209.052.165.60

- Can interchange roles: given IP address find corresponding URL



Symbol Table Applications

Application	Purpose	Key	Value
Phone book	Look up phone number	Name	Phone number
Bank	Process transaction	Account number	Transaction details
File share	Find song to download	Name of song	Computer ID
File system	Find file on disk	Filename	Location on disk
Dictionary	Look up word	Word	Definition
Web search	Find relevant documents	Keyword	List of documents
Book index	Find relevant pages	Keyword	List of pages
Web cache	Download	Filename	File contents
Genomics	Find markers	DNA string	Known positions
DNS	Find IP address given URL	URL	IP address
Reverse DNS	Find URL given IP address	IP address	URL
Compiler	Find properties of variable	Variable name	Value and type
Routing table	Route Internet packets	Destination	Best route

Symbol Table API

- **Associative array abstraction:** Unique value associated with each key.

```
public class *ST<Key extends Comparable<Key>, Value>
```

*ST()	create a symbol table	
void put(Key key, Value val)	put key-value pair into the table	insert
Value get(Key key)	return value paired with key (null if key not in table)	search
boolean contains(Key key)	is there a value paired with key?	
void remove(Key key)	remove key-value pair from table	
Iterator<Key> iterator()	iterator through keys in table	> stay tuned

- Our conventions:

1. Values are not **null**.
2. Method **get()** returns **null** if key not present

enables this code in all implementations:

```
public boolean contains(Key key)
{ return get(key) != null; }
```

3. Method **put()** overwrites old value with new value.

a[key] = val; ← Some languages (not Java) allow this notation

ST client: make a dictionary and process lookup requests

□ Command line arguments

- ◆ a comma-separated value (CSV) file
- ◆ key field
- ◆ value field

```
% more ip.csv
www.princeton.edu,128.112.128.15
www.cs.princeton.edu,128.112.136.35
www.math.princeton.edu,128.112.18.11
www.cs.harvard.edu,140.247.50.127
www.harvard.edu,128.103.60.24
www.yale.edu,130.132.51.8
www.econ.yale.edu,128.36.236.74
www.cs.yale.edu,128.36.229.30
espn.com,199.181.135.201
yahoo.com,66.94.234.13
msn.com,207.68.172.246
google.com,64.233.167.99
baidu.com,202.108.22.33
yahoo.co.jp,202.93.91.141
sina.com.cn,202.108.33.32
ebay.com,66.135.192.87
adobe.com,192.150.18.60
163.com,220.181.29.154
passport.net,65.54.179.226
tom.com,61.135.158.237
nate.com,203.226.253.11
cnn.com,64.236.16.20
daum.net,211.115.77.211
blogger.com,66.102.15.100
fastclick.com,205.180.86.4
wikipedia.org,66.230.200.100
rakuten.co.jp,202.72.51.22
...
```

□ Example 1: DNS lookup

```
% java Lookup ip.csv 0 1
adobe.com
192.150.18.60
www.princeton.edu
128.112.128.15
ebay.edu
Not found
```

URL is key IP is value


```
% java Lookup ip.csv 1 0
128.112.128.15
www.princeton.edu
999.999.999.99
Not found
```

IP is key URL is value

ST client: make a dictionary and process lookup requests

```
public class Lookup
{
    public static void main(String[] args)
    {
        In in = new In(args[0]);
        int keyField = Integer.parseInt(args[1]);
        int valField = Integer.parseInt(args[2]);
        String[] database = in.readAll().split("\n");

        ST<String, String> st = new ST<String, String>();
        for (int i = 0; i < database.length; i++)
        {
            String[] tokens = database[i].split(",");
            String key = tokens[keyField];
            String val = tokens[valField];
            st.put(key, val);
        }

        while (!StdIn.isEmpty())
        {
            String s = StdIn.readString();
            if (!st.contains(s)) StdOut.println("Not found");
            else                 StdOut.println(st.get(s));
        }
    }
}
```

← process input file

← build symbol table

← process lookups with standard I/O



ST client: make a dictionary and process lookup requests

□ Command line arguments

- ◆ a comma-separated value (CSV) file
- ◆ key field
- ◆ value field

□ Example 2: Amino acids

```
% % java Lookup amino.csv 0 3
ACT
Threonine
TAG
Stop
CAT
Histidine
```

codon is key name is value

% more amino.csv
TTT,Phe,F,Phenylalanine
TTC,Phe,F,Phenylalanine
TTA,Leu,L,Leucine
TTG,Leu,L,Leucine
TCT,Ser,S,Serine
TCC,Ser,S,Serine
TCA,Ser,S,Serine
TCG,Ser,S,Serine
TAT,Tyr,Y,Tyrosine
TAC,Tyr,Y,Tyrosine
TAA,Stop,Stop,Stop
TAG,Stop,Stop,Stop
TGT,Cys,C,Cysteine
TGC,Cys,C,Cysteine
TGA,Stop,Stop,Stop
TGG,Trp,W,Tryptophan
CTT,Leu,L,Leucine
CTC,Leu,L,Leucine
CTA,Leu,L,Leucine
CTG,Leu,L,Leucine
CCT,Pro,P,Proline
CCC,Pro,P,Proline
CCA,Pro,P,Proline
CCG,Pro,P,Proline
CAT,His,H,Histidine
CAC,His,H,Histidine
CAA,Gln,Q,Glutamine
CAG,Gln,Q,Glutamine
CGT,Arg,R,Arginine
CGC,Arg,R,Arginine
CGA,Arg,R,Arginine
CGG,Arg,R,Arginine
ATT,Ile,I,Isoleucine
ATC,Ile,I,Isoleucine
ATA,Ile,I,Isoleucine
ATG,Met,M,Methionine
...

ST client: make a dictionary and process lookup requests

□ Command line arguments

- ◆ a comma-separated value (CSV) file
- ◆ key field
- ◆ value field

□ Example 3: Class lists

```
% java Lookup classlist.csv 3 1  
jsh  
Jeffrey Scott Harris  
dgtwo  
Daniel Gopstein  
ye  
Michael Weiyang Ye
```

login is key name is value

```
% java Lookup classlist.csv 3 2  
jsh  
P01A  
dgtwo  
P01
```

login is key precept is value

```
* more classlist.csv  
10,Bo Ling,P03,bling  
10,Steven A Ross,P01,saross  
10,Thomas Oliver Horton  
Conway,P03,oconway  
08,Michael R. Corces  
Zimmerman,P01A,mcorces  
09,Bruce David Halperin,P02,bhalperi  
09,Glenn Charles Snyders Jr.,P03,gsnyders  
09,Siyu Yang,P01A,siyuyang  
08,Taofik O. Kolade,P01,tkolade  
09,Katharine Paris  
Klosterman,P01A,kkloster  
SP,Daniel Gopstein,P01,dgtwo  
10,Sauhard Sahi,P01,ssahi  
10,Eric Daniel Cohen,P01A,edcohen  
09,Brian Anthony Geistwhite,P02,bgeistwh  
09,Boris Pivtorak,P01A,pivtorak  
09,Jonathan Patrick  
Zebrowski,P01A,jzebrows  
09,Dexter James Doyle,P01A,ddoyle  
09,Michael Weiyang Ye,P03,ye  
08,Delwin Uy Olivan,P02,dolivan  
08,Edward George Conbeer,P01A,econbeer  
09,Mark Daniel Stefanski,P01,mstefans  
09,Carter Adams Cleveland,P03,cclevela  
10,Jacob Stephen Lewellen,P02,jlewelle  
10,Ilya Trubov,P02,itrubov  
09,Kenton William Murray,P03,kwmurray  
07,Daniel Steven Marks,P02,dmarks  
09,Vittal Kadapakkam,P01,vkadapak  
10,Eric Ruben Domb,P01A,edomb  
07,Jie Wu,P03,jiewu  
08,Pritha Ghosh,P02,prithag  
10,Minh Quang Anh Do,P01,mqdo  
...
```



Keys and Values

- **Associative array abstraction.** `a[key] = val;`
 - ◆ Unique value associated with each key
 - ◆ If client presents duplicate key, overwrite to change value.
- **Key type:** several possibilities
 1. Assume keys are any generic type, use `equals()` to test equality.
 2. Assume keys are `Comparable`, use `compareTo()`.
 3. Use `equals()` to test equality and `hashCode()` to scramble key.
- **Value type.** Any `generic` type.
- **Best practices.** Use `immutable` types for symbol table keys.
 - ◆ Immutable in Java: `String`, `Integer`, `BigInteger`.
 - ◆ Mutable in Java: `Date`, `GregorianCalendar`, `StringBuilder`.

Elementary ST implementations

Unordered array

Ordered array

Unordered linked list

Ordered linked list

□ Why study elementary implementations?

- ◆ API details need to be worked out
- ◆ performance benchmarks
- ◆ method of choice can be one of these in many situations
- ◆ basis for advanced implementations

□ Always good practice to study elementary implementations



6. Symbol Table

- API
- basic implementations
- iterators
- Comparable keys
- challenges

Unordered array ST implementation

- Maintain **parallel arrays** of keys and values.
- Instance variables
 - ◆ array **keys[]** holds the keys.
 - ◆ array **vals[]** holds the values.
 - ◆ integer **N** holds the number of entries.

- Need to use standard array-doubling technique (chap. 2)

N = 6
↓

	0	1	2	3	4	5
keys[]	it	was	the	best	of	times
vals[]	2	2	1	1	1	1

- Alternative: define **inner type** for entries
 - ◆ space overhead for entry objects
 - ◆ more complicated code

Unordered array ST implementation (skeleton)

```
public class UnorderedST<Key, Value>
{
    private Value[] vals;
    private Key[] keys;
    private int N = 0;

    public UnorderedST(int maxN)
    {
        keys = (Key[]) new Object[maxN];
        vals = (Value[]) new Object[maxN];
    }

    public boolean isEmpty()
    {   return N == 0; }

    public void put(Key key, Value val)
    // see next slide

    public Value get(Key key)
    // see next slide
}
```

parallel arrays lead to cleaner code
than defining a type for entries

standard array doubling code omitted

standard ugly casts

Unordered array ST implementation (search)

```
public Value get(Key key)
{
    for (int i = 0; i < N; i++)
        if (keys[i].equals(key))
            return vals[i];
    return null;
}
```

	0	1	2	3	4	5
keys[]	it	was	the	best	of	times
vals[]	2	2	1	1	1	1
	↑	↑	↑			

get("the")
returns 1

	0	1	2	3	4	5
keys[]	it	was	the	best	of	times
vals[]	2	2	1	1	1	1
	↑	↑	↑	↑	↑	↑

get("worst")

- Key, Value are generic and can be **any** type

Unordered array ST implementation (search)

```
public void put(Key key, Value val)
{
    int i;
    for (i = 0; i < N; i++)
        if (key.equals(keys[i]))
            break;
    vals[i] = val;
    keys[i] = key;
    if (i == N) N++;
}
```

	0	1	2	3	4	5
keys[]	it	was	the	best	of	times
vals[]	2	2	1	1	1	1

	0	1	2	3	4	5
keys[]	it	was	the	best	of	times
vals[]	2	2	2	1	1	1

put("the", 2)
overwrites the 1

	0	1	2	3	4	5	6
keys[]	it	was	the	best	of	times	worst
vals[]	2	2	2	1	1	1	1

put("worst", 1)
adds a new entry

□ Associative array abstraction

- ◆ must search for key and **overwrite** with new value if it is there
- ◆ otherwise, **add** new key, value at the end (as in stack)



Java conventions for equals()

- All objects implement equals() but default implementation is $(x == y)$
 - is the object referred to by x the same object that is referred to by y ?
- Customized implementations.
 - ◆ String, URL, Integer.
- User-defined implementations.
 - ◆ Some care needed (example: type of argument must be Object)
- Equivalence relation. For any references x , y and z :
 - ◆ Reflexive: $x.equals(x)$ is true.
 - ◆ Symmetric: $x.equals(y)$ iff $y.equals(x)$.
 - ◆ Transitive: If $x.equals(y)$ and $y.equals(z)$, then $x.equals(z)$.
 - ◆ Non-null: $x.equals(null)$ is false.
 - ◆ Consistency: Multiple calls to $x.equals(y)$ return same value.

Implementing equals()

- Seems easy

```
public class PhoneNumber
{
    private int area, exch, ext;
    ...
    public boolean equals(PhoneNumber y)
    {
        PhoneNumber a = this;
        PhoneNumber b = (PhoneNumber) y;
        return (a.area == b.area)
            && (a.exch == b.exch)
            && (a.ext == b.ext);
    }
}
```



Implementing equals()

- Seems easy, but requires some care

```
public final class PhoneNumber
{
    private final int area, exch, ext;
    ...
    public boolean equals( Object y)
    {
        if (y == this) return true;           ← Optimize for true object equality
        if (y == null) return false;          ← If I'm executing this code,
                                                I'm not null.
        if (y.getClass() != this.getClass())
            return false;                  ← Objects must be in the same class.

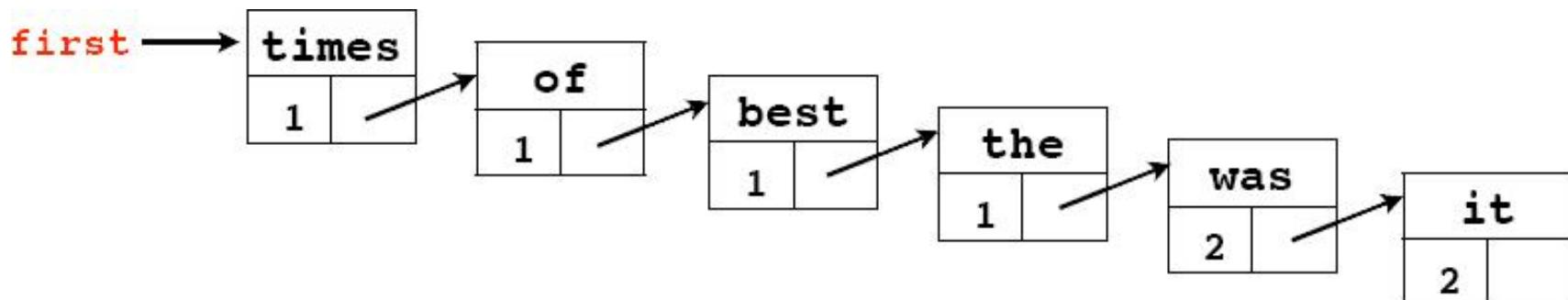
        PhoneNumber a = this;
        PhoneNumber b = (PhoneNumber) y;
        return (a.area == b.area)
            && (a.exch == b.exch)
            && (a.ext == b.ext);
    }
}
```

no safe way to use with inheritance

Must be Object.
Why? Experts still debate.

Linked list ST implementation

- Maintain a linked list with keys and values.
- inner Node class
 - ◆ instance variable **key** holds the key
 - ◆ instance variable **val** holds the value
- instance variable
 - ◆ Node **first** refers to the first node in the list



Linked list ST implementation (skeleton)

```
public class LinkedListST<Key, Value>
{
    private Node first;           ← instance variable

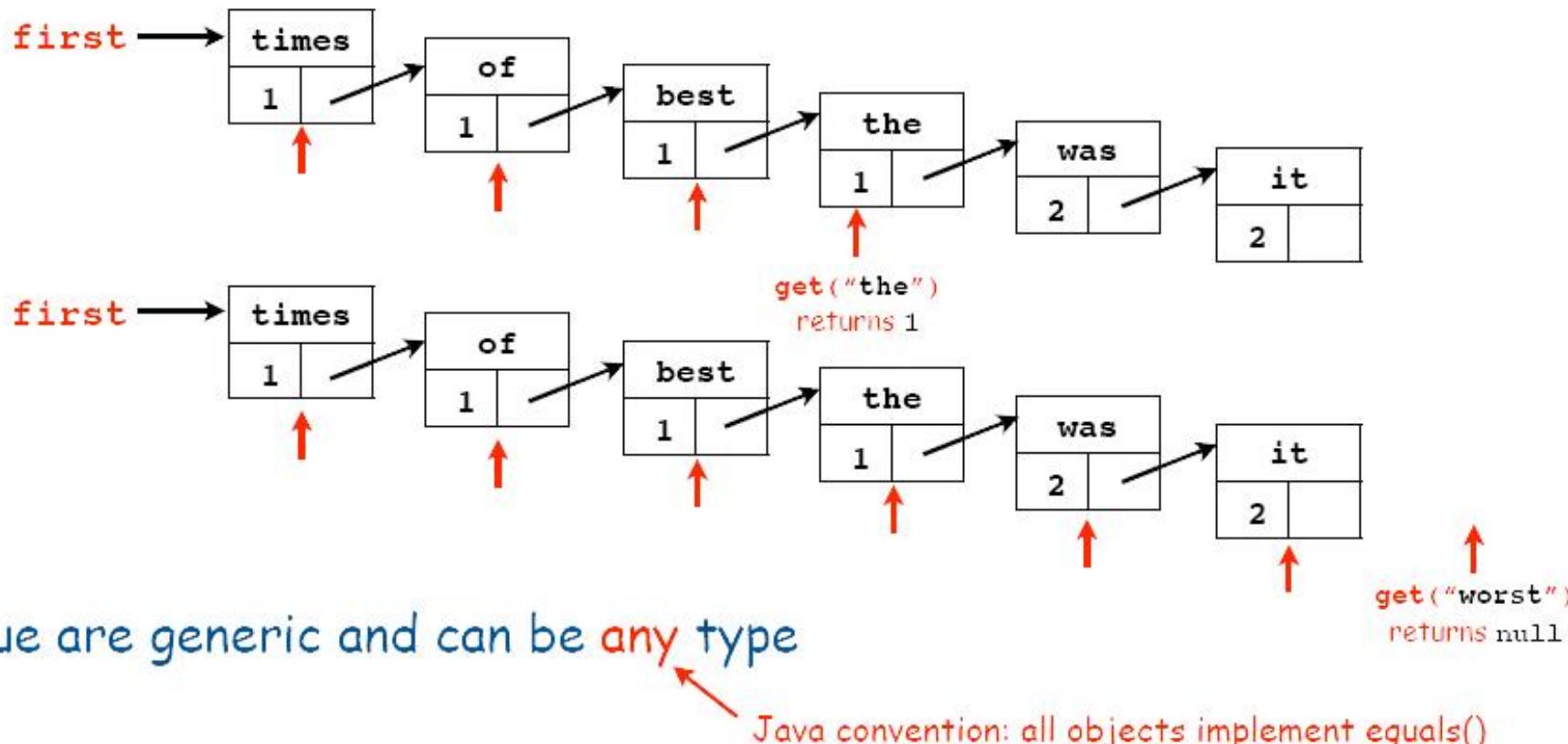
    private class Node            ← inner class
    {
        Key key;
        Value val;
        Node next;
        Node(Key key, Value val, Node next)
        {
            this.key = key;
            this.val = val;
            this.next = next;
        }
    }

    public void put(Key key, Value val)
    // see next slides

    public Val get(Key key)
    // see next slides
}
```

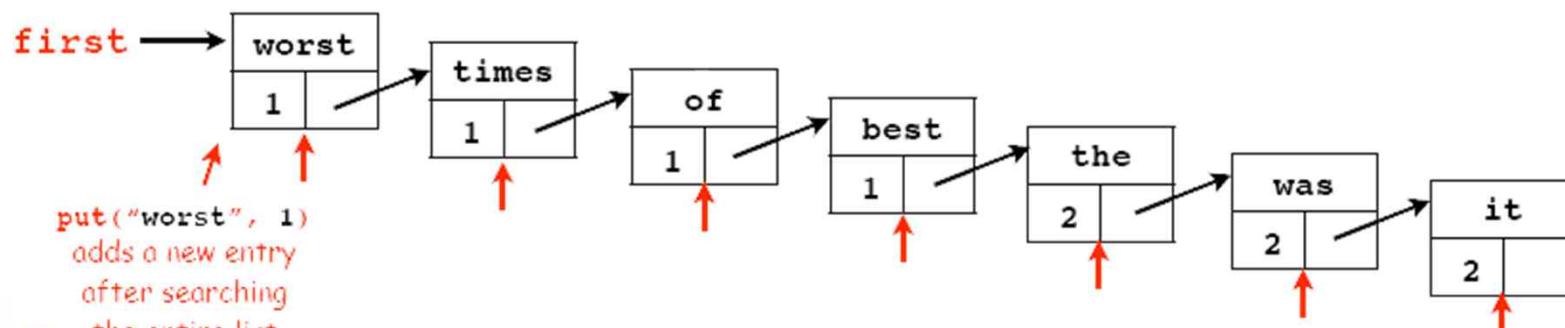
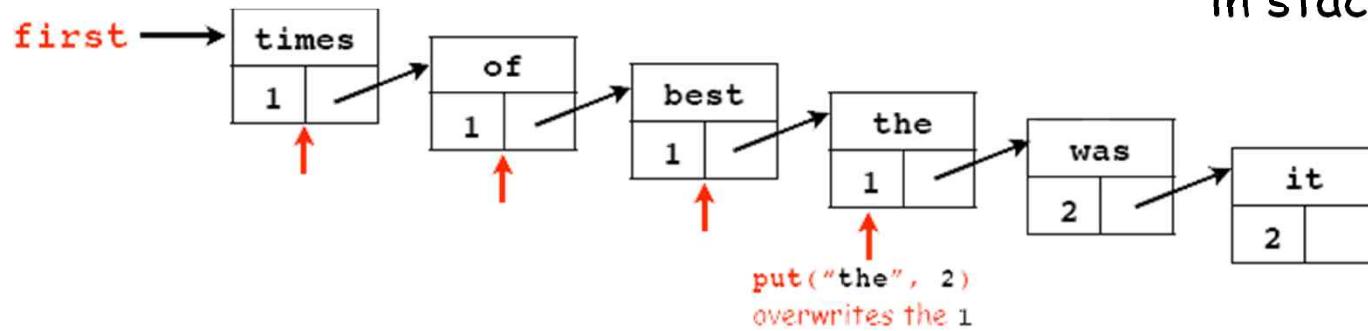
Linked list ST implementation (skeleton)

```
public Value get(Key key)
{
    for (Node x = first; x != null; x = x.next)
        if (key.equals(x.key))
            return vals[i];
    return null;
}
```



Linked list ST implementation (skeleton)

```
public void put(Key key, Value val)
{
    for (Node x = first; x != null; x = x.next)
        if (key.equals(x.key))
            { x.value = value; return; }
    first = new Node(key, value, first);
}
```



□ Associative array abstraction

- ◆ must search for key and, if it is there, **overwrite** with new value
- ◆ otherwise, **add** new key, value at the beginning (as in stack)

6. Symbol Table

- API
- basic implementations
- iterators
- Comparable keys
- challenges

Iterators

- Symbol tables should be Iterable

```
public interface Iterable<Item>
{
    Iterator<Item> iterator();
}
```

- Q. What is Iterable?

java.util.Iterator

- A. Implements iterator()

```
public interface Iterator<Item>
{
    boolean hasNext();
    Item next();
    void remove(); ← optional in Java
}
```

use at your own risk

- Q. What is an Iterator?

- A. Implements hasNext() and next().

- Q. Why should symbol tables be iterable?

- A. Java language supports elegant client code for iterators

"foreach" statement

```
for (String s: st)
    StdOut.println(st.get(s));
```

equivalent code

```
Iterator<String> i = st.iterator();
while (i.hasNext())
{
    String s = i.next();
    StdOut.println(st.get(s));
}
```

Iterable ST client: count frequencies of occurrence of input strings

- Standard input: A file (of strings)
- Standard output: All the distinct strings in the file with frequency

```
% more tiny.txt  
it was the best of times  
it was the worst of times  
it was the age of wisdom  
it was the age of foolishness
```

```
% java FrequencyCount < tiny.txt  
2 age  
1 best  
1 foolishness  
4 it  
4 of  
4 the  
2 times  
4 was  
1 wisdom  
1 worst
```

tiny example
24 words
10 distinct

```
% more tale.txt  
it was the best of times  
it was the worst of times  
it was the age of wisdom  
it was the age of foolishness  
it was the epoch of belief  
it was the epoch of incredulity  
it was the season of light  
it was the season of darkness  
it was the spring of hope  
it was the winter of despair  
we had everything before us  
we had nothing before us  
...  
% java FrequencyCount < tale.txt  
2941 a  
1 aback  
1 abandon  
10 abandoned  
1 abandoning  
1 abandonment  
1 abashed  
1 abate  
1 abated  
5 abbaye  
2 abed  
1 abhorrence  
1 abided  
1 abiding  
1 abilities  
2 ability  
1 abject  
1 ablaze  
17 able  
1 abnegating
```

real example
137177 words
9888 distinct



Iterable ST client: count frequencies of occurrence of input strings

```
public class FrequencyCount
{
    public static void main(String[] args)
    {
        ST<String, Integer> st;
        st = new ST<String, Integer>();

        while (!StdIn.isEmpty())
        {
            String key = StdIn.readString();           ← read a string

            if (!st.contains(key))
                st.put(key, 1);                      ← insert
            else
                st.put(key, st.get(key) + 1);         ← increment

        }

        for (String s: st)
            StdOut.println(st.get(s) + " " + s);   ← print all strings
    }
}
```

- Note: Only slightly more work required to build an **index** of all of the places where each key occurs in the text.

Iterators for array, linked list ST implementations

```
import java.util.Iterator;
public class UnorderedST<Key, Value>
    implements Iterable<Key>

{
    ...

    public Iterator<Key> iterator()
    { return new ArrayIterator(); }

    private class ArrayIterator
        implements Iterator<Key>
    {
        private int i = 0;

        public boolean hasNext()
        { return i < N; }

        public void remove() { }

        public Key next()
        { return keys[i++]; }
    }
}
```

```
import java.util.Iterator;
public class LinkedListST<Key, Value>
    implements Iterable<Key>

{
    ...

    public Iterator<Key> iterator()
    { return new ListIterator(); }

    private class ListIterator
        implements Iterator<Key>
    {
        private Node current = first;

        public boolean hasNext()
        { return current != null; }

        public void remove() { }

        public Key next()
        {
            Key key = current.key;
            current = current.next;
            return key;
        }
    }
}
```

Iterable ST client: A problem?

Use UnorderedST in FrequencyCount

```
% more tiny.txt  
it was the best of times  
it was the worst of times  
it was the age of wisdom  
it was the age of foolishness
```

```
% java FrequencyCount < tiny.txt  
4 it  
4 was  
4 the  
1 best  
4 of  
2 times  
1 worst  
2 age  
1 wisdom  
1 foolishness
```

Array

Use UnorderedST in FrequencyCount

```
% more tiny.txt  
it was the best of times  
it was the worst of times  
it was the age of wisdom  
it was the age of foolishness
```

```
% java FrequencyCount < tiny.txt  
1 foolishness  
1 wisdom  
2 age  
1 worst  
2 times  
4 of  
1 best  
4 the  
4 was  
4 it
```

Linked List

- Clients who use Comparable keys might expect ordered iteration
 - ◆ not a requirement for some clients
 - ◆ not a problem if postprocessing, e.g. with sort or grep
 - ◆ not in API



6. Symbol Table

- API
- basic implementations
- iterators
- Comparable keys
- challenges

Ordered array ST implementation

- Assume that keys are Comparable
- Maintain parallel arrays with keys and values that are sorted by key.
- Instance variables
 - ◆ `keys[i]` holds the *i*th smallest key
 - ◆ `vals[i]` holds the value associated with the *i*th smallest key
 - ◆ integer `N` holds the number of entries.

$N = 6$
↓

- Note: no duplicate keys

	0	1	2	3	4	5
keys[]	best	it	of	the	times	was
vals[]	1	2	1	1	1	2

- Need to use standard array-doubling technique

- Two reasons to consider using ordered arrays

- ◆ provides ordered iteration (for free)
- ◆ can use binary search to significantly speed up search



Ordered array ST implementation (skeleton)

```
public class OrderedST
    <Key extends Comparable<Key>, Value>
    implements Iterable<Key>           ← standard array iterator code omitted
{
    private Value[] vals;
    private Key[] keys;
    private int N = 0;
    public OrderedST(int maxN)
    {
        keys = (Key[]) new Object[maxN];
        vals = (Value[]) new Object[maxN];
    }

    public boolean isEmpty()
    {   return N == 0; }

    public void put(Key key, Value val)
        // see next slides

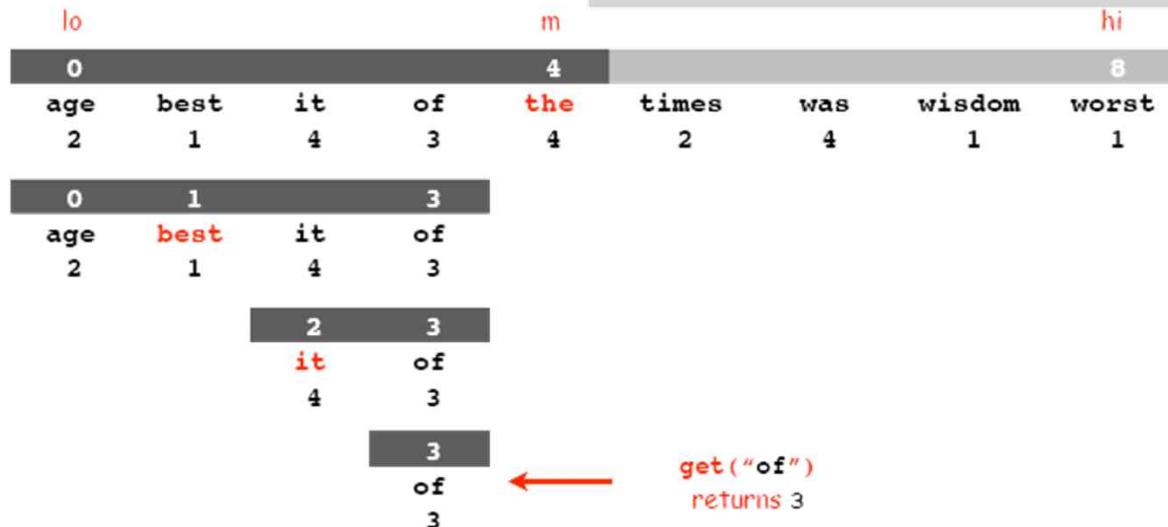
    public Val get(Key key)
        // see next slides
}
```

Ordered array ST implementation (skeleton)

- Keeping array in order enables **binary search algorithm**

```
public Value get(Key key)
{
    int i = bsearch(key);
    if (i == -1) return null;
    return vals[i];
}
```

```
private int bsearch(Key key)
{
    int lo = 0, hi = N-1;
    while (lo <= hi)
    {
        int m = lo + (hi - lo) / 2;
        int cmp = key.compareTo(keys[m]);
        if (cmp < 0) hi = m - 1;
        else if (cmp > 0) lo = m + 1;
        else return m;
    }
    return -1;
}
```



Binary search analysis: Comparison count

- Def. $T(N) \equiv$ number of comparisons to binary search in an ST of size N
 $= T(N/2) + 1$

Left or right half middle

Binary search recurrence $T(N) = T(N/2) + 1$

for $N > 1$, with $T(1) = 0$

- ◆ not quite right for odd N
- ◆ same recurrence holds for many algorithms
- ◆ same number of comparisons for any input of size N .

Solution of binary search recurrence $T(N) \sim \lg N$

- ◆ true for all N
- ◆ easy to prove when N is a power of 2.

can then use induction for general N

Binary search recurrence: Proof by telescoping

$$T(N) = T(N/2) + 1$$

for $N > 1$, with $T(1) = 0$

(assume that N is a power of 2)

Pf.

$$\begin{aligned} T(N) &= T(N/2) + 1 && \text{given} \\ &= T(N/4) + 1 + 1 && \text{Telescope (apply to first term)} \\ &= T(N/8) + 1 + 1 + 1 && \text{Telescope again} \\ &\dots \\ &= T(N/N) + 1 + 1 + \dots + 1 && \text{Stop telescoping, } T(1) = 0 \\ &= \lg N \end{aligned}$$

$$T(N) = \lg N$$

Ordered array ST implementation (insert)

- Binary search is little help for put(): still need to move larger keys

```
public Val put(Key key, Value val)
{
    int i = bsearch(key);
    if (i != -1)
    {   vals[i] = val; return; }

    for ( i = N; i > 0; i-- )
    {
        if key.compareTo(keys[i-1] < 0) break;
        keys[i] = keys[i-1];
        vals[i] = vals[i-1];
    }
    vals[i] = val;
    keys[i] = key;
    N++;
}
```

← overwrite with new value
if key in table

← move larger keys to make room
if key not in table



Ordered array ST implementation: an important special case

- Test whether key is equal to or greater than largest key

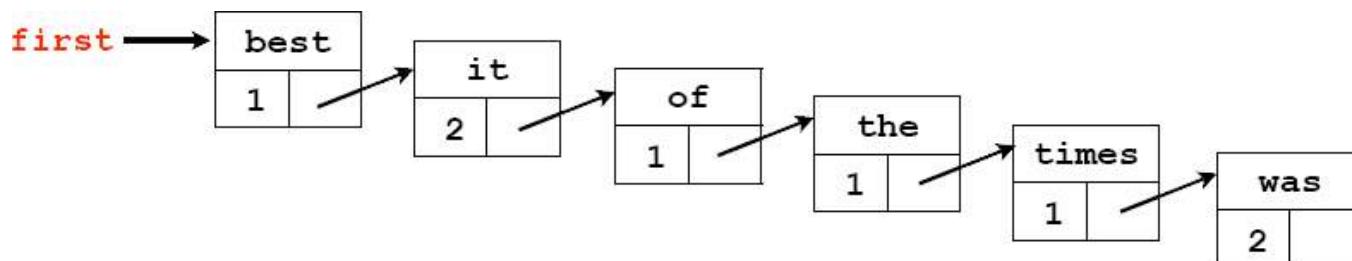
```
public Val put(Key key, Value val)
{
    if (key.compareTo(keys[N-1]) == 0)
    {   vals[N-1] = val; return; }

    if (key.compareTo(keys[N-1]) > 0)
    {
        vals[N] = val;
        keys[N] = key;
        N++;
        return;
    }
}
```

- If either test succeeds, **constant-time** insert!
- Method of choice for some clients:
 - ◆ sort database by key
 - ◆ insert N key-value pairs in order by key
 - ◆ support searches that never use more than $\lg N$ compares
 - ◆ support occasional (expensive) inserts

Ordered linked-list ST implementation

- Binary search **depends on array indexing** for efficiency.
- Jump to the middle of a linked list?
- Advantages of keeping linked list in order for Comparable keys:
 - ◆ support ordered iterator (for free)
 - ◆ cuts search/insert **time in half** (on average) for random search/insert
- [code omitted]



6. Symbol Table

- API
- basic implementations
- iterators
- Comparable keys
- challenges

Searching challenge 1A:

- Problem: maintain symbol table of song names for an iPod
- Assumption A: **hundreds** of songs

- Which searching method to use?
 - 1) unordered array
 - 2) ordered linked list
 - 3) ordered array with binary search
 - 4) need better method, all too slow
 - 5) doesn't matter much, all fast enough

Searching challenge 1B:

- Problem: maintain symbol table of song names for an iPod
- Assumption B: **thousands** of songs

- Which searching method to use?
 - 1) unordered array
 - 2) ordered linked list
 - 3) ordered array with binary search
 - 4) need better method, all too slow
 - 5) doesn't matter much, all fast enough

Searching challenge 2A:

- Problem: IP lookups in a web monitoring device
- Assumption A: billions of lookups, millions of distinct addresses

- Which searching method to use?
 - 1) unordered array
 - 2) ordered linked list
 - 3) ordered array with binary search
 - 4) need better method, all too slow
 - 5) doesn't matter much, all fast enough

Searching challenge 2B:

- Problem: IP lookups in a web monitoring device
- Assumption B: billions of lookups, thousands of distinct addresses

- Which searching method to use?
 - 1) unordered array
 - 2) ordered linked list
 - 3) ordered array with binary search
 - 4) need better method, all too slow
 - 5) doesn't matter much, all fast enough

Searching challenge 3:

- Problem: Frequency counts in "Tale of Two Cities"
- Assumptions: book has 135,000+ words
about 10,000 distinct words
- Which searching method to use?
 - 1) unordered array
 - 2) ordered linked list
 - 3) ordered array with binary search
 - 4) need better method, all too slow
 - 5) doesn't matter much, all fast enough

Searching challenge 4:

- Problem: Spell checking for a book
- Assumptions: dictionary has 25,000 words
book has 100,000+ words

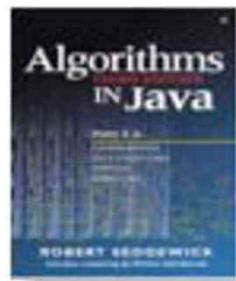
- Which searching method to use?
 - 1) unordered array
 - 2) ordered linked list
 - 3) ordered array with binary search
 - 4) need better method, all too slow
 - 5) doesn't matter much, all fast enough

Searching challenge 5:

- ❑ Problem: Sparse matrix-vector multiplication
 - ❑ Assumptions: matrix dimension is billions by billions
average number of nonzero entries/row is ~10

- Which searching method to use?
 - 1) unordered array
 - 2) ordered linked list
 - 3) ordered array with binary search
 - 4) need better method, all too slow
 - 5) doesn't matter much, all fast enough

Summary and Roadmap



- ◆ basic algorithmics
- ◆ no generics
- ◆ more code
- ◆ more analysis
- ◆ equal keys in ST (not associative arrays)



- ◆ Iterators
- ◆ ST as associative array (all keys distinct)
- ◆ BST implementations
- ◆ Applications



- ◆ distinguish algs by operations on keys
- ◆ ST as associative array (all keys distinct)
- ◆ important special case for binary search
- ◆ challenges

Elementary implementations: summary

implementation	worst case		average case		ordered iteration?	operations on keys
	search	insert	search	insert		
unordered array	N	N	N/2	N/2	no	<code>equals()</code>
ordered array	$\lg N$	N	$\lg N$	N/2	yes	<code>compareTo()</code>
unordered list	N	N	N/2	N	no	<code>equals()</code>
ordered list	N	N	N/2	N/2	yes	<code>compareTo()</code>

□ Next challenge.

- ◆ Efficient implementations of **search** and **insert** and **ordered iteration** for arbitrary sequences of operations.



(ordered array meets challenge if keys arrive approximately in order)

Data Structures & Algorithms

7. Binary Search Trees

- basic implementations
- randomized BSTs
- deletion in BSTs



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

Elementary implementations: summary

implementation	worst case		average case		ordered iteration?	operations on keys
	search	insert	search	insert		
unordered array	N	N	N/2	N/2	no	<code>equals()</code>
ordered array	$\lg N$	N	$\lg N$	N/2	yes	<code>compareTo()</code>
unordered list	N	N	N/2	N	no	<code>equals()</code>
ordered list	N	N	N/2	N/2	yes	<code>compareTo()</code>

Challenge:

Efficient implementations of `get()` and `put()` and ordered iteration.

7. Binary Search Trees

- basic implementations
- randomized BSTs
- deletion in BSTs

BST representation

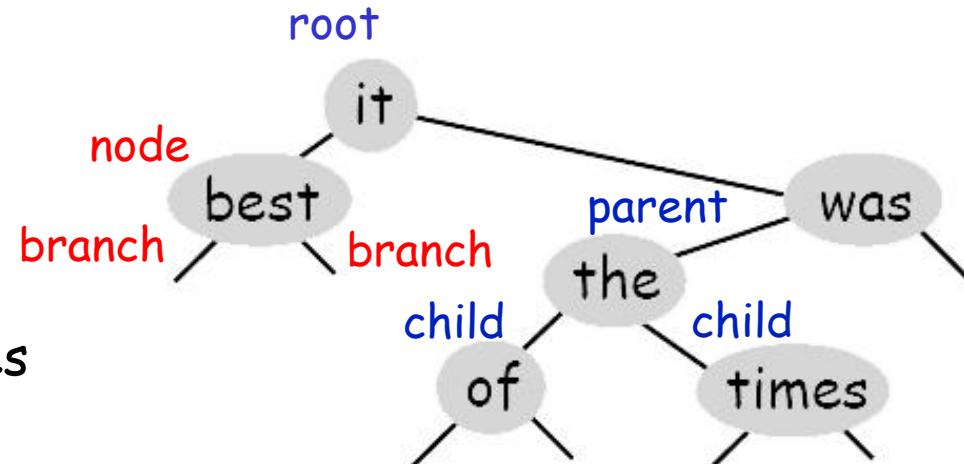


- Def. A BINARY SEARCH TREE is a **binary tree** in **symmetric order**.

- A **binary tree** is either:

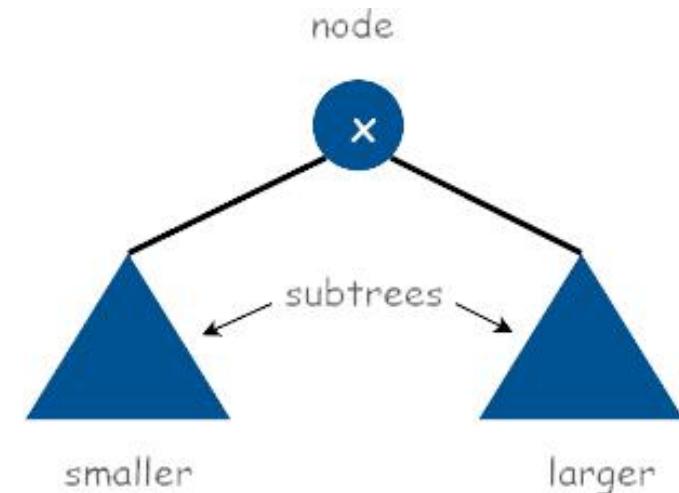
- ◆ Empty
- ◆ a key-value pair and two binary trees
[neither of which contain that key]

equal keys ruled out to facilitate
associative array implementations



- **Symmetric order** means that:

- ◆ every node has a **key**
- ◆ every node's key is
 - larger than all keys in its **left** subtree
 - smaller than all keys in its **right** subtree



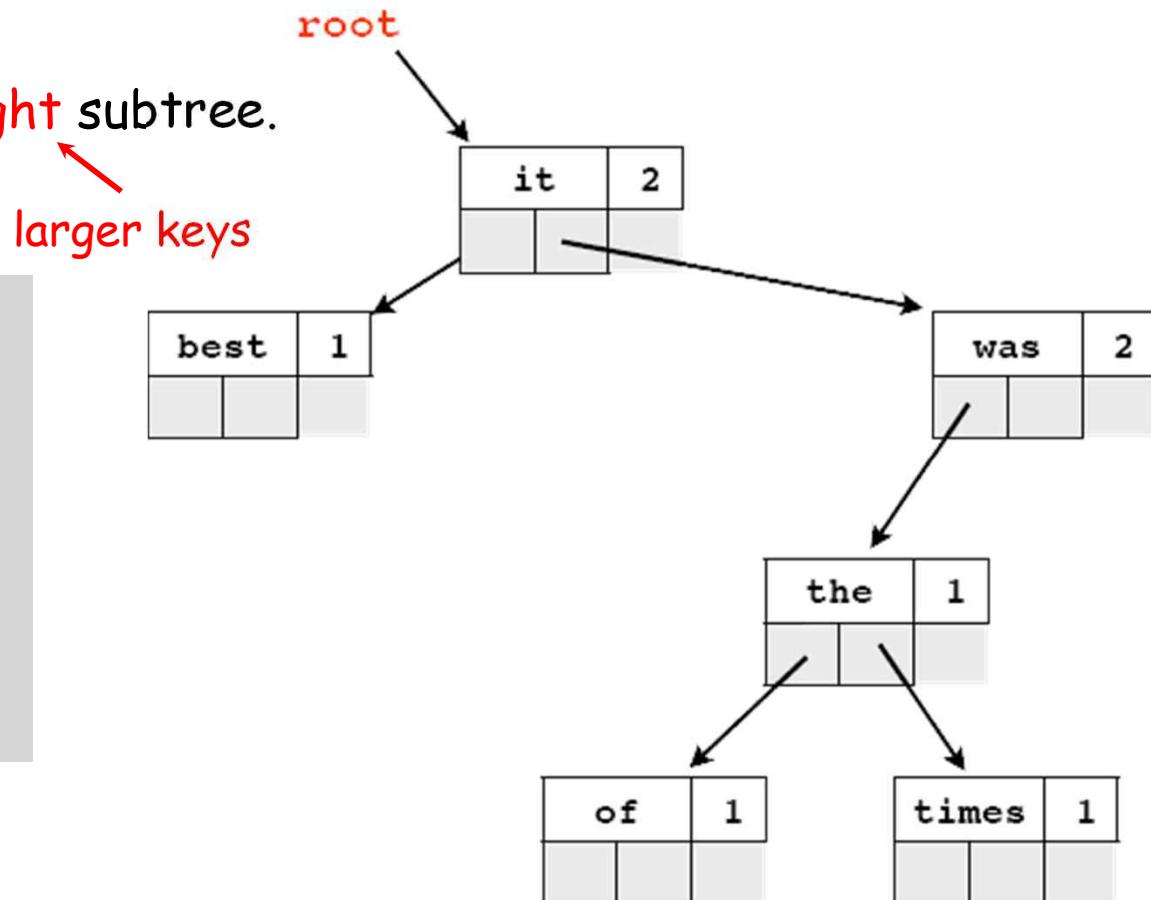
BST representation

- A BST is a reference to a Node.
- A Node is comprised of four fields:

- ◆ A **key** and a **value**.
- ◆ A reference to the **left** and **right** subtree.

```
private class Node
{
    Key key;
    Value val;
    Node left, right;
}
```

Key and Value are **generic types**;
Key is **Comparable**



BST implementation (skeleton)

```
public class BST<Key extends Comparable<Key>, Value>
    implements Iterable<Key>
{
    private Node root;                                ← instance variable

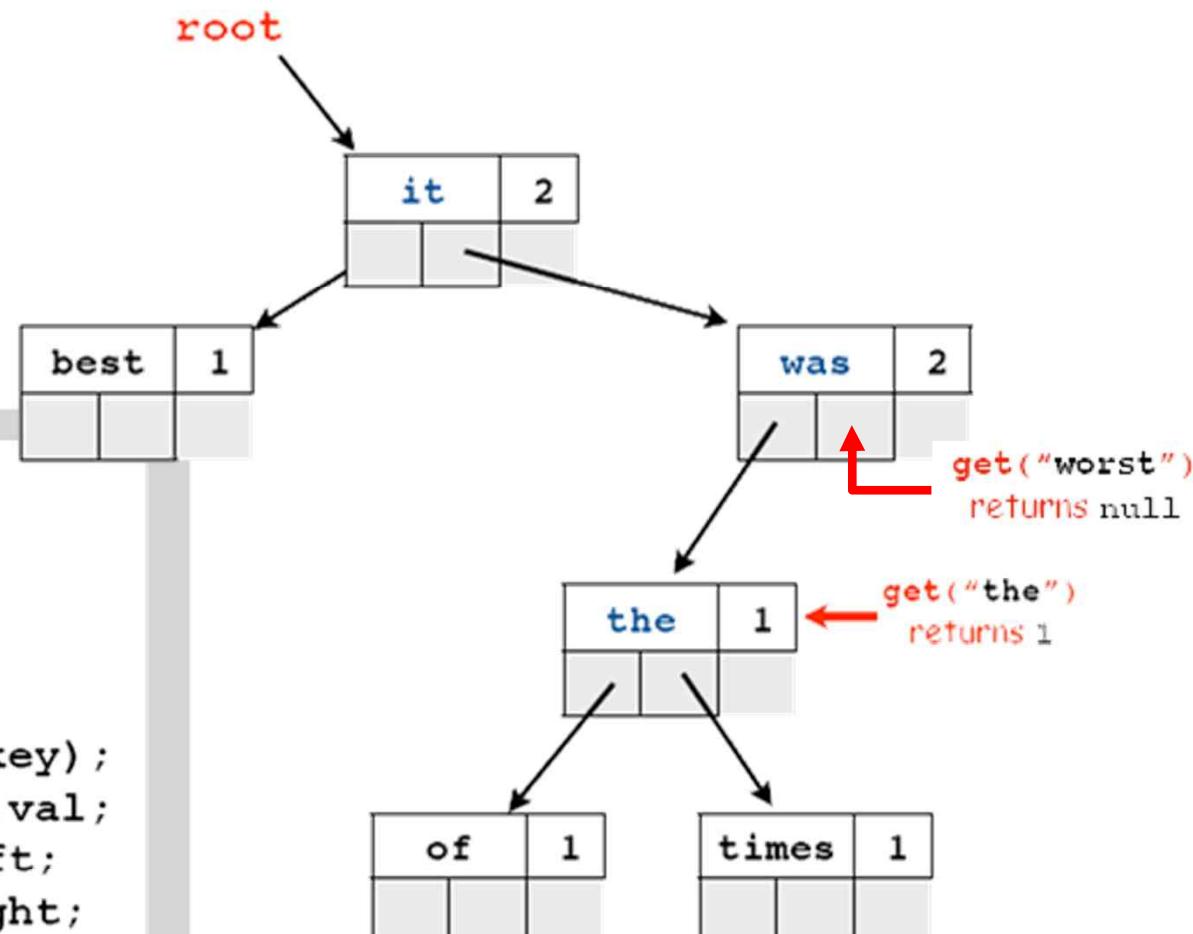
    private class Node                               ← inner class
    {
        Key key;
        Value val;
        Node left, right;
        Node(Key key, Value val)
        {
            this.key = key;
            this.val = val;
        }
    }

    public void put(Key key, Value val)
    // see next slides

    public Value get(Key key)
    // see next slides
}
```

BST implementation (search)

```
public Value get(Key key)
{
    Node x = root;
    while (x != null)
    {
        int cmp = key.compareTo(x.key);
        if (cmp == 0)      return x.val;
        else if (cmp < 0) x = x.left;
        else if (cmp > 0) x = x.right;
    }
    return null;
}
```

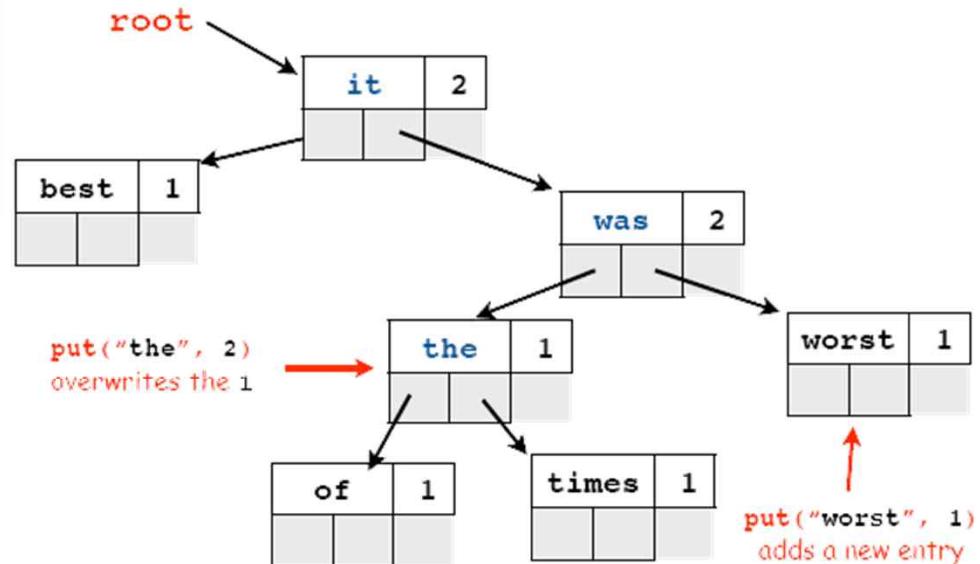


BST implementation (insert)

```
public void put(Key key, Value val)
{   root = put(root, key, val); }
```

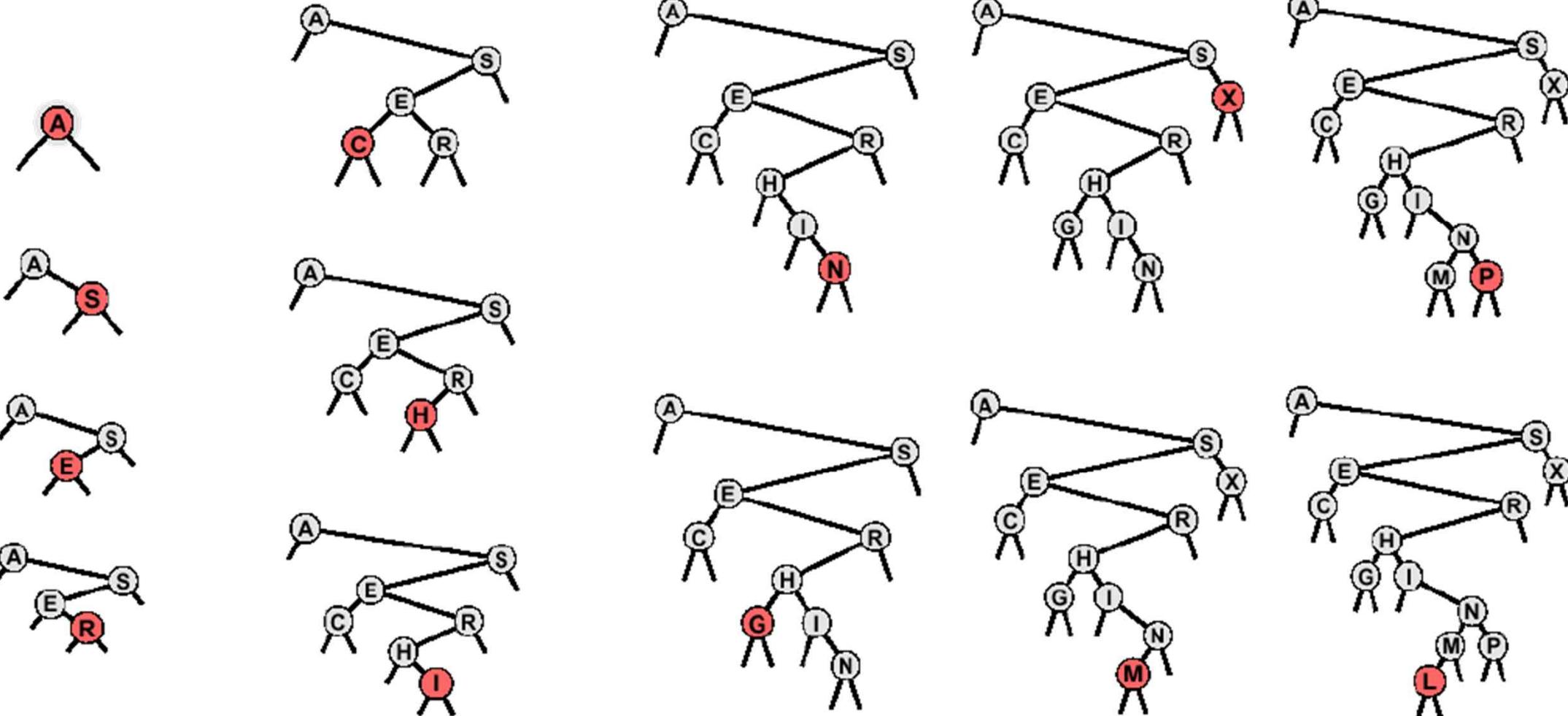
Caution: tricky recursive code.
Read carefully!

```
private Node put(Node x, Key key, Value val)
{
    if (x == null) return new Node(key, val);
    int cmp = key.compareTo(x.key);
    if (cmp == 0)      x.val = val;
    else if (cmp < 0) x.left = put(x.left, key, val);
    else if (cmp > 0) x.right = put(x.right, key, val);
    return x;
}
```



BST: Construction

- Insert the following keys into BST. A S E R C H I N G X M P L

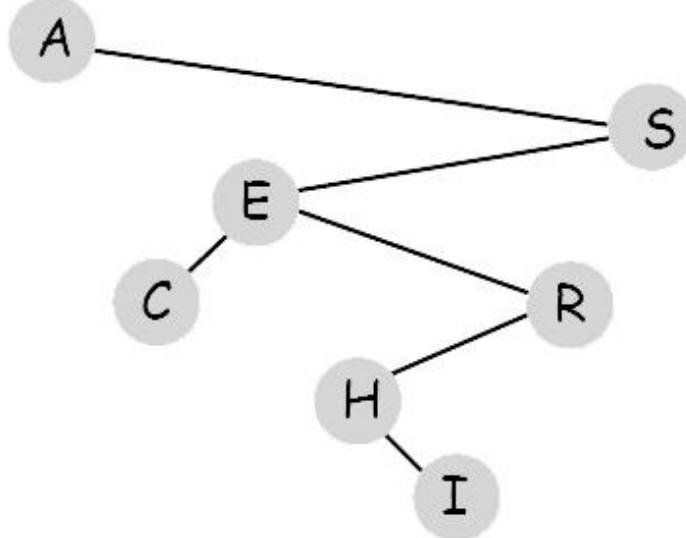


BST: Construction

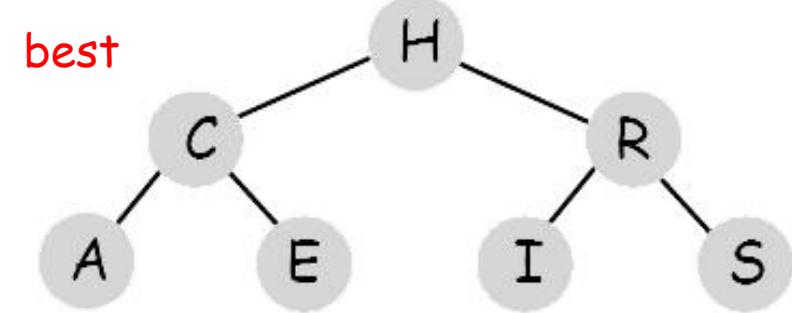
- Tree shape.

- ◆ Many BSTs correspond to **same** input data.
- ◆ Cost of search/insert is proportional to **depth** of node.

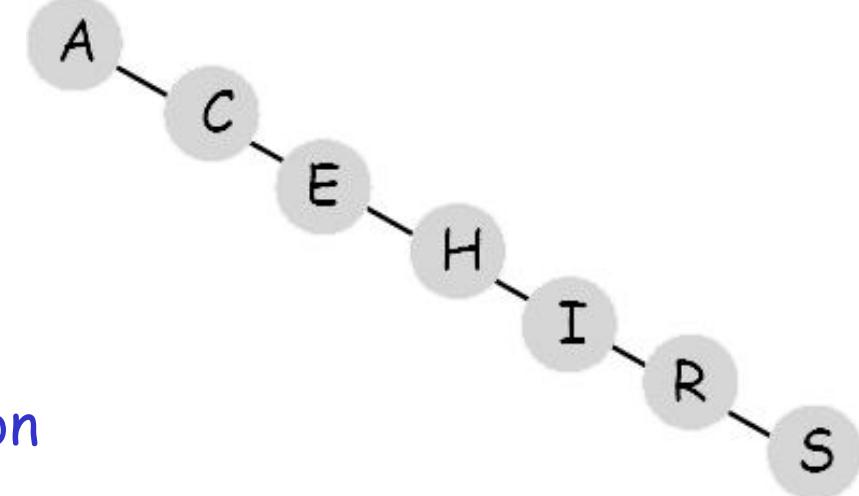
typical



best



worst



- Tree shape depends on **order of insertion**

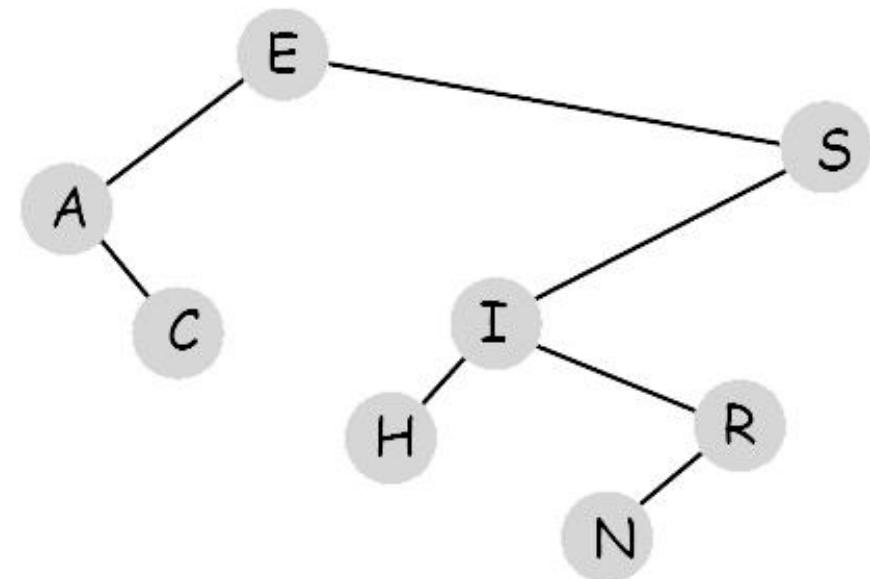
BST implementation: iterator?

```
public Iterator<Key> iterator()
{ return new BSTIterator(); }

private class BSTIterator
    implements Iterator<Key>
{
    BSTIterator()
    {
        ...
    }

    public boolean hasNext()
    {
        ...
    }

    public Key next()
    {
        ...
    }
}
```



BST implementation: iterator?

- Approach: mimic recursive **inorder** traversal

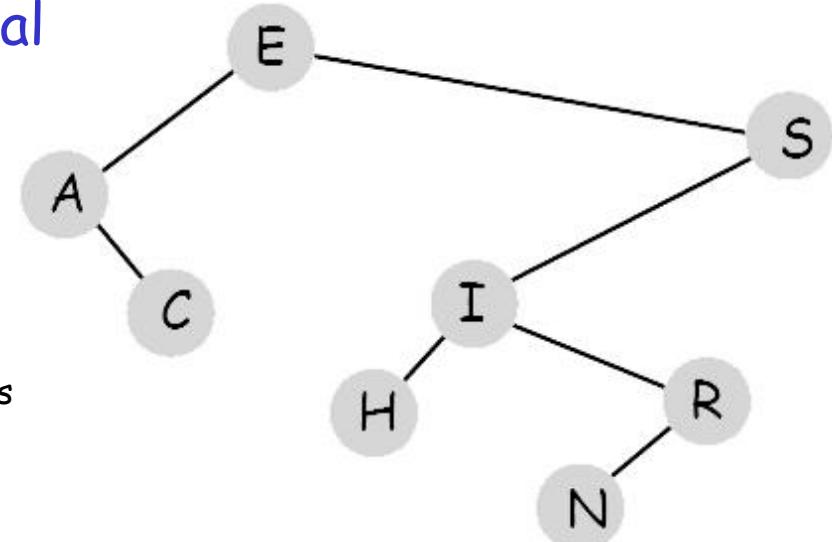
```
public void visit(Node x)
{
    if (x == null) return;
    visit(x.left)
    StdOut.println(x.key);
    visit(x.right);
}
```

```
visit(E)
visit(A)
    print A
visit(C)
    print C
print E
visit(S)
visit(I)
visit(H)
    print H
print I
visit(R)
visit(N)
    print N
    print R
print S
```

A
C
E
H
I
N
R
S

E	
A	E
E	
C	E
E	
S	
I	S
H	I
I	S
S	
R	S
N	R
R	S
S	

Stack contents



- To process a node

- ◆ follow **left** links until empty (pushing onto stack)
- ◆ pop and process
- ◆ process node at **right** link

BST implementation: iterator

```
public Iterator<Key> iterator()
{ return new BSTIterator(); }

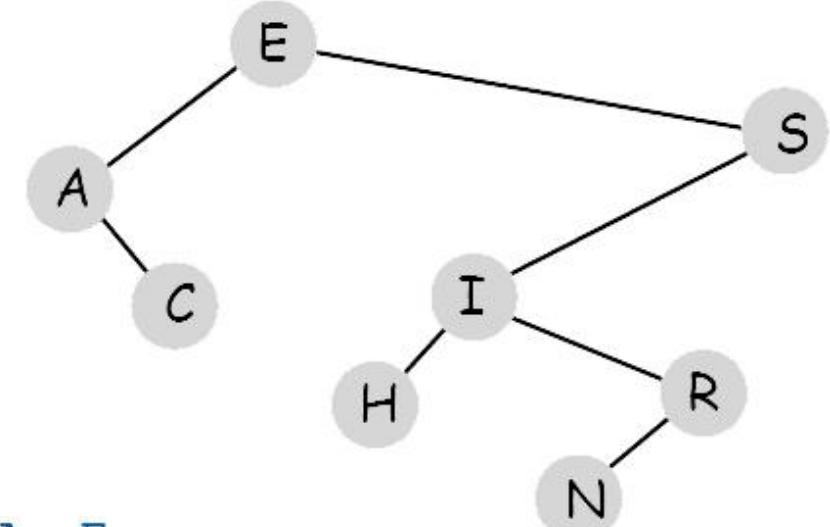
private class BSTIterator
    implements Iterator<Key>
{
    private Stack<Node>
        stack = new Stack<Node>();

    private void pushLeft(Node x)
    {
        while (x != null)
        { stack.push(x); x = x.left; }
    }

    BSTIterator()
    { pushLeft(root); }

    public boolean hasNext()
    { return !stack.isEmpty(); }

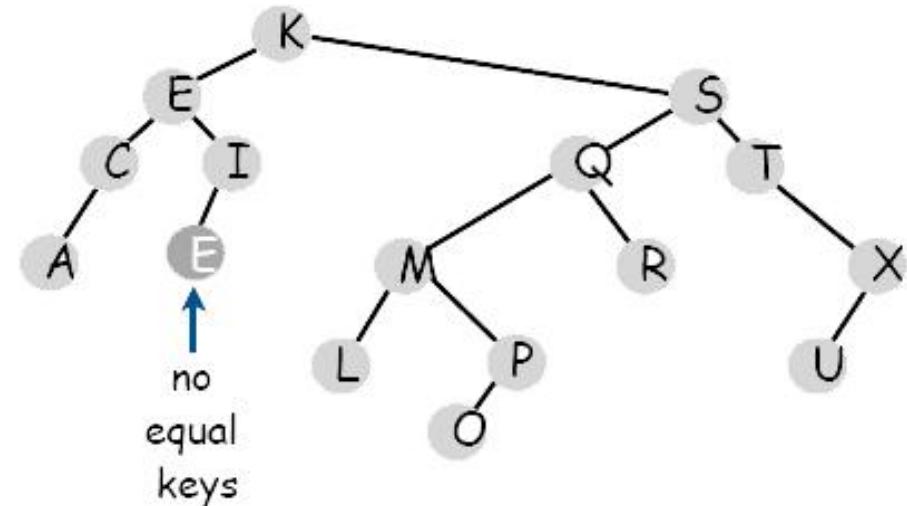
    public Key next()
    {
        Node x = stack.pop();
        pushLeft(x.right);
        return x.key;
    }
}
```



A	E
A	C E
C	E
E	H I S
H	I S
I	N R S
N	R S
R	S
S	

1-1 Correspondence between BSTs and Quicksort partitioning

Q U I C K S O R T E X A M P L E
E R A T E S L P U I M Q C X O K
E C A I E K L P U T M Q R X O S
A C E I E K L P U T M Q R X O S
A C E I E K L P U T M Q R X O S
A C E E I K L P U T M Q R X O S
A C E E I K L P U T M Q R X O S
A C E E I K L P O R M Q S X U T
A C E E I K L P O M Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T
A C E E I K L M O P Q R S X U T



BSTs: Analysis

Theorem. If keys are inserted in **random order**, the expected number of comparisons for a search/insert is about $2 \ln N$.

$$\approx 1.38 \lg N, \text{variance} = O(1)$$

- Proof: 1-1 correspondence with quicksort partitioning

- Theorem. If keys are inserted in **random order**, height of tree is proportional to $\lg N$, except with exponentially small probability.

$$\text{mean } \approx 6.22 \lg N, \text{variance} = O(1)$$

- But... Worst-case for search/insert/height is N .

e.g., keys inserted in ascending order

Searching Challenge 3 (Revisited):

- Problem: Frequency counts in "Tale of Two Cities"
 - Assumptions: book has 135,000+ words
about 10,000 distinct words
 - Which searching method to use?
 - 1) unordered array
 - 2) unordered linked list
 - 3) ordered array with binary search
 - 4) need better method, all too slow
 - 5) doesn't matter much, all fast enough
 - 6) BSTs
- insertion cost < $10000 * 1.38 * \lg 10000 < .2 \text{ million}$
lookup cost < $135000 * 1.38 * \lg 10000 < 2.5 \text{ million}$
- 

Elementary Implementations: Summary

implementation	guarantee		average case		ordered iteration?	operations on keys
	search	insert	search	insert		
unordered array	N	N	N/2	N/2	no	<code>equals()</code>
ordered array	$\lg N$	N	$\lg N$	N/2	yes	<code>compareTo()</code>
unordered list	N	N	N/2	N	no	<code>equals()</code>
ordered list	N	N	N/2	N/2	yes	<code>compareTo()</code>
BST	N	N	$1.38 \lg N$	$1.38 \lg N$	yes	<code>compareTo()</code>

Next challenge:

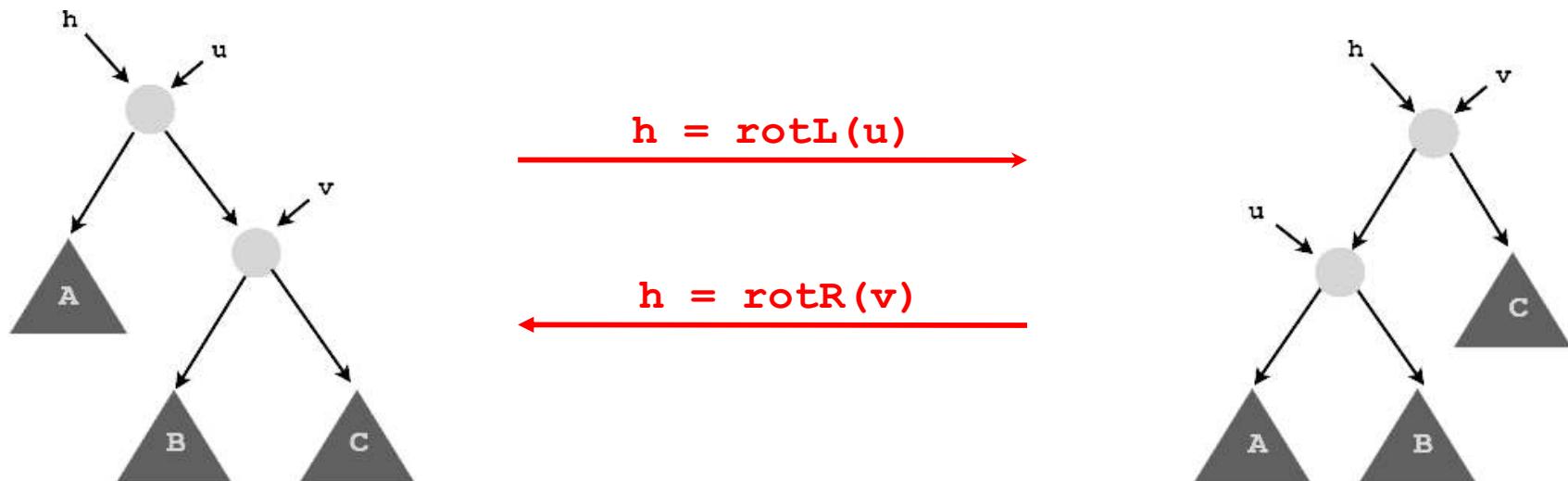
Guaranteed efficiency for `get()` and `put()` and ordered iteration.

7. Binary Search Trees

- basic implementations
- randomized BSTs
- deletion in BSTs

Rotation in BSTs

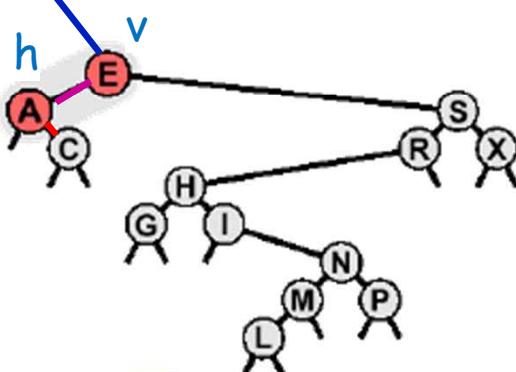
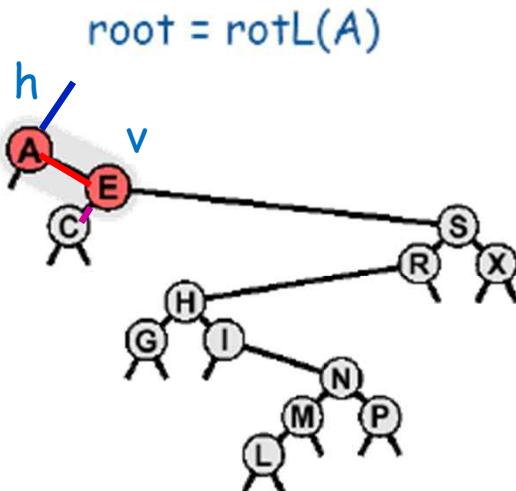
- Two fundamental operations to rearrange nodes in a tree.
 - ◆ maintain symmetric order.
 - ◆ local transformations (change just 3 pointers).
 - ◆ basis for advanced BST algorithms
- Strategy: use rotations on insert to adjust tree shape to be more balanced



Key point: no change in search code (!)

Rotation

- Fundamental operation to rearrange nodes in a tree.
 - ◆ easier done than said
 - ◆ raise some nodes, lowers some others

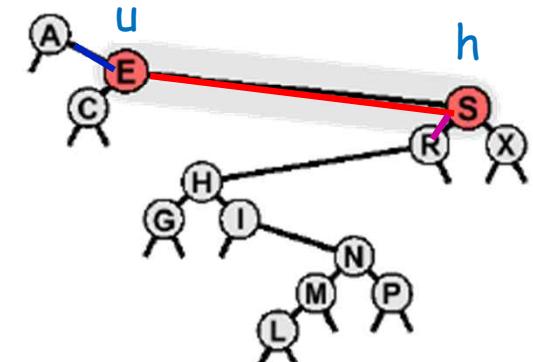
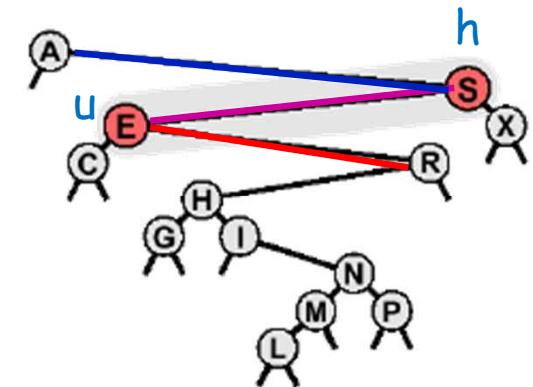


```
private Node rotL(Node h)
{
    Node v = h.r;
    h.r = v.l;
    v.l = h;
    return v;
}
```

.right
.left

```
private Node rotR(Node h)
{
    Node u = h.l;
    h.l = u.r;
    u.r = h;
    return u;
}
```

A.right = rotR(S)



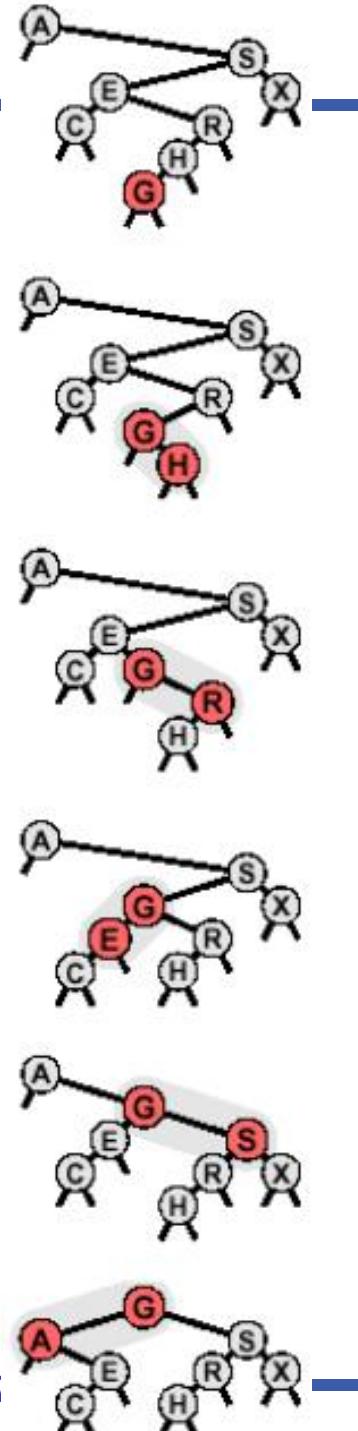
Recursive BST Root Insertion

- Root insertion: insert a node and make it the new root.

- Insert as in standard BST.
- Rotate inserted node to the root.
- Easy recursive implementation

Caution: **very** tricky recursive code.
Read very carefully!

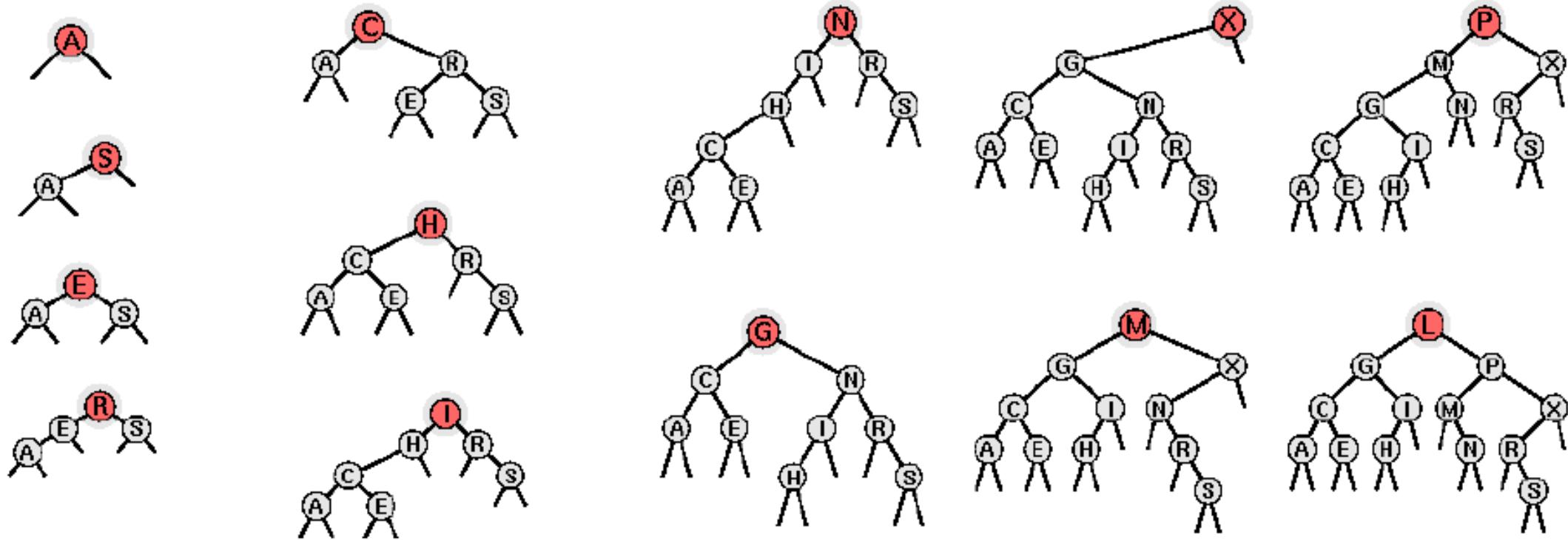
```
private Node putRoot(Node x, Key key, Val val)
{
    if (x == null) return new Node(key, val);
    int cmp = key.compareTo(x.key);
    if (cmp == 0) x.val = val;
    else if (cmp < 0)
    {
        x.left = putRoot(x.left, key, val); x = rotR(x);
    }
    else if (cmp > 0)
    {
        x.right = putRoot(x.right, key, val); x = rotL(x);
    }
    return x;
}
```



Algo:

Constructing a BST with Root Insertion

Ex. A SERCHING X M P L



Why bother?

- ◆ Recently inserted keys are near the top (better for some clients).
 - ◆ Basis for advanced algorithms.

Randomized BSTs (Roura, 1996)

- Intuition. If tree is random, height is logarithmic.
- Fact. Each node in a random tree is **equally likely** to be the root.
- Idea. Since new node should be the root with probability $1/(N+1)$, make it the root (via root insertion) **with probability $1/(N+1)$** .

```
private Node put(Node x, Key key, Value val)
{
    if (x == null) return new Node(key, val);

    int cmp = key.compareTo(x.key);
    if (cmp == 0) { x.val = val; return x; }

    if (StdRandom.bernoulli(1.0 / (x.N + 1.0)))
        return putRoot(x, key, val);

    if (cmp < 0) x.left = put(x.left, key, val);
    else if (cmp > 0) x.right = put(x.right, key, val);
    x.N++;
    return x;
}
```

need to maintain count of nodes in tree rooted at x



Randomized BSTs (Roura, 1996)

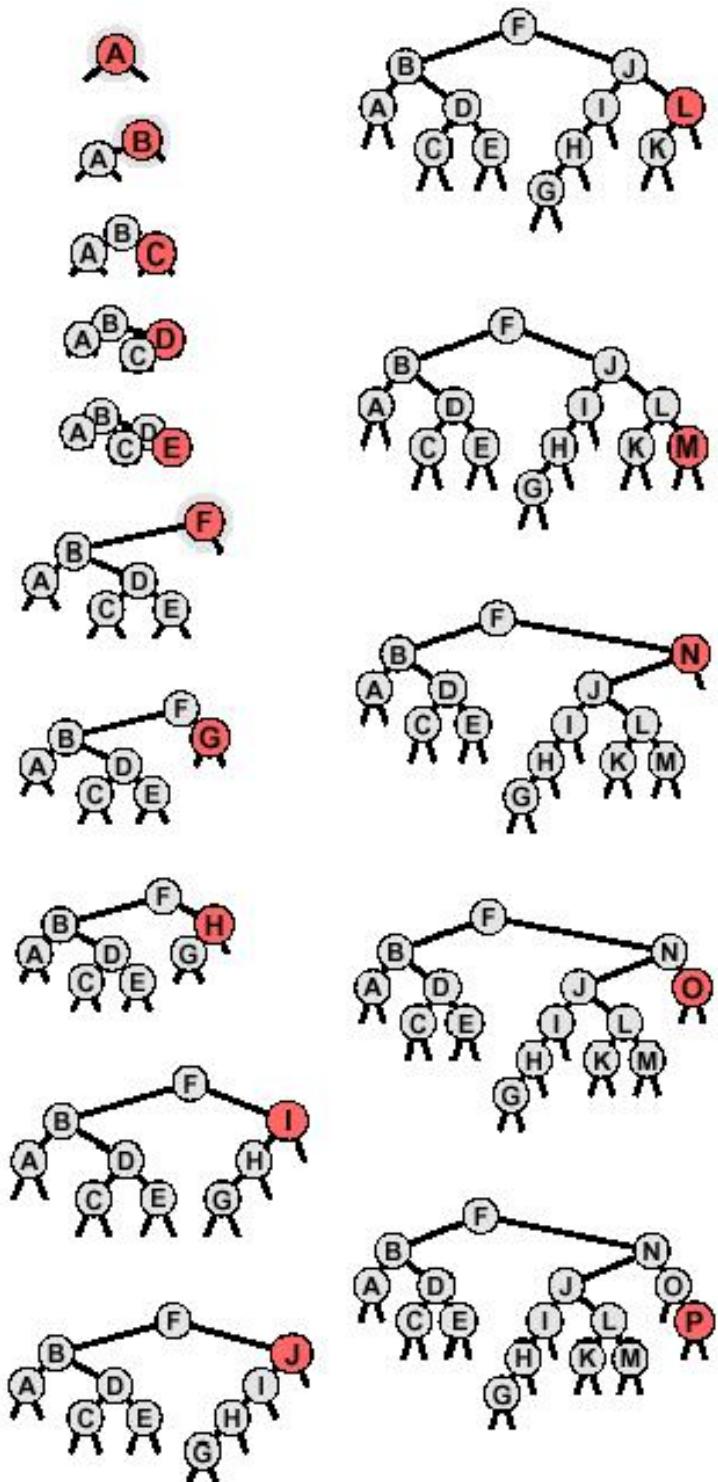
- Ex: Insert distinct keys in ascending order.

- Surprising fact:

Tree has same shape as if keys were inserted in **random** order.

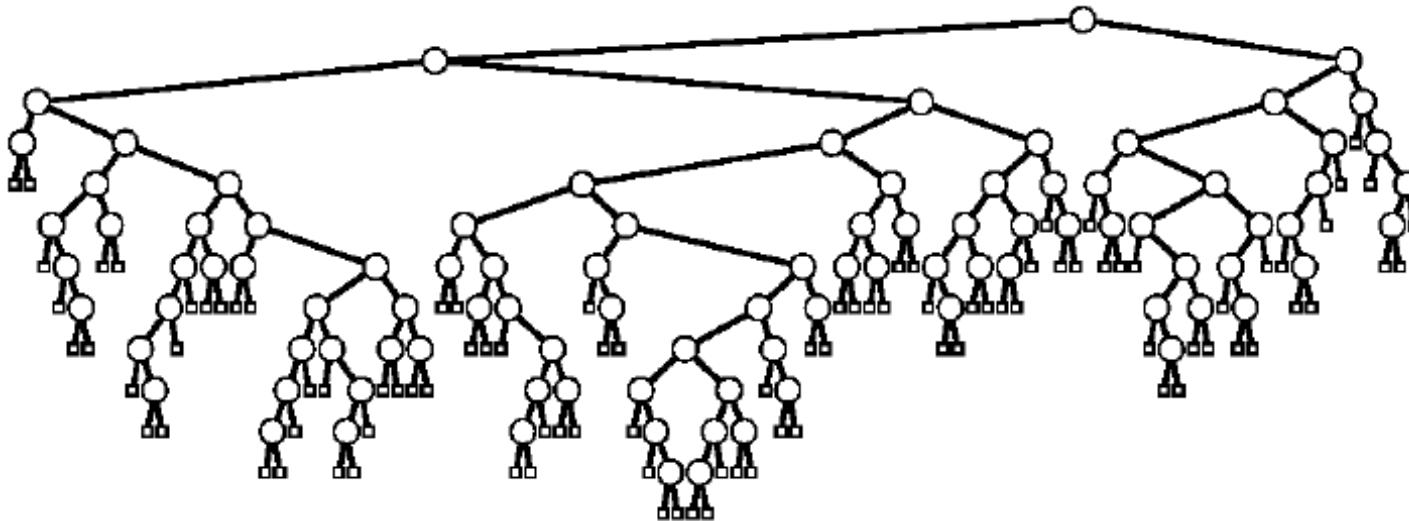
Random trees result from **any** insert order

Note: to maintain associative array abstraction
need to check whether key is in table and replace
value without rotations if that is the case.



Randomized BST

- **Property.** Randomized BSTs have the same distribution as BSTs under **random insertion order, no matter in what order** keys are inserted.



- ◆ Expected height is $\sim 6.22 \lg N$
- ◆ Average search cost is $\sim 1.38 \lg N$.
- ◆ Exponentially small chance of bad balance.

- **Implementation cost.** Need to maintain subtree size in each node.

Summary of Symbol-Table Implementations

implementation	guarantee		average case		ordered iteration?	operations on keys
	search	insert	search	insert		
unordered array	N	N	N/2	N/2	no	<code>equals()</code>
ordered array	$\lg N$	N	$\lg N$	N/2	yes	<code>compareTo()</code>
unordered list	N	N	N/2	N	no	<code>equals()</code>
ordered list	N	N	N/2	N/2	yes	<code>compareTo()</code>
BST	N	N	$1.38 \lg N$	$1.38 \lg N$	yes	<code>compareTo()</code>
randomized BST	$7 \lg N$	$7 \lg N$	$1.38 \lg N$	$1.38 \lg N$	yes	<code>compareTo()</code>

- Randomized BSTs provide the desired guarantee

probabilistic, with exponentially small chance of quadratic time

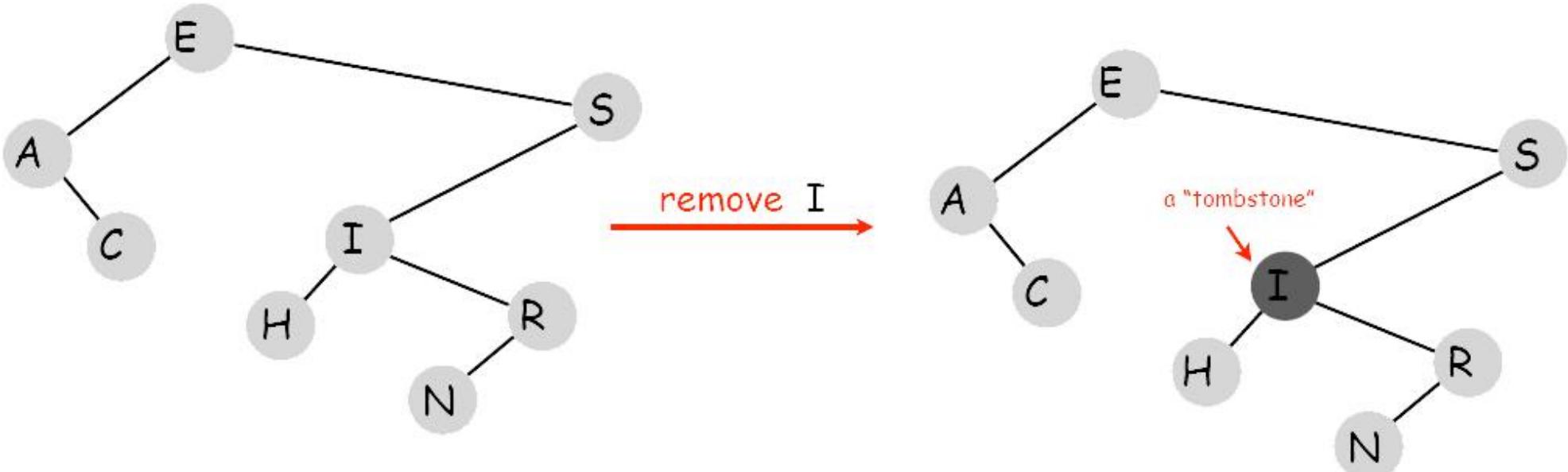
- Bonus (next): Randomized BSTs also support delete (!)

7. Binary Search Trees

- basic implementations
- randomized BSTs
- deletion in BSTs

BST delete: Lazy Approach

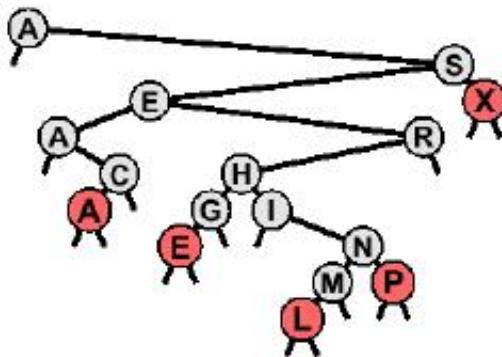
- To remove a node with a given key
 - ◆ set its value to null
 - ◆ leave key in tree to guide searches
[but do not consider it equal to any search key]



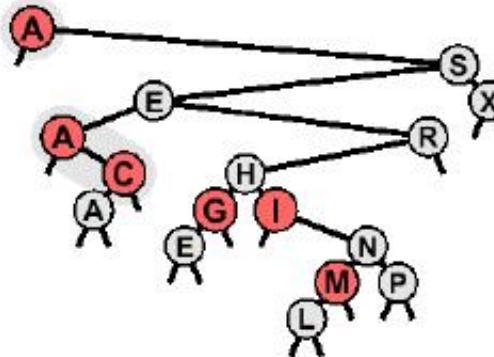
- Cost. $O(\log N')$ per insert, search, and delete, where N' is the number of elements ever inserted in the BST.

BST delete: First Approach

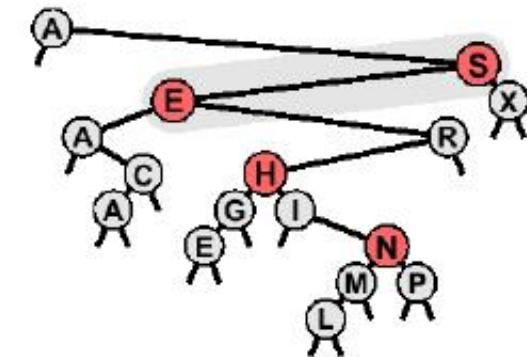
- To remove a node from a BST. [Hibbard, 1960s]
 - ◆ Zero children: just remove it.
 - ◆ One child: pass the child up.
 - ◆ Two children: find the **next largest** node using **right-left*** (**left-left-...**) swap with next largest remove as above.



zero children



one child



two children

Unsatisfactory solution. Not symmetric, code is clumsy.

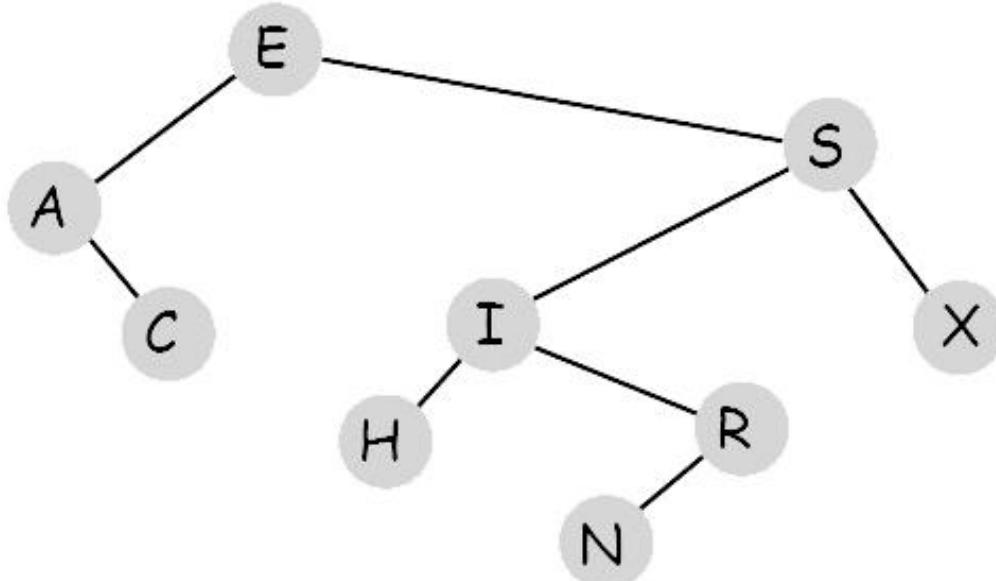
Surprising consequence. Trees not random (!) $\Rightarrow \sqrt{N}$ per op.

- Longstanding open problem: simple and efficient delete for BSTs

Deletion in Randomized BSTs

- To delete a node containing a given key
 - ◆ remove the node
 - ◆ **join** the two remaining subtrees to make a tree

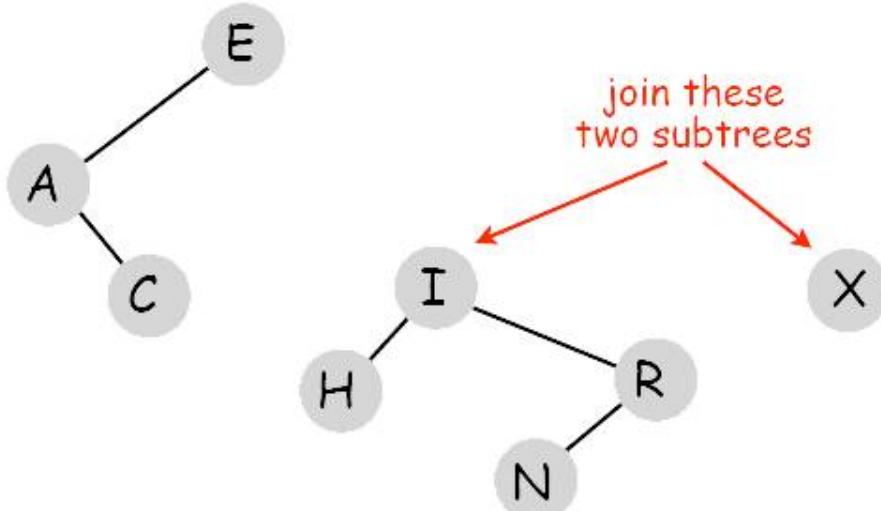
Ex. Delete **S** in



Deletion in Randomized BSTs

- To delete a node containing a given key
 - ◆ remove the node
 - ◆ **join** its two subtrees

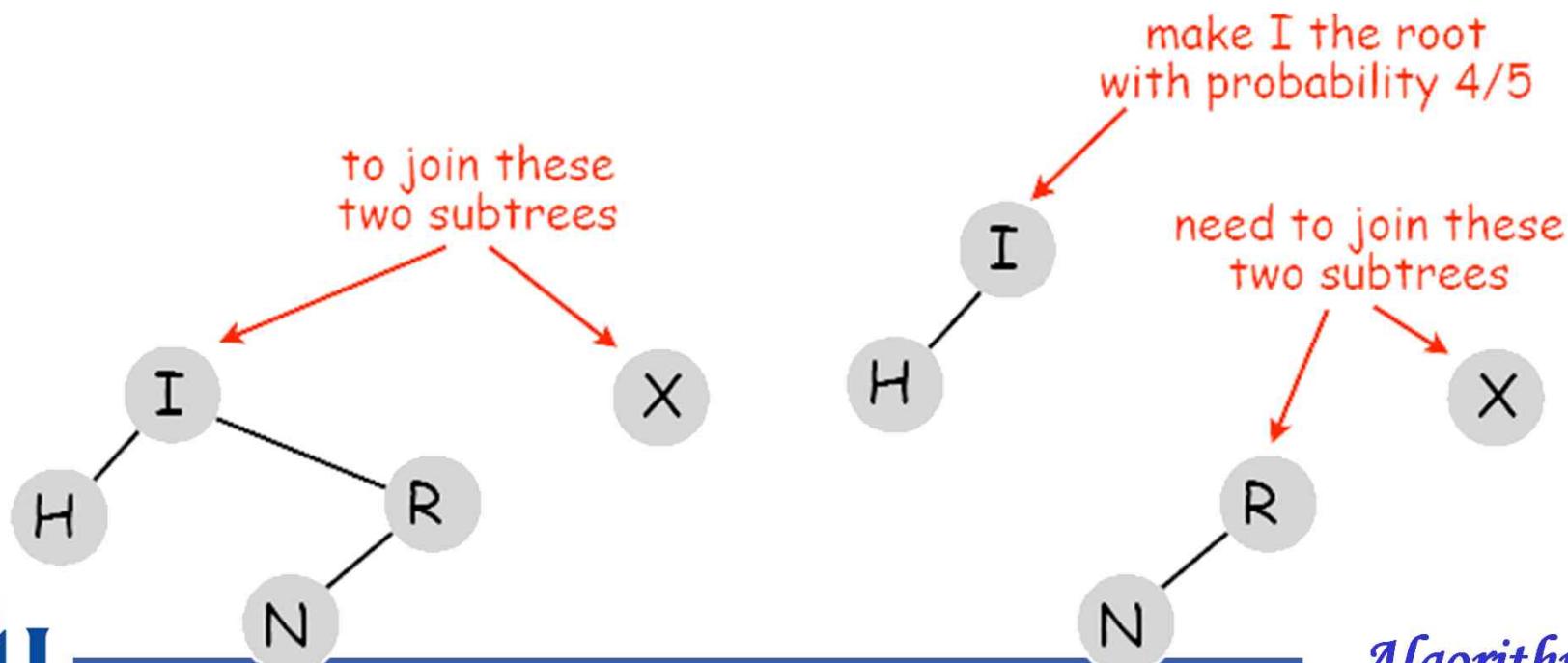
Ex. Delete S in



```
private Node remove(Node x, Key key)
{
    if (x == null)
        return null;
    int cmp = key.compareTo(x.key);
    if (cmp == 0)
        return join(x.left, x.right);
    else if (cmp < 0)
        x.left = remove(x.left, key);
    else if (cmp > 0)
        x.right = remove(x.right, key);
    return x;
}
```

Join in Randomized BSTs

- To join two subtrees with L keys in one and R keys in the other
 - ◆ maintain counts of nodes in subtrees (L and R)
 - ◆ with probability $L/(L+R)$
 - ✓ make the root of the left the root
 - ✓ make its left subtree the left subtree of the root
 - ✓ join its right subtree to R to make the right subtree of the root
 - ◆ with probability $R/(L+R)$ do the symmetric moves on the right



Join in Randomized BSTs

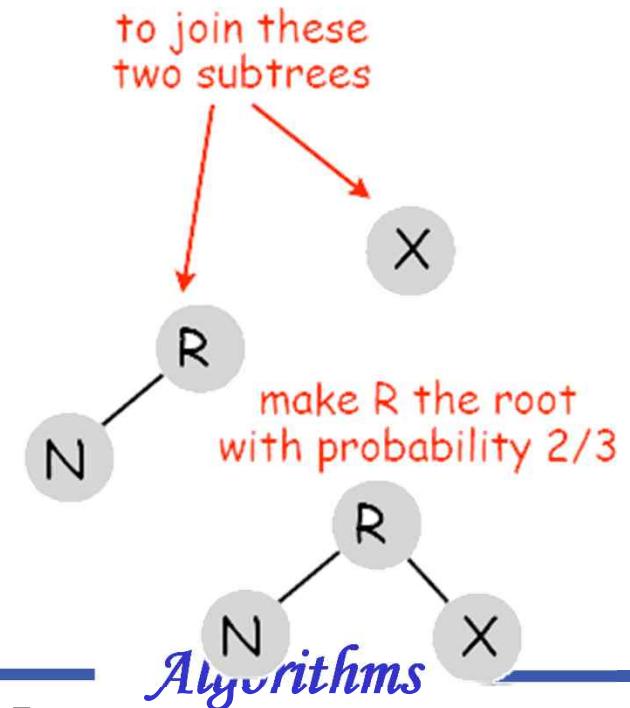
- To join two subtrees with L keys in one and R keys in the other
 - ◆ maintain counts of nodes in subtrees (L and R)
 - ◆ with probability $L/(L+R)$
 - ✓ make the root of the left the root
 - ✓ make its left subtree the left subtree of the root
 - ✓ join its right subtree to R to make the right subtree of the root
 - ◆ with probability $R/(L+R)$ do the symmetric moves on the right

```
private Node join(Node a, Node b)
{
    if (a == null) return b;
    if (b == null) return a;

    if (StdRandom.bernoulli((double) a.N / (a.N + b.N)))
        { a.right = join(a.right, b); return a; }
    else
        { b.left = join(a, b.left); return b; }
}
```

$a.N = a.N + b.N;$

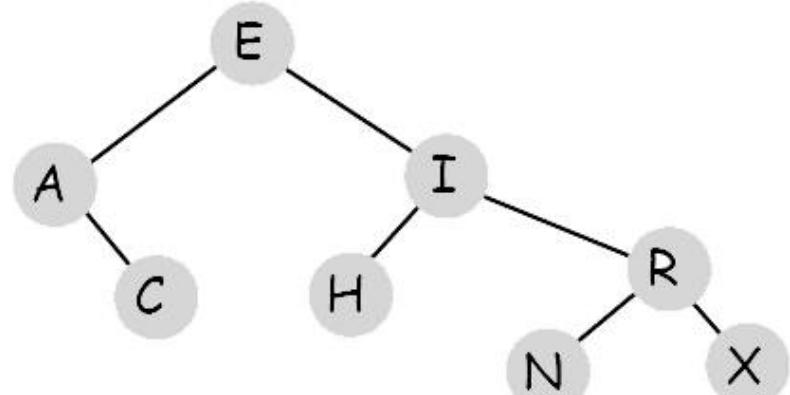
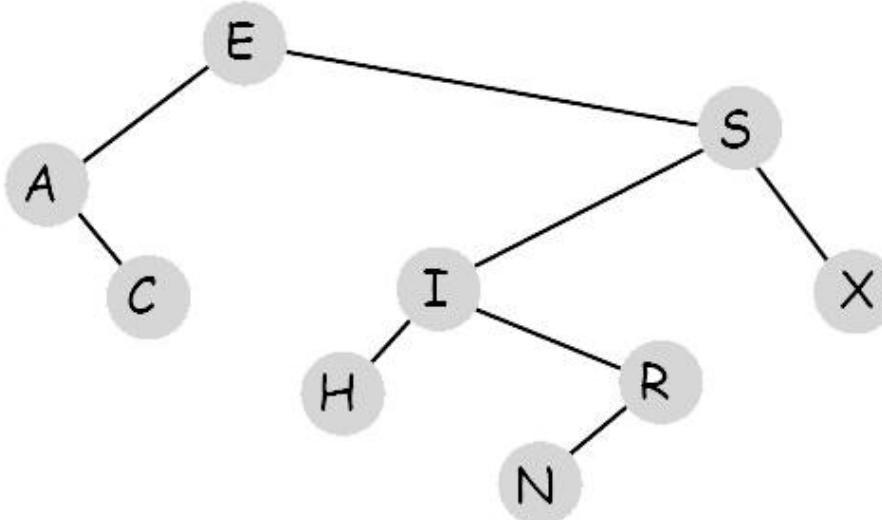
$b.N = a.N + b.N;$



Deletion in Randomized BSTs

- To delete a node containing a given key
 - ◆ remove the node
 - ◆ **join** its two subtrees

Ex. Delete **S** in



- Theorem. Tree **still random** after delete (!)
Bottom line. Logarithmic guarantee for search/insert/delete

Summary of Symbol-Table Implementations

implementation	guarantee			average case			ordered iteration?
	search	insert	delete	search	insert	delete	
unordered array	N	N	N	N/2	N/2	N/2	no
ordered array	$\lg N$	N	N	$\lg N$	N/2	N/2	yes
unordered list	N	N	N	N/2	N	N/2	no
ordered list	N	N	N	N/2	N/2	N/2	yes
BST	N	N	N	$1.38 \lg N$	$1.38 \lg N$?	yes
randomized BST	$7 \lg N$	$7 \lg N$	$7 \lg N$	$1.38 \lg N$	$1.38 \lg N$	$1.38 \lg N$	yes

- Randomized BSTs provide the desired guarantees

↑
probabilistic, with
exponentially small
chance of error

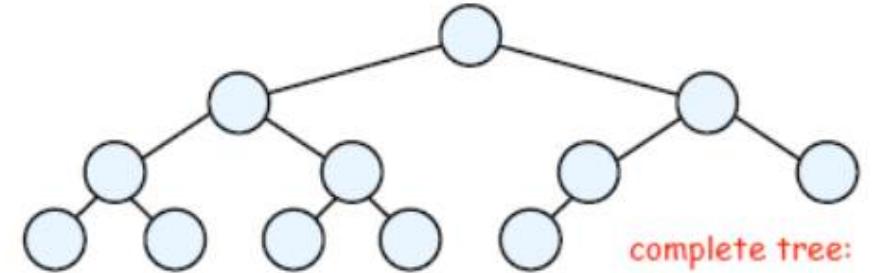
- Next lecture: Can we do better?



- **Heap:** **Array** representation of a heap-ordered complete binary tree.

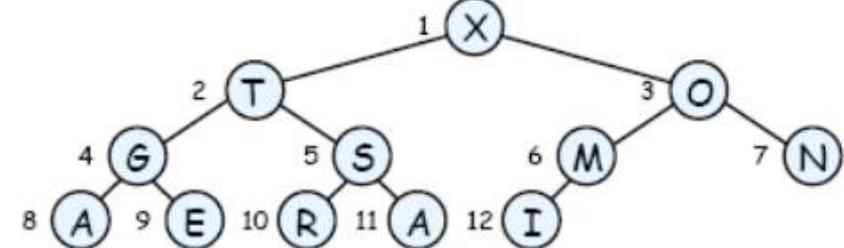
- **Binary tree.**

- ◆ Empty or
- ◆ Node with links to left and right trees.



- **Heap-ordered binary tree.**

- ◆ Keys in nodes.
- ◆ No smaller than children's keys.



- **Array representation.**

- ◆ Take nodes in level order.
- ◆ **No explicit links** needed since tree is **complete**.

1	2	3	4	5	6	7	8	9	10	11	12
X	T	O	G	S	M	N	A	E	R	A	I

BST implementation (skeleton)

DS&A-ch7, p6

```
public class BST<Key extends Comparable<Key>, Value>
    implements Iterable<Key>
{
    private Node root;                                ← instance variable

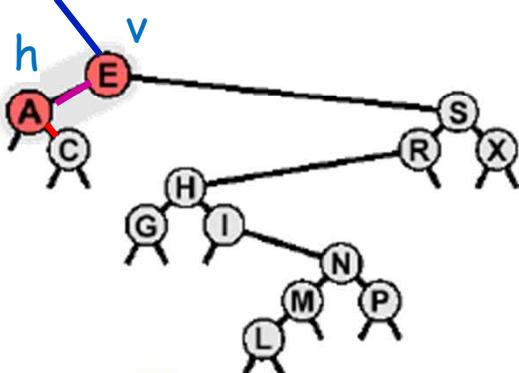
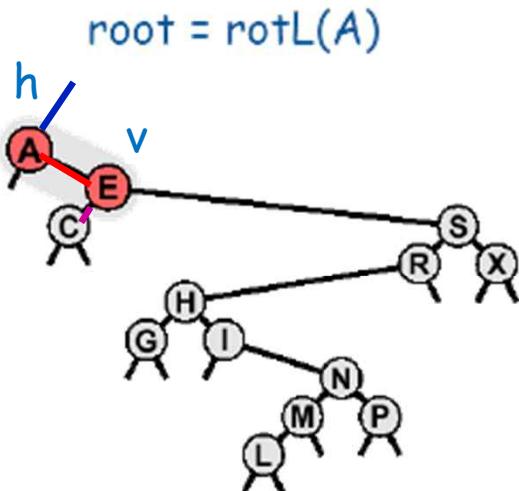
    private class Node                               ← inner class
    {
        Key key;          int N;
        Value val;
        Node left, right;
        Node(Key key, Value val)
        {
            this.key = key;
            this.val = val;
        }   this.N = 1;
    }

    public void put(Key key, Value val)
        // see next slides

    public Val get(Key key)
        // see next slides
    }
}
```



- Fundamental operation to rearrange nodes in a tree.
 - ◆ easier done than said
 - ◆ raise some nodes, lowers some others

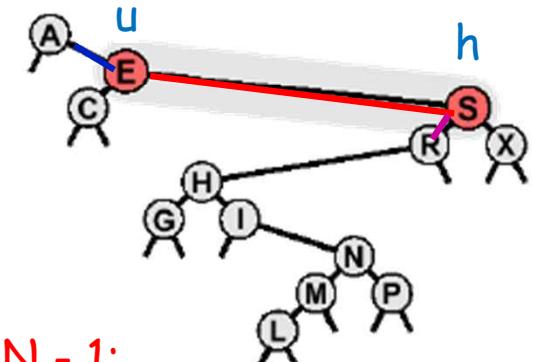
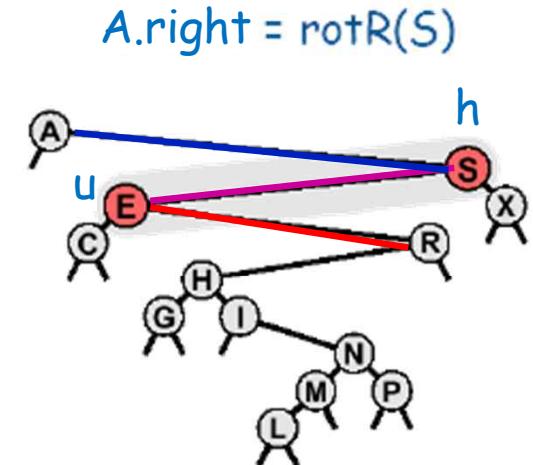


```
private Node rotL(Node h)
{
    Node v = h.r;
    h.r = v.l;
    v.l = h;
    return v;
}
```

$$v.N = h.N; \\ h.N = v.N - v.right.N - 1;$$

```
private Node rotR(Node h)
{
    Node u = h.l;
    h.l = u.r;
    u.r = h;
    return u;
}
```

$$u.N = h.N; \\ h.N = u.N - u.left.N - 1;$$



Recursive BST Root Insertion

DS&A-ch7 , p21

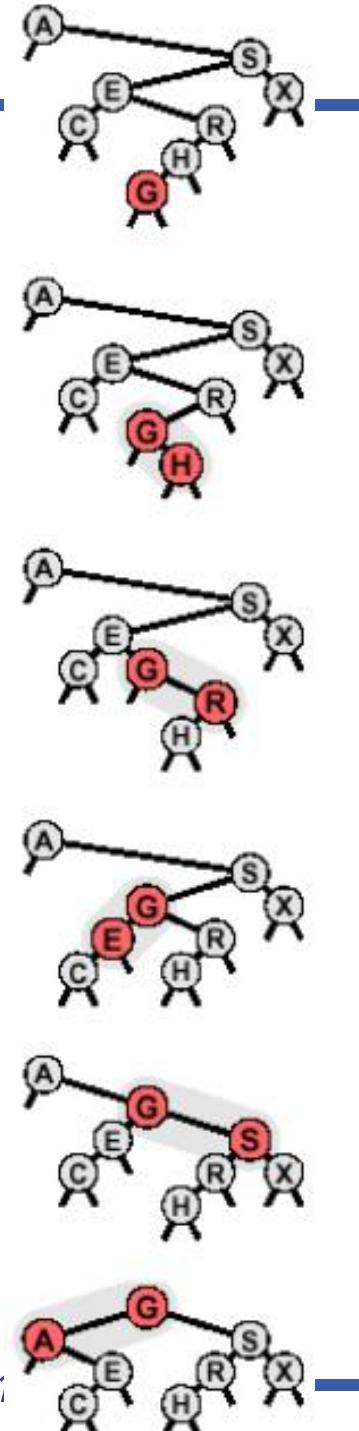
- Root insertion: insert a node and make it the new root.

- ◆ Insert as in standard BST.
- ◆ Rotate inserted node to the root.
- ◆ Easy recursive implementation

Caution: **very** tricky recursive code.
Read very carefully!

```
private Node putRoot(Node x, Key key, Val val)
{
    if (x == null) return new Node(key, val);
    int cmp = key.compareTo(x.key);
    if (cmp == 0) x.val = val;
    else if (cmp < 0)
        { x.left = putRoot(x.left, key, val); x = rotR(x); }
    else if (cmp > 0)
        { x.right = putRoot(x.right, key, val); x = rotL(x); }
    return x;
}
```

Algo:



Data Structures & Algorithms

8. Balanced Trees

- 2-3-4 trees
- red-black trees
- B-trees



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

Symbol Table Review

- Symbol table: key-value pair abstraction.
 - ◆ **Insert** a value with specified key.
 - ◆ **Search** for value given key.
 - ◆ **Delete** value with given key.
- Randomized BST.
 - ◆ Guarantee of $\sim c \lg N$ time per operation (probabilistic).
 - ◆ Need subtree count in each node.
 - ◆ Need random numbers for each insert/delete op.
- This lecture. 2-3-4 trees, **left-leaning red-black trees**, B-trees.

↑
new for Fall 2007!



Summary of symbol-table implementations

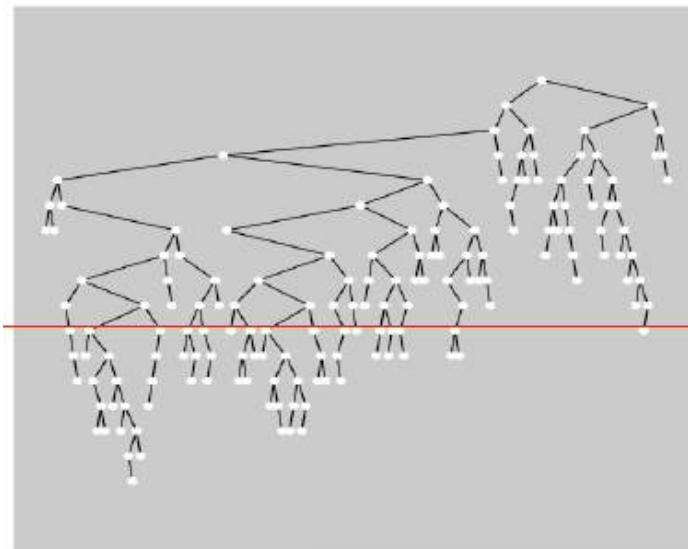
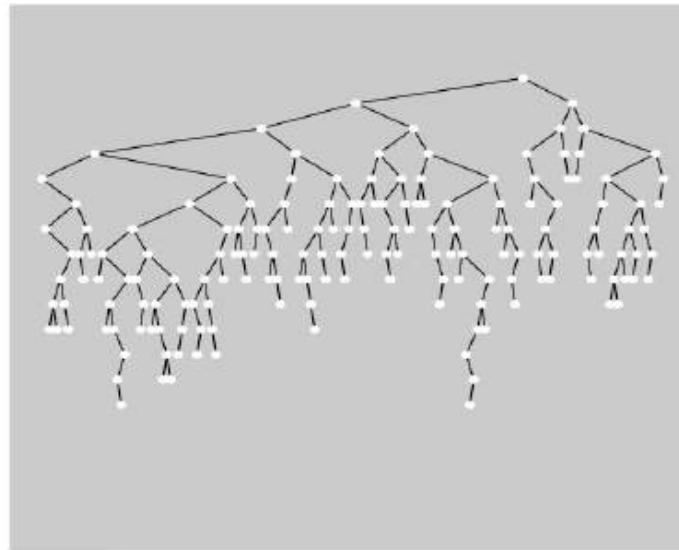
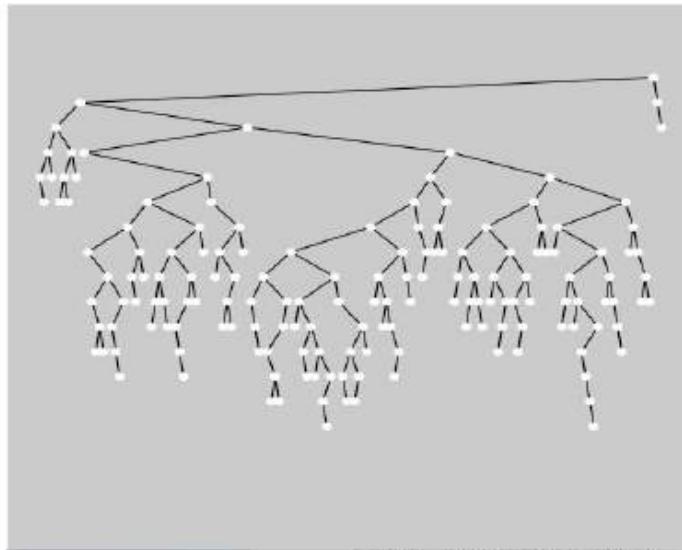
implementation	guarantee			average case			ordered iteration?
	search	insert	delete	search	insert	delete	
unordered array	N	N	N	N/2	N/2	N/2	no
ordered array	$\lg N$	N	N	$\lg N$	N/2	N/2	yes
unordered list	N	N	N	N/2	N	N/2	no
ordered list	N	N	N	N/2	N/2	N/2	yes
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$?	yes
randomized BST	$7 \lg N$	$7 \lg N$	$7 \lg N$	$1.39 \lg N$	$1.39 \lg N$	$1.39 \lg N$	yes

- Randomized BSTs provide the desired guarantees

probabilistic, with
exponentially small
chance of quadratic time

- This lecture: Can we do better?

Typical random BSTs



$$\begin{aligned}N &= 250 \\ \lg N &\approx 8 \\ 1.39 \lg N &\approx 11\end{aligned}$$

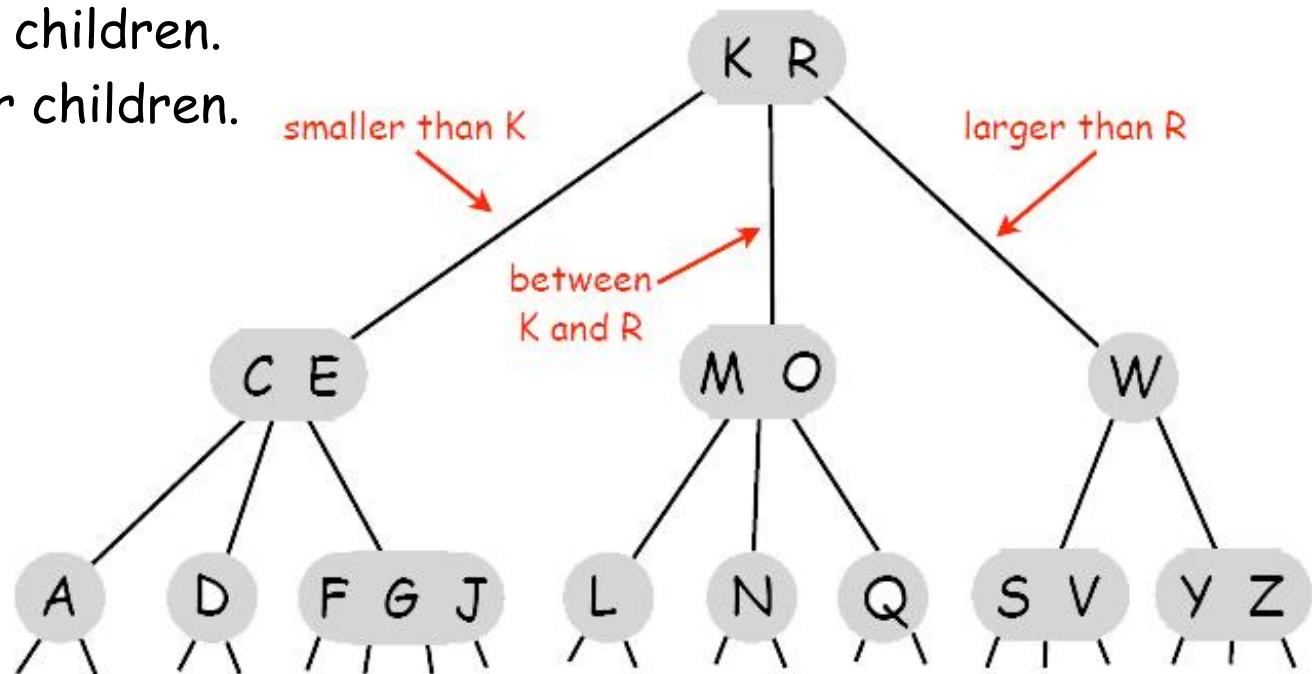
average node depth

8. Balanced Trees

- 2-3-4 trees
- red-black trees
- B-trees

2-3-4 Tree

- 2-3-4 tree. Generalize node to allow multiple keys; keep tree balanced.
- Perfect balance. Every path from root to leaf has same length.
- Allow 1, 2, or 3 keys per node.
 - ◆ 2-node: one key, two children.
 - ◆ 3-node: two keys, three children.
 - ◆ 4-node: three keys, four children.

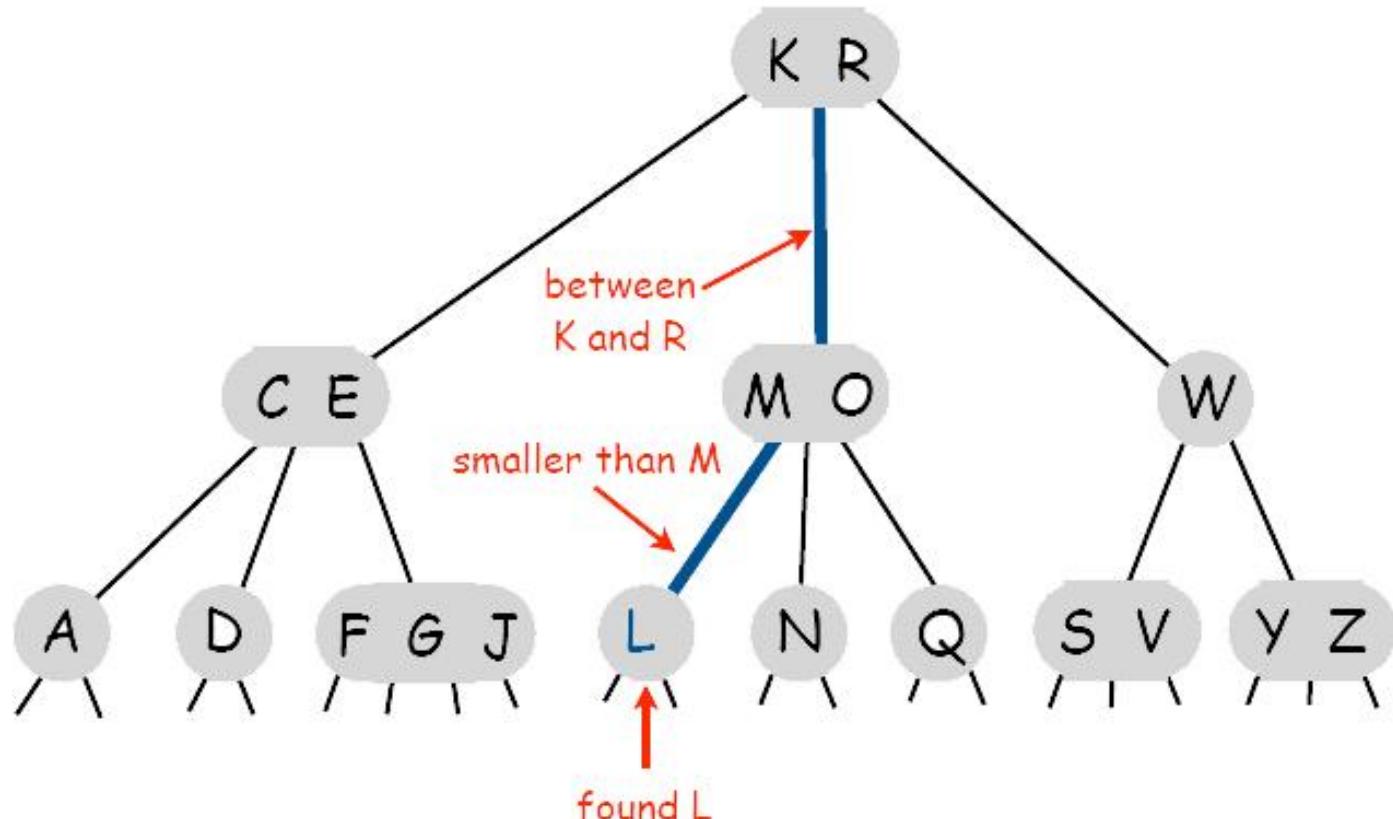


Searching in a 2-3-4 Tree

□ Search.

- ◆ Compare search key against keys in node.
- ◆ Find interval containing search key.
- ◆ Follow associated link (recursively).

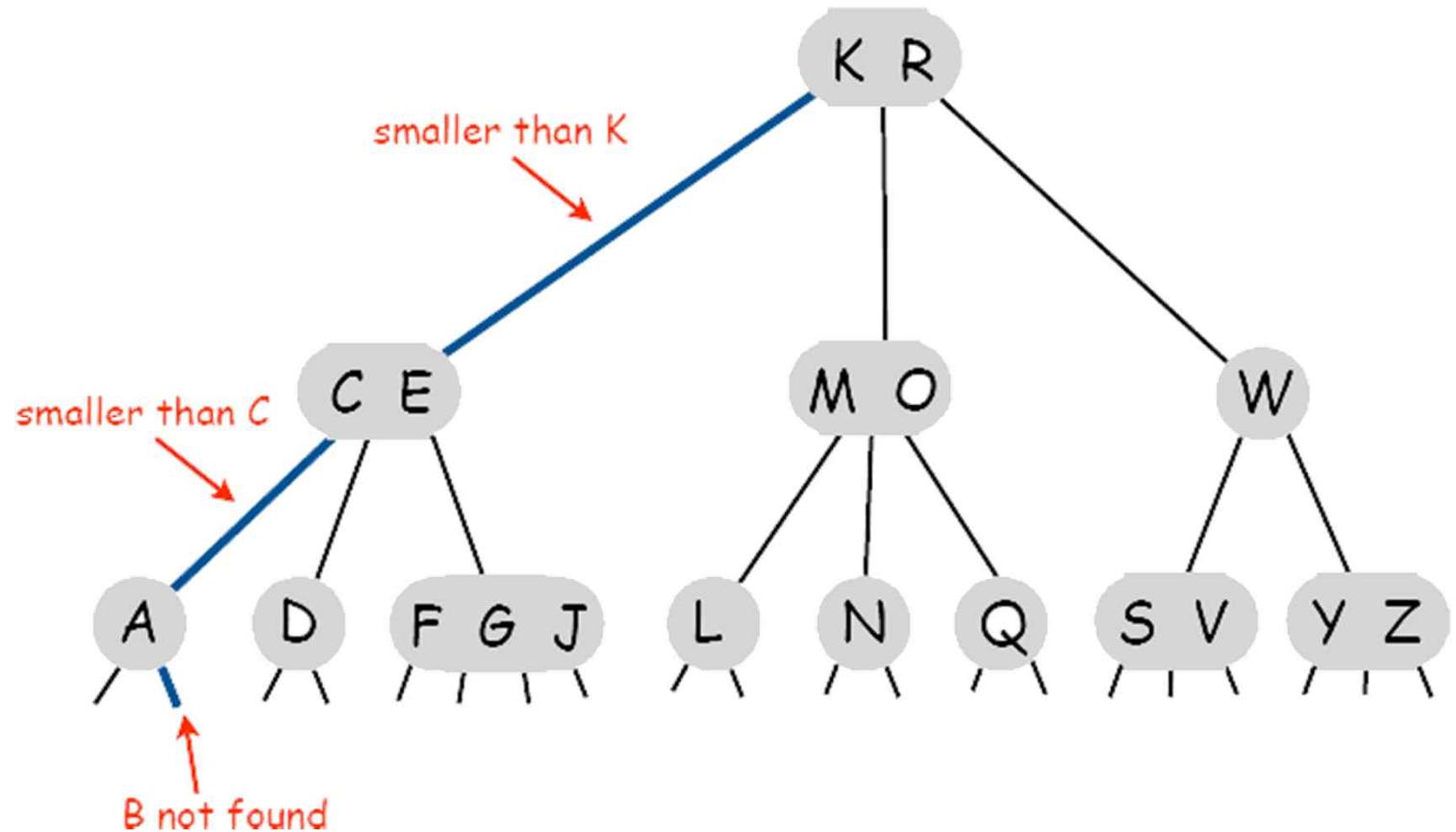
□ Ex. Search for L



Insertion in a 2-3-4 Tree

- Insert.
 - ◆ Search to bottom for key.

- Ex. Insert B

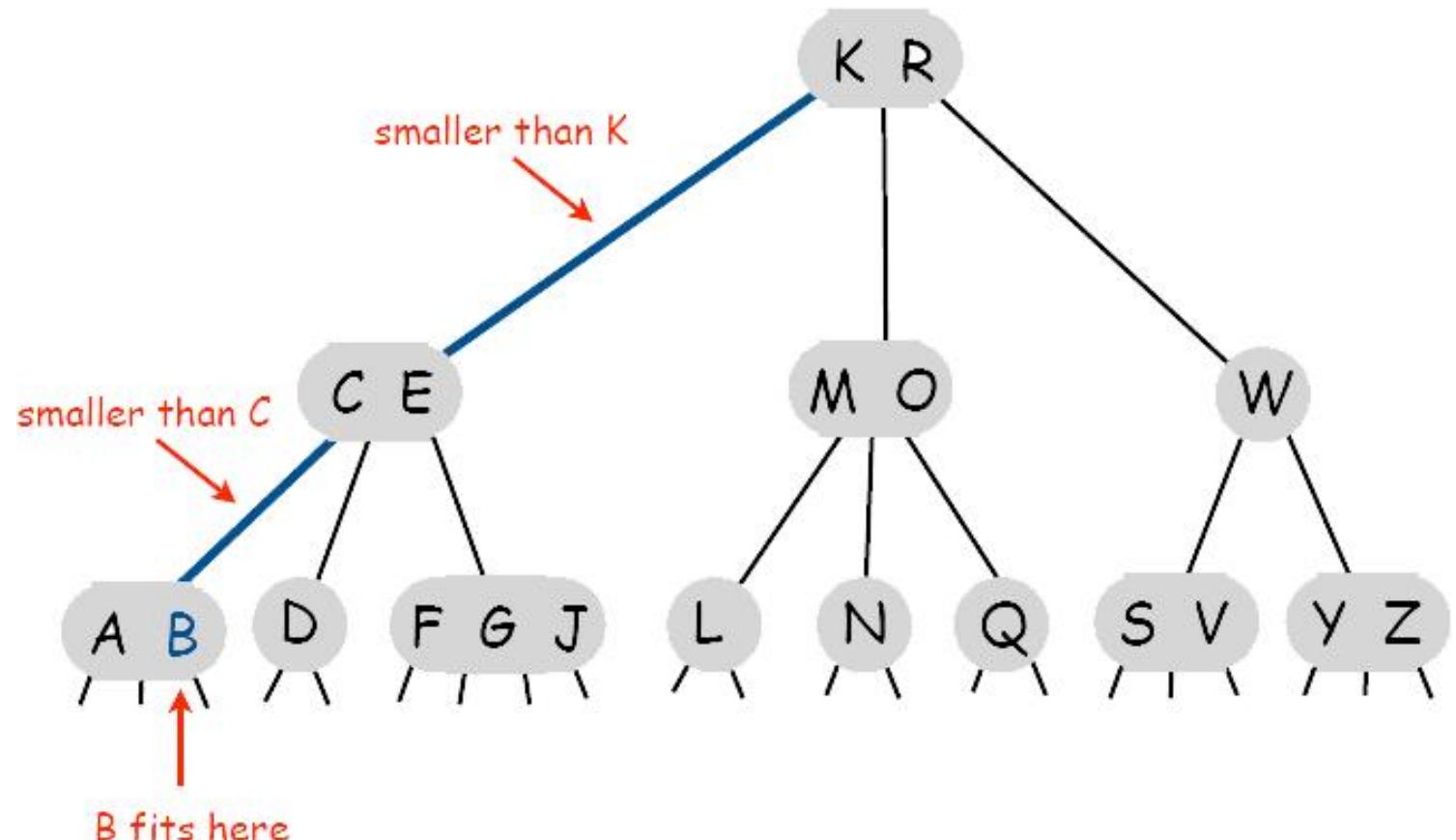


Insertion in a 2-3-4 Tree

□ Insert.

- ◆ Search to bottom for key.
- ◆ 2-node at bottom: convert to 3-node.

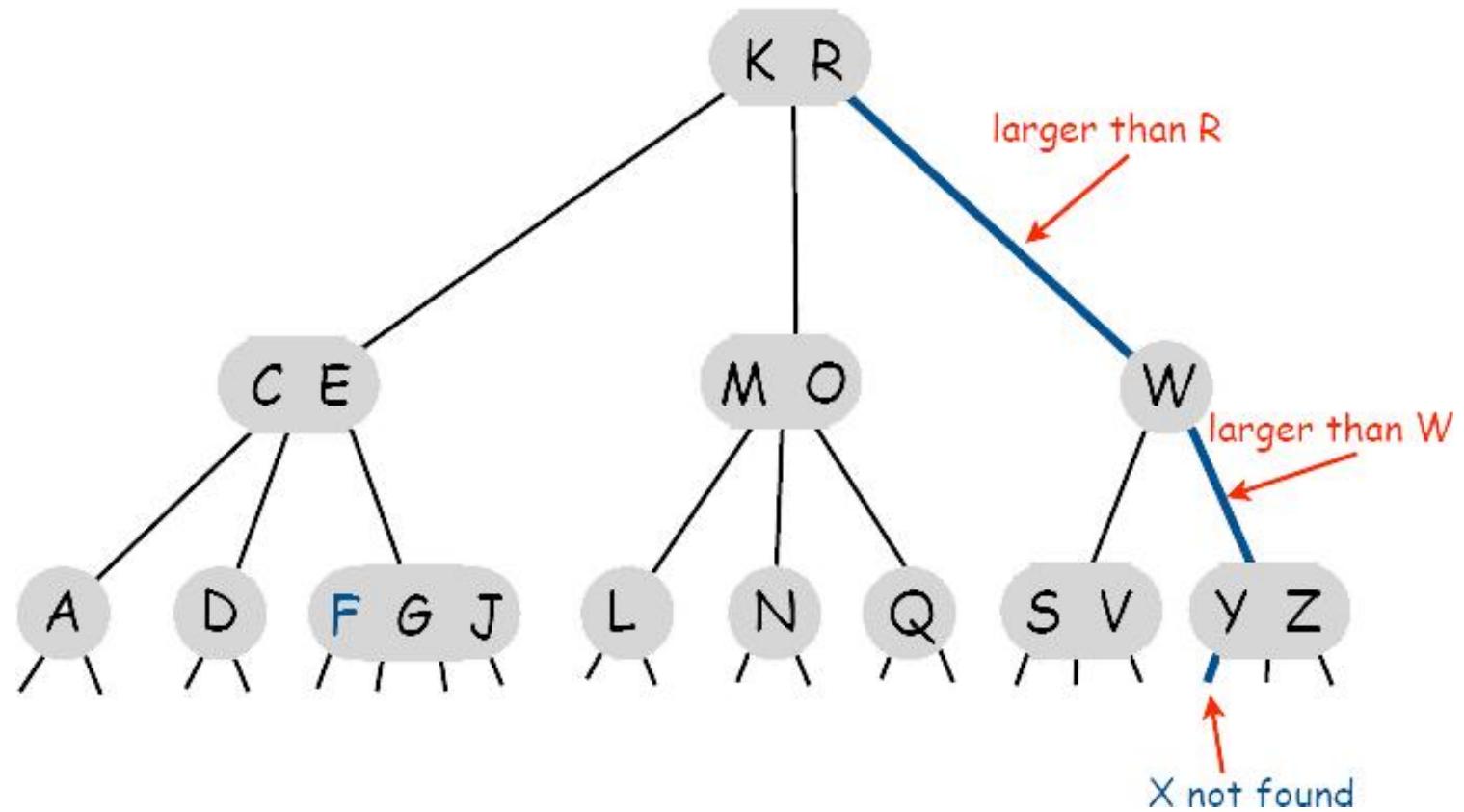
□ Ex. Insert B



Insertion in a 2-3-4 Tree

- Insert.
 - ◆ Search to bottom for key.

- Ex. Insert X

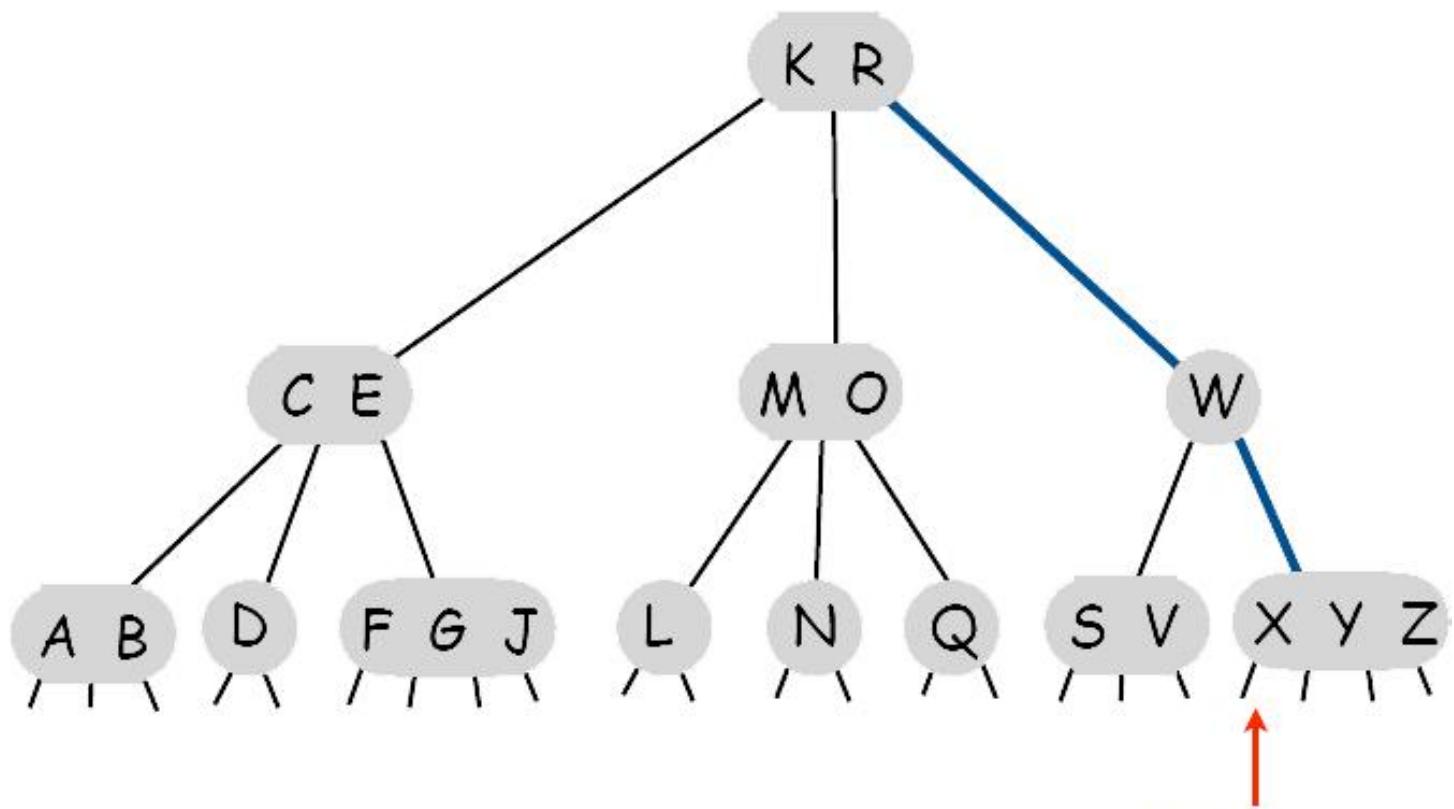


Insertion in a 2-3-4 Tree

□ Insert.

- ◆ Search to bottom for key.
- ◆ 2-node at bottom: convert to 3-node.
- ◆ 3-node at bottom: convert to 4-node.

□ Ex. Insert X



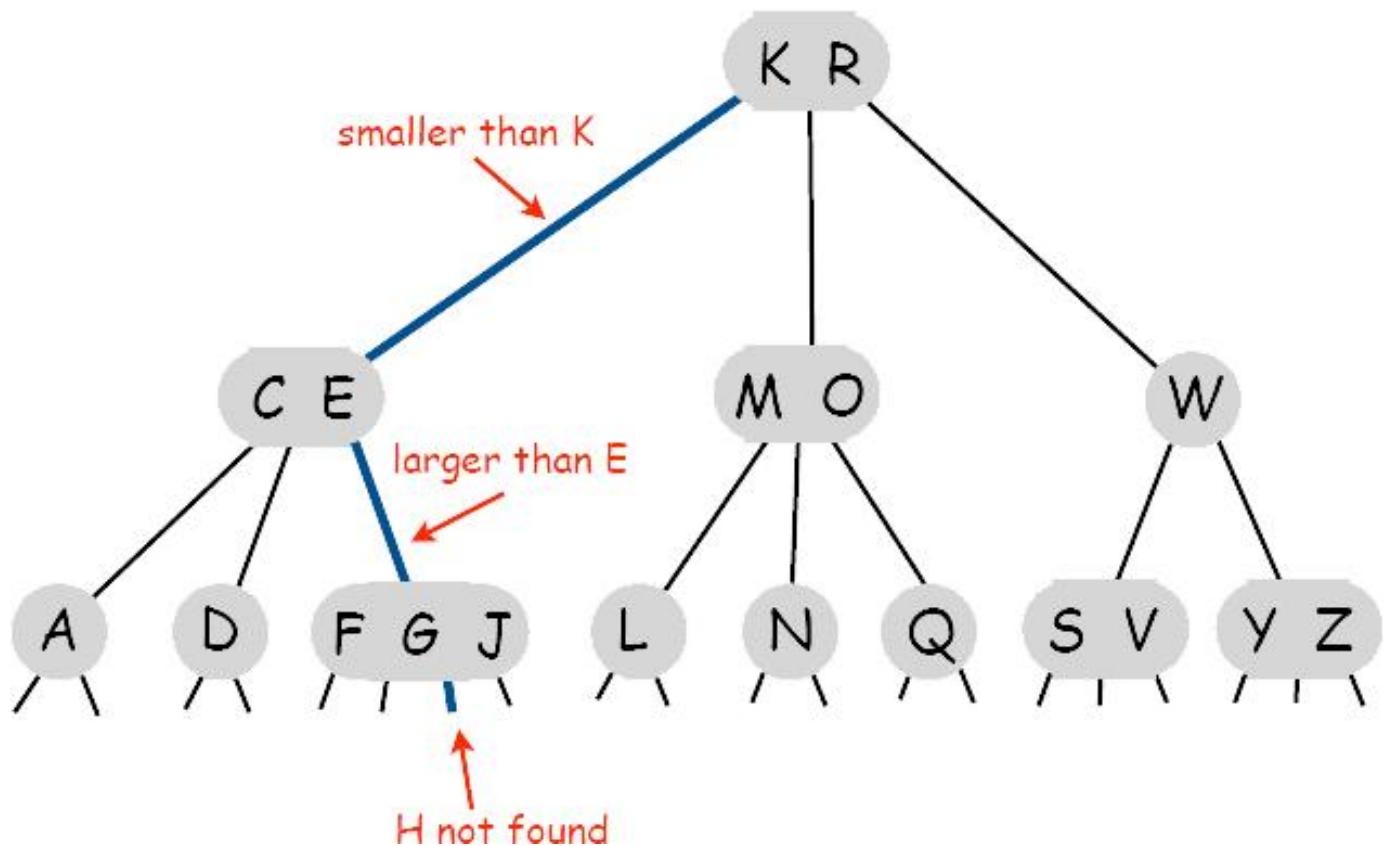
X fits here

Algorithms

Insertion in a 2-3-4 Tree

- Insert.
 - ◆ Search to bottom for key.

- Ex. Insert H

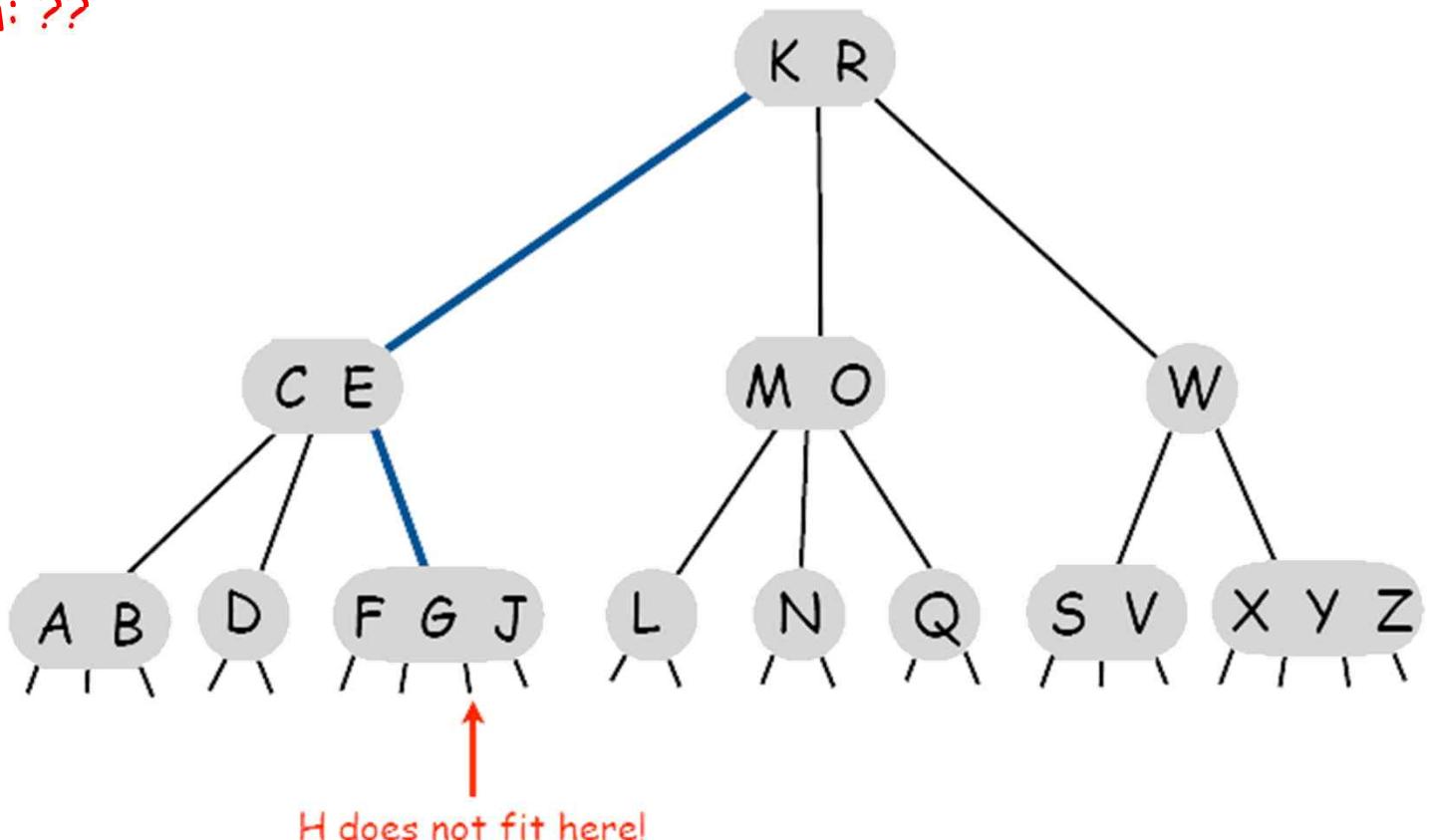


Insertion in a 2-3-4 Tree

□ Insert.

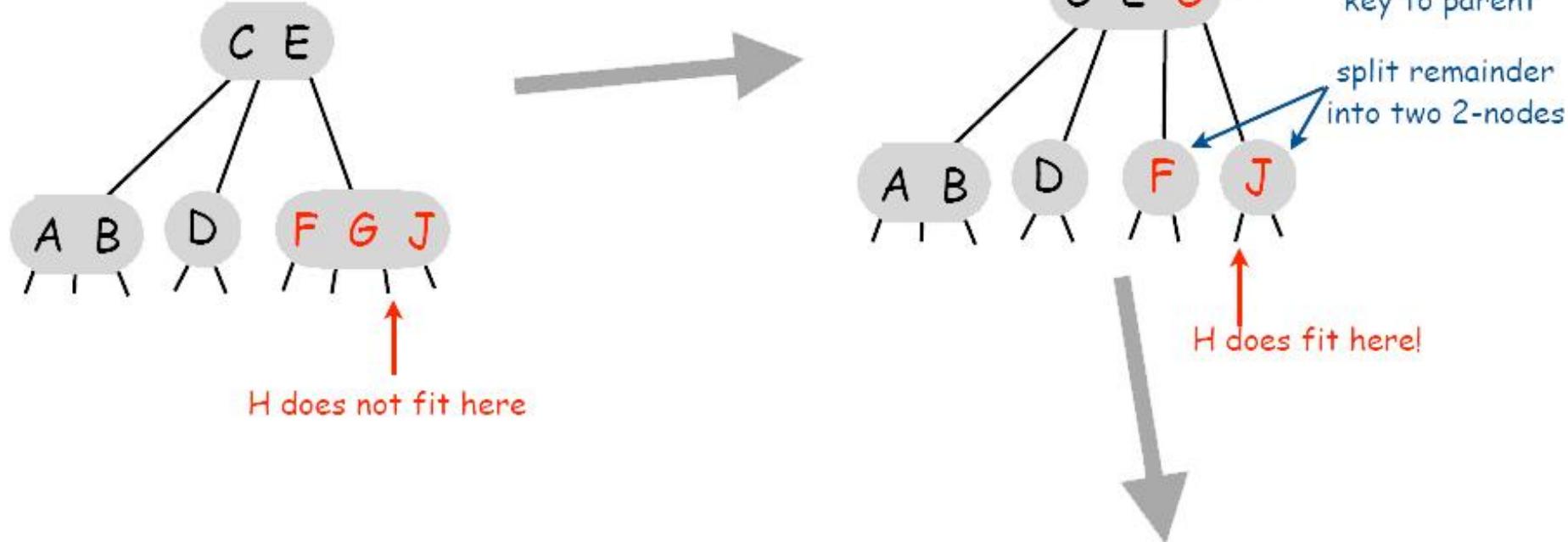
- ◆ Search to bottom for key.
- ◆ 2-node at bottom: convert to 3-node.
- ◆ 3-node at bottom: convert to 4-node.
- ◆ 4-node at bottom: ??

□ Ex. Insert H

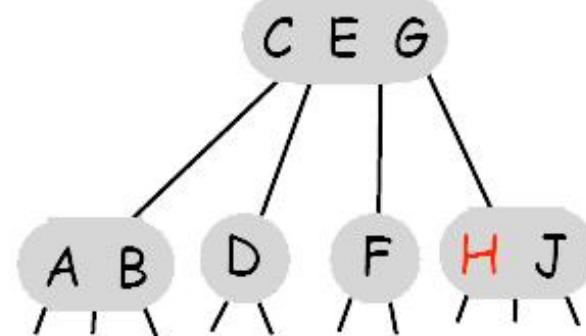


Splitting a 4-node in a 2-3-4 tree

- Idea: split the 4-node to make room



- Problem: Doesn't work if parent is a 4-node
- Solution 1: Split the parent
 - (and continue splitting up while necessary).
- Solution 2: Split 4-nodes on the way down.



Splitting a 4-node in a 2-3-4 tree

- Idea: split 4-nodes on the way down the tree.
 - ◆ Ensures that most recently seen node is not a 4-node.
 - ◆ Transformations to split 4-nodes:



local transformations
that work **anywhere**
in the tree

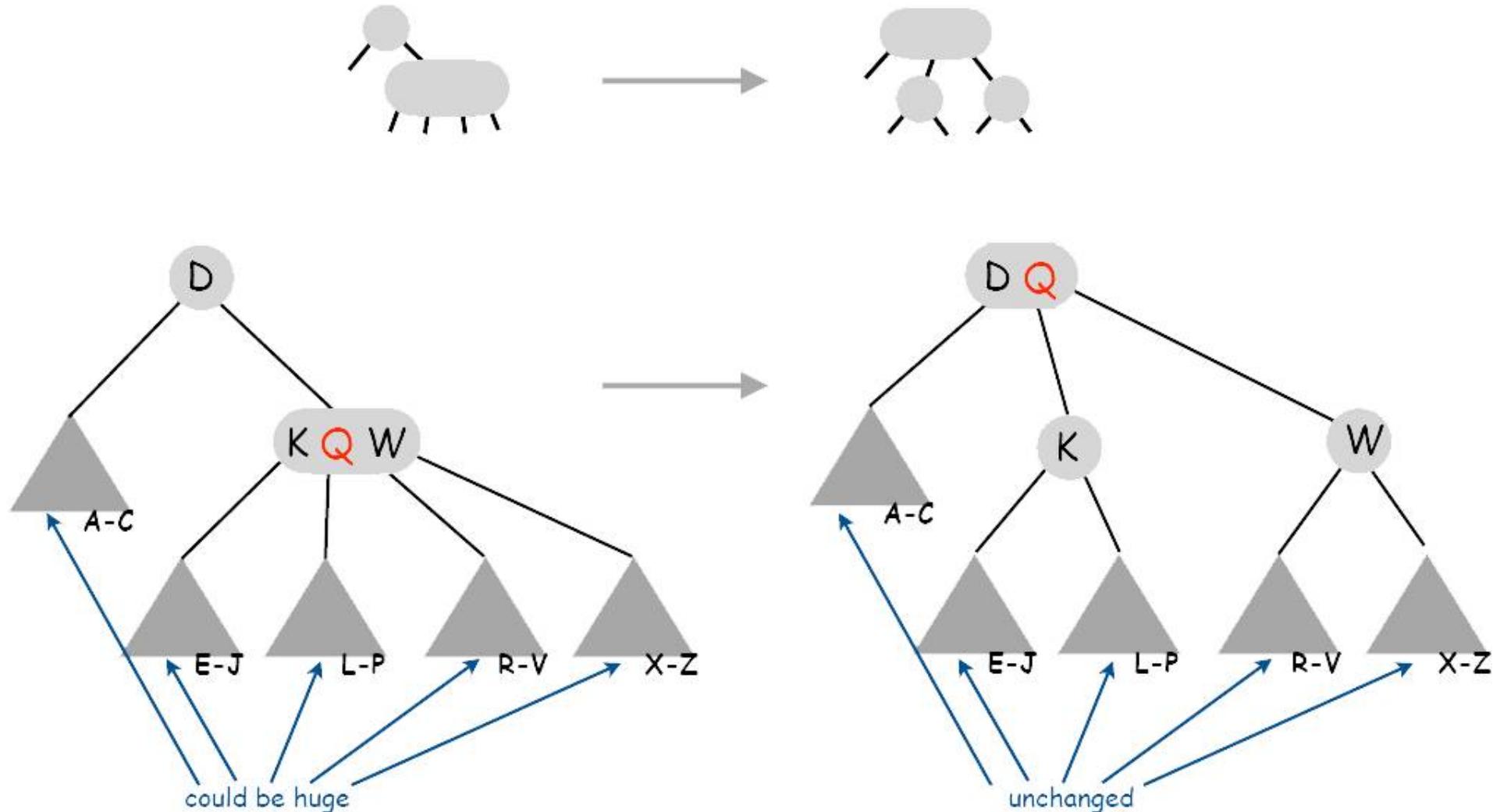


Invariant. **Current node is not a 4-node.**

- Consequences
 - ◆ 4-node below a 4-node case never happens
 - ◆ insertion at bottom node is easy since it's not a 4-node.

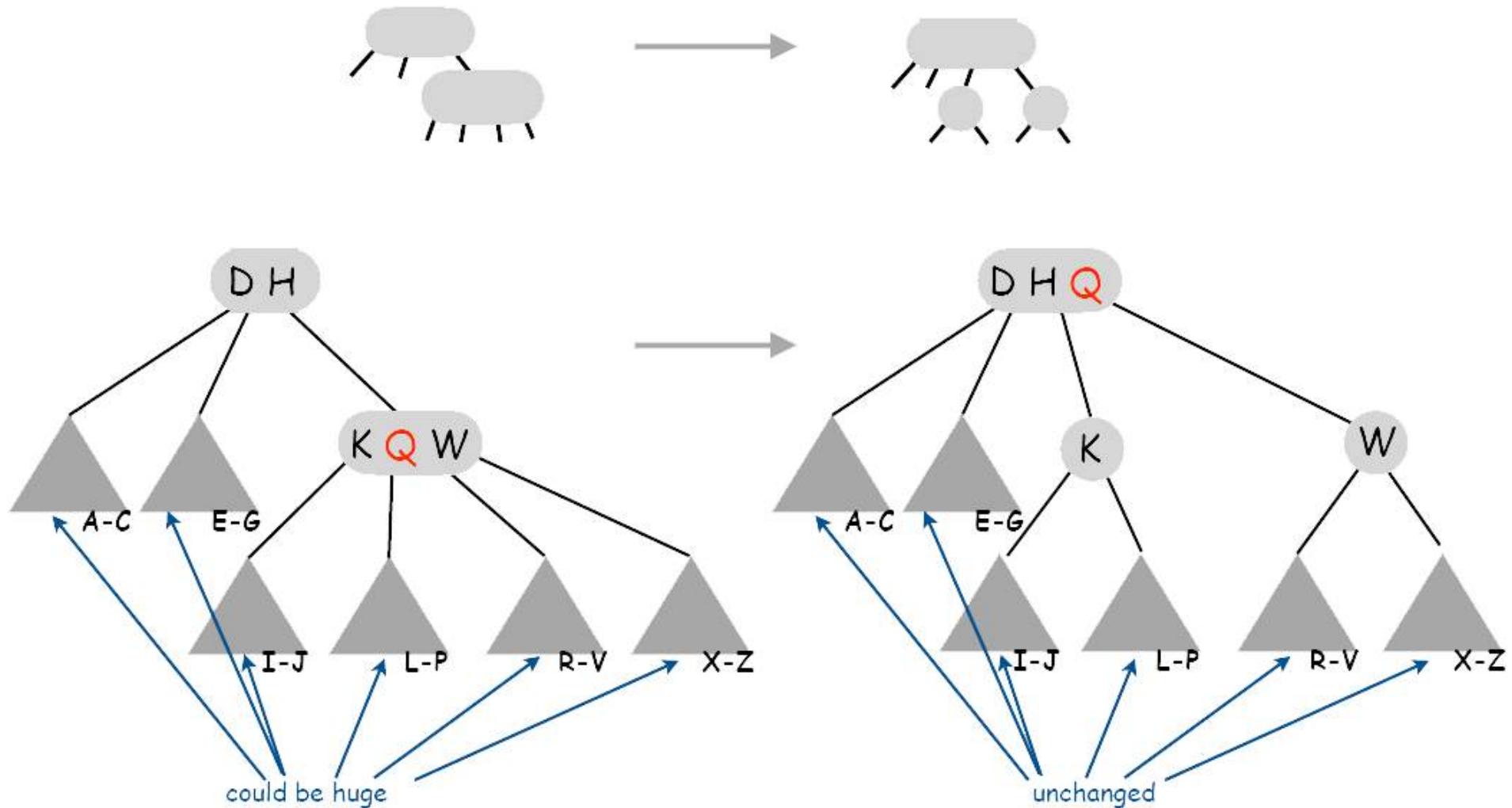
Splitting a 4-node below a 2-node in a 2-3-4 tree

- A **local** transformation that works anywhere in the tree



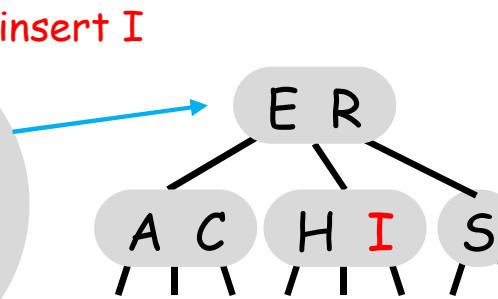
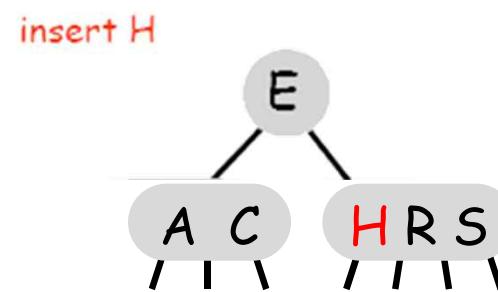
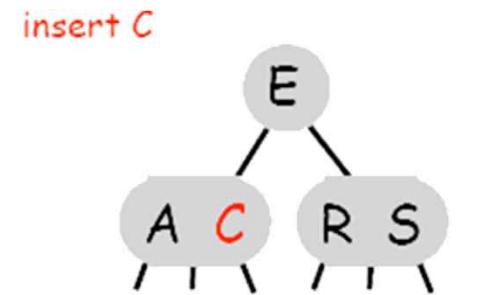
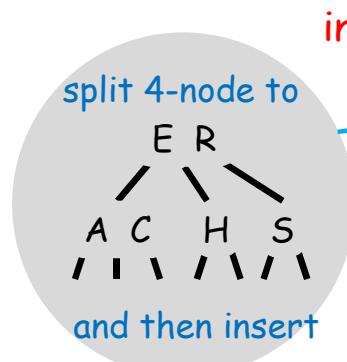
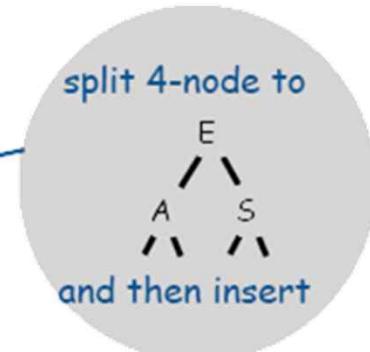
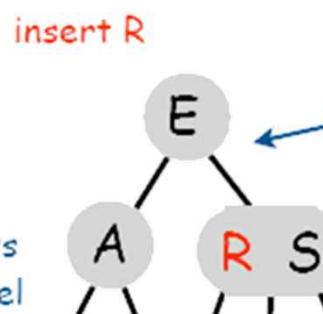
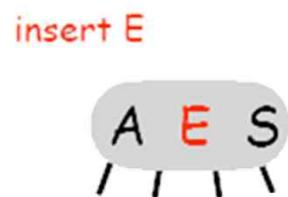
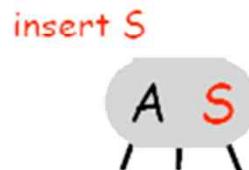
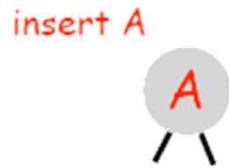
Splitting a 4-node below a 3-node in a 2-3-4 tree

- A **local** transformation that works anywhere in the tree



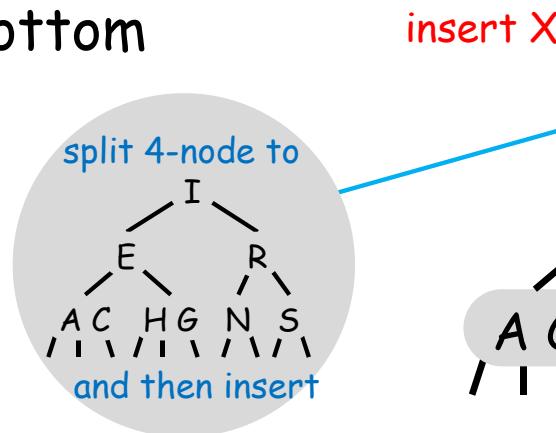
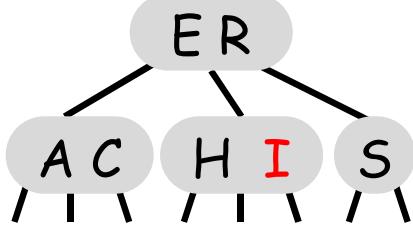
Growth of a 2-3-4 tree

- Tree grows **up** from the bottom

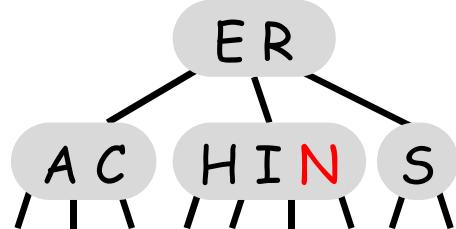


Growth of a 2-3-4 tree (continued)

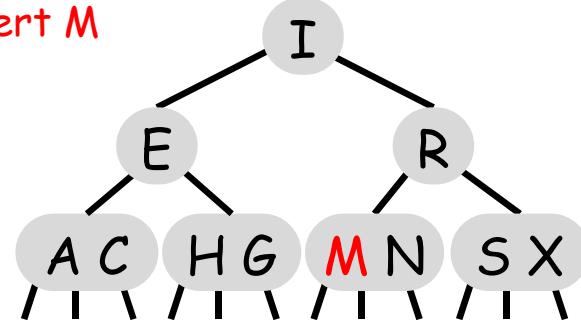
- Tree grows **up** from the bottom



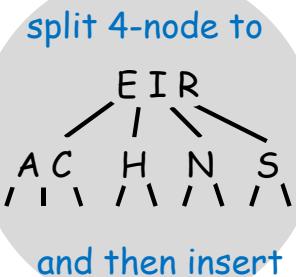
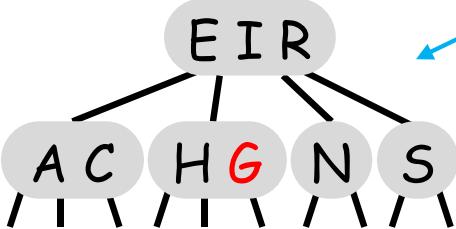
insert N



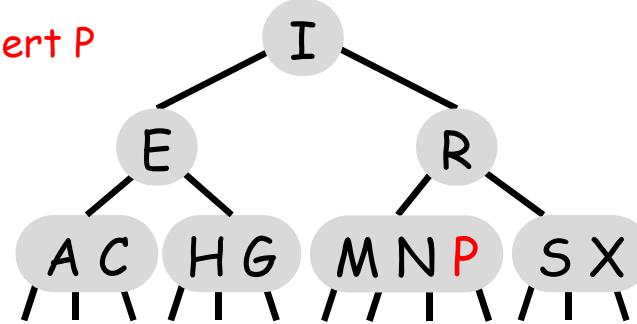
insert M



insert G

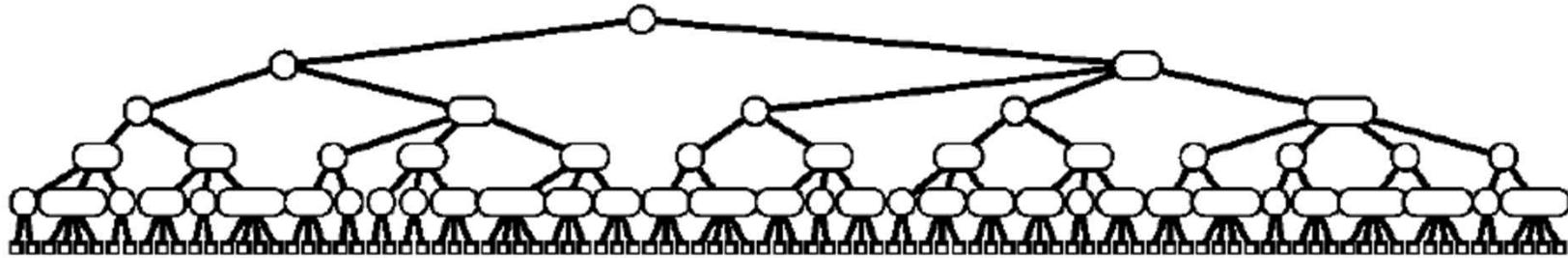


insert P



Balance in 2-3-4 trees

- Key property: All paths from root to leaf have same length.



Tree height.

- Worst case: $\lg N$ [all 2-nodes]
- Best case: $\log_4 N = 1/2 \lg N$ [all 4-nodes]
- Between 10 and 20 for a million nodes.
- Between 15 and 30 for a billion nodes.

2-3-4 Tree: Implementation?

- Direct implementation is complicated, because:
 - ◆ Maintaining multiple node types is cumbersome.
 - ◆ Implementation of getChild() involves multiple compares.
 - ◆ Large number of cases for split(), make3Node(), and make4Node().

```
private void insert(Key key, Val val)
{
    Node x = root;
    while (x.getChild(key) != null)
    {
        x = x.getChild(key);
        if (x.is4Node()) x.split();
    }
    if (x.is2Node()) x.make3Node(key, val);
    else if (x.is3Node()) x.make4Node(key, val);
}
```

fantasy code

- Bottom line: could do it, but stay tuned for an easier way.



Summary of symbol-table implementations

implementation	guarantee			average case			ordered iteration?
	search	insert	delete	search	insert	delete	
unordered array	N	N	N	N/2	N/2	N/2	no
ordered array	$\lg N$	N	N	$\lg N$	N/2	N/2	yes
unordered list	N	N	N	N/2	N	N/2	no
ordered list	N	N	N	N/2	N/2	N/2	yes
BST	N	N	N	$1.38 \lg N$	$1.38 \lg N$?	yes
randomized BST	$7 \lg N$	$7 \lg N$	$7 \lg N$	$1.38 \lg N$	$1.38 \lg N$	$1.38 \lg N$	yes
2-3-4 tree	$c \lg N$	$c \lg N$		$c \lg N$	$c \lg N$		yes

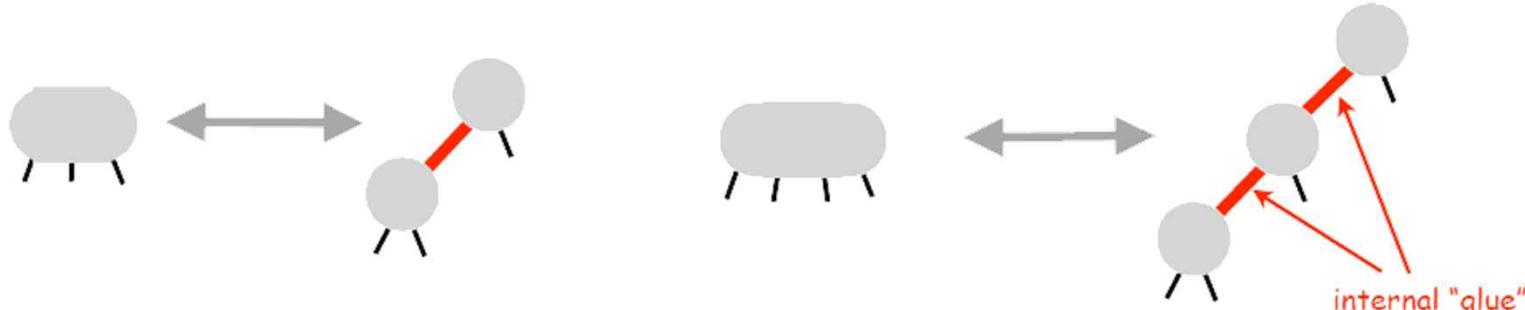
c constants depend upon implementation

8. Balanced Trees

- 2-3-4 trees
- red-black trees
- B-trees

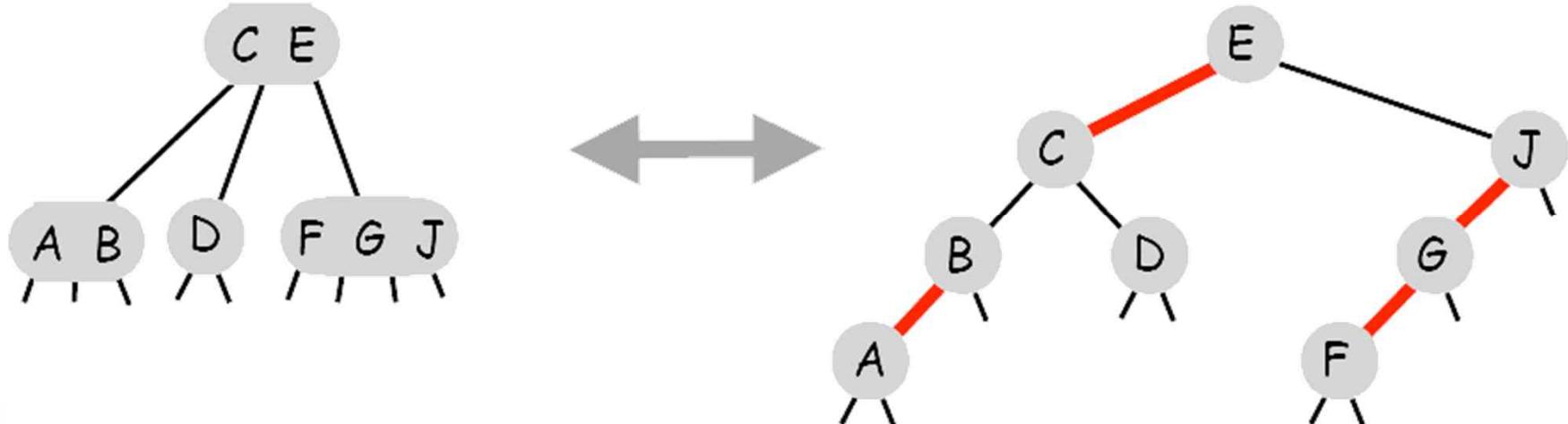
Left-leaning red-black trees (Guibas-Sedgewick, 1979 and Sedgewick, 2007)

1. Represent 2-3-4 tree as a BST.
2. Use "internal" **left-leaning** edges for 3- and 4- nodes.



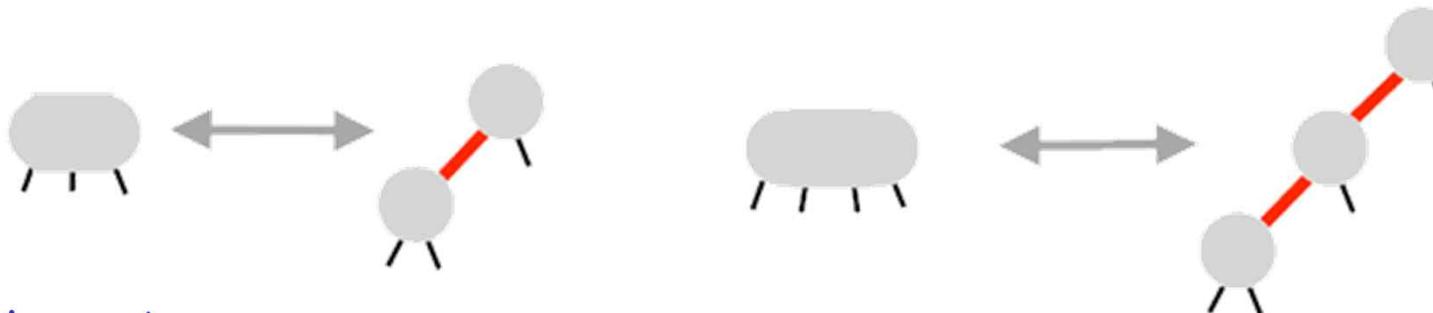
Key Properties

- elementary BST search works
- 1-1 correspondence between 2-3-4 and left-leaning red-black trees



Left-leaning red-black trees

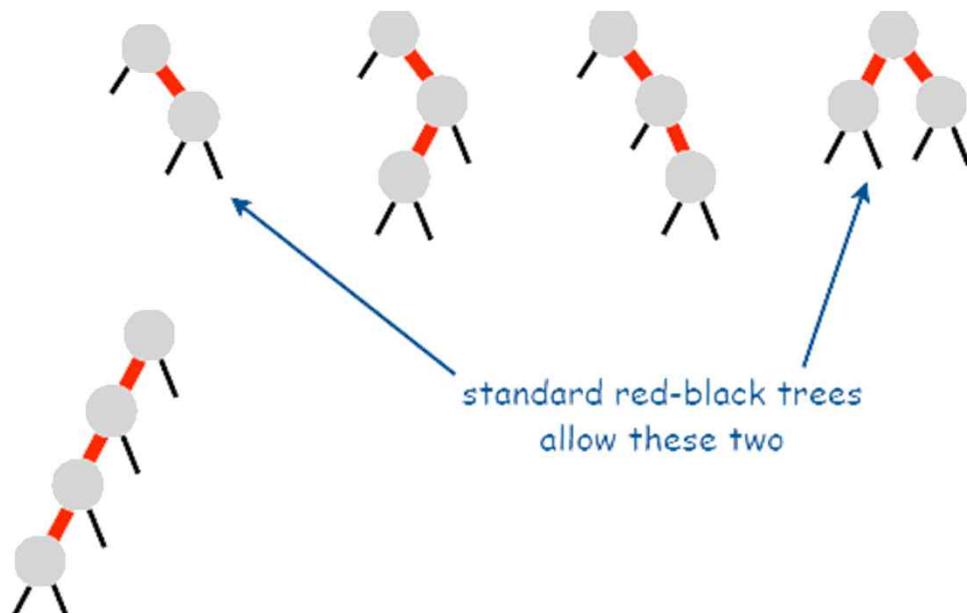
1. Represent 2-3-4 tree as a BST.
2. Use "internal" **left-leaning** edges for 3- and 4- nodes.



□ Disallowed:

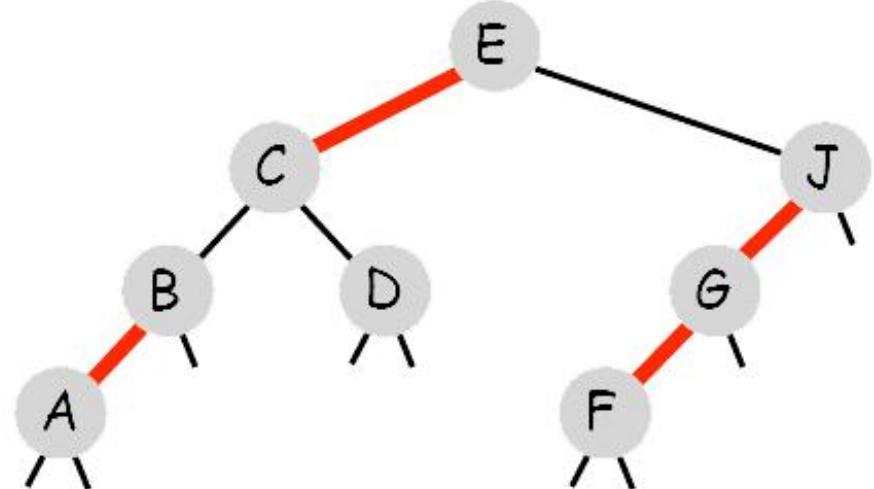
- ◆ right-leaning red edges

- ◆ three red edges in a row



Search implementation for red-black trees

```
public Val get(Key key)
{
    Node x = root;
    while (x != null)
    {
        int cmp = key.compareTo(x.key);
        if (cmp == 0)      return x.val;
        else if (cmp < 0) x = x.left;
        else if (cmp > 0) x = x.right;
    }
    return null;
}
```



Search code is **the same** as elementary BST (ignores the color)
[runs faster because of better balance in tree]

- Note: iterator code is also the same.

Insert implementation for red-black trees (skeleton)

```
public class BST<Key extends Comparable<Key>, Value>
    implements Iterable<Key>
{
    private static final boolean RED = true;
    private static final boolean BLACK = false;
    private Node root;

    private class Node
    {
        Key key;
        Value val;
        Node left, right; color of incoming link
        boolean color;
        Node(Key key, Value val, boolean color)
        {
            this.key = key;
            this.val = val;
            this.color = color;
        }
    }

    public void put(Key key, Value val)
    {
        root = put(root, key, val);
        root.color = BLACK;
    }
}
```

color of incoming link

helper method to test node color

```
private boolean isRed(Node x)
{
    if (x == null) return false;
    return (x.color == RED);
}
```

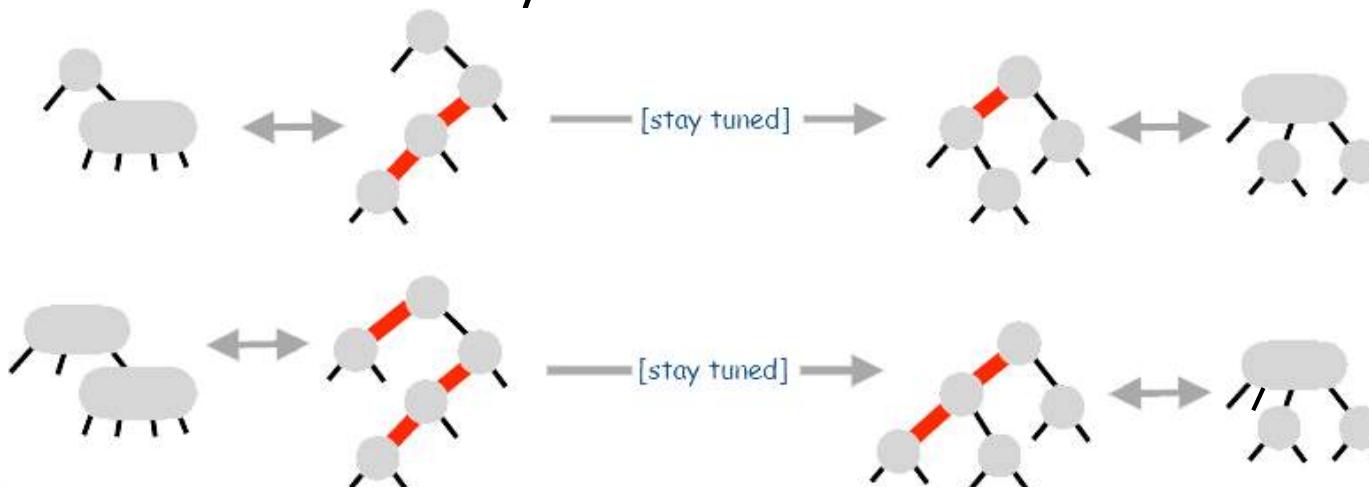
Insert implementation for left-leaning red-black trees (strategy)

- Basic idea: maintain 1-1 correspondence with 2-3-4 trees

1. If key found on recursive search, reset value as usual
2. If key not found, **insert a new red node at the bottom**



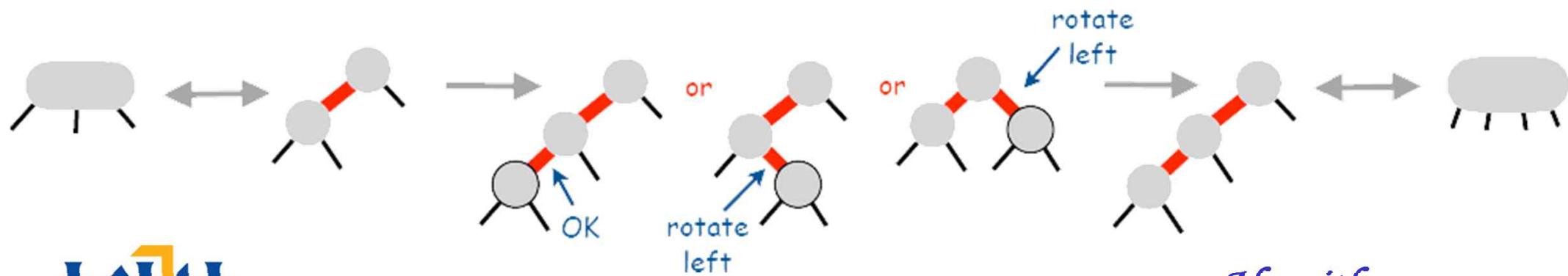
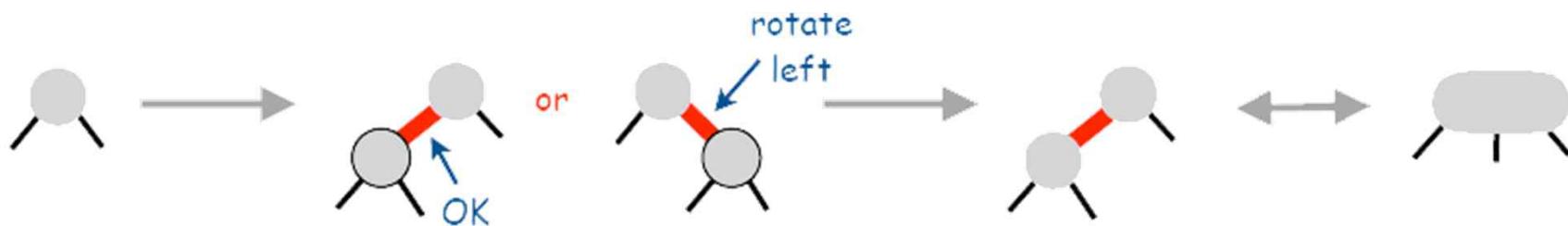
3. **Split 4-nodes** on the way DOWN the tree.



Inserting a new node at the bottom in a LLRB tree

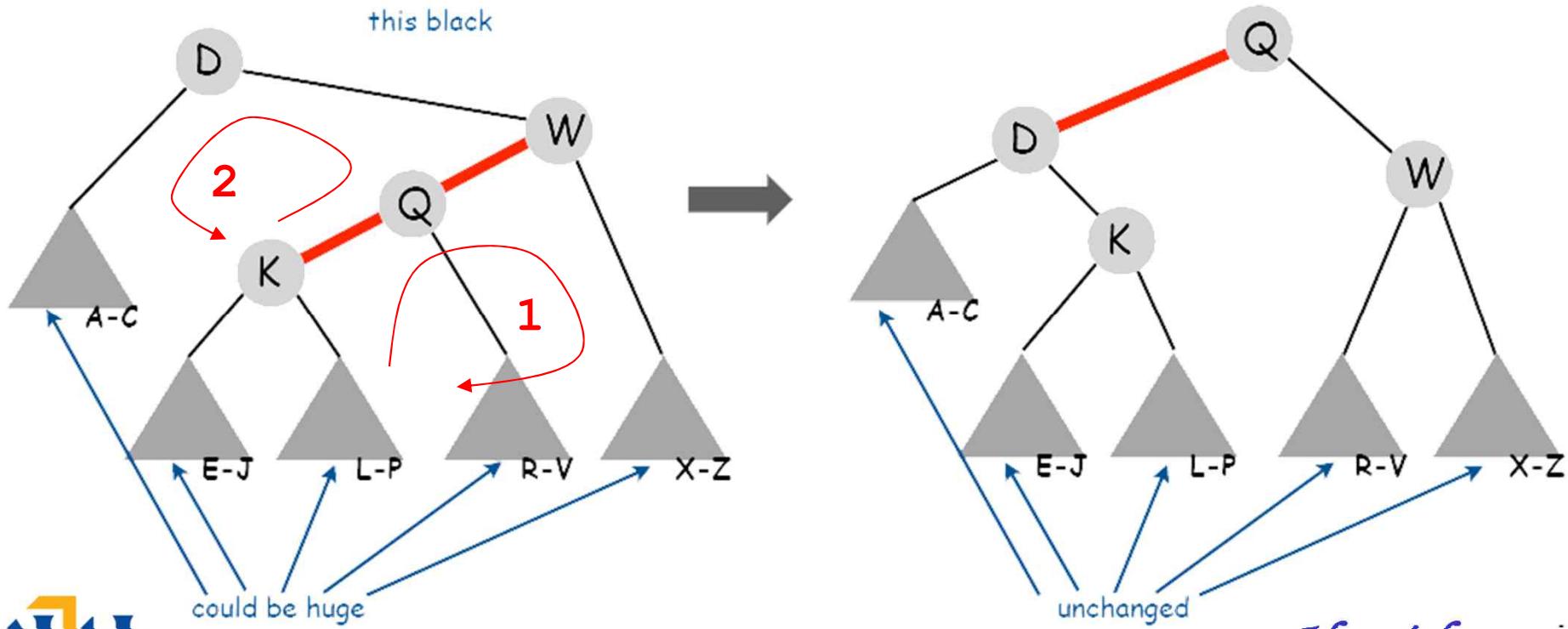
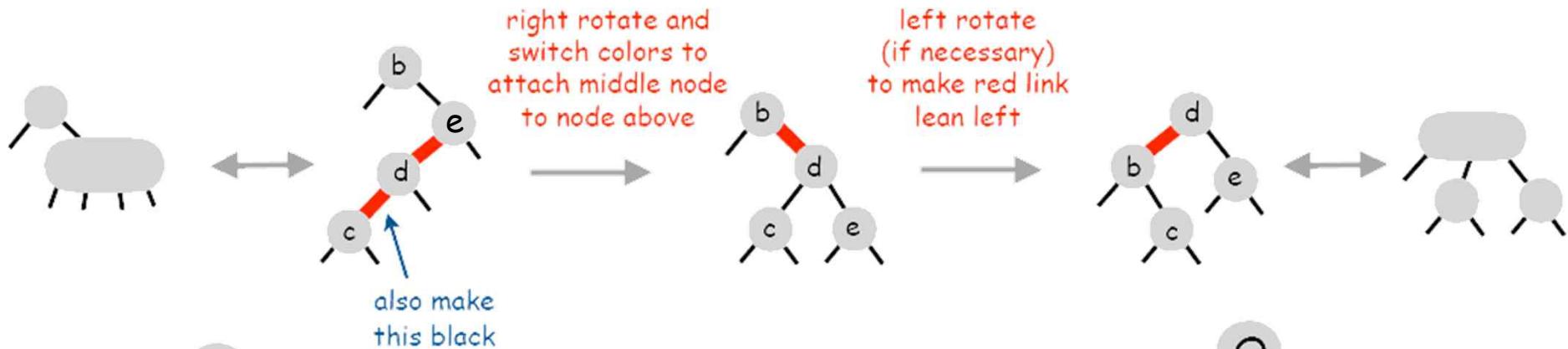
- Maintain 1-1 correspondence with 2-3-4 trees

- Add new node as usual, with red link to glue it to node above
- Rotate left if necessary to make link lean left



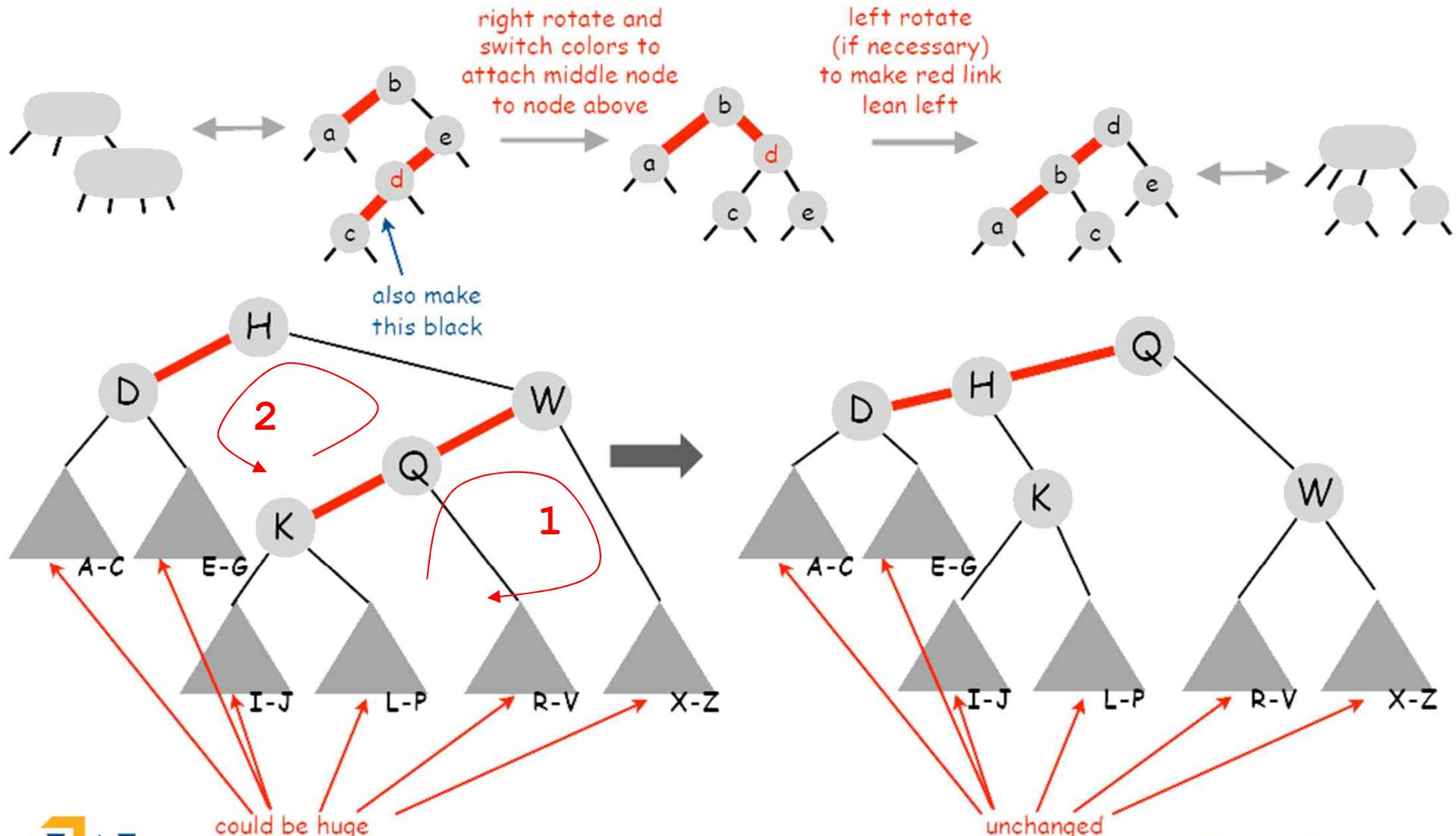
Splitting a 4-node below a 2-node in a left-leaning red-black tree

- Maintain correspondence with 2-3-4 trees



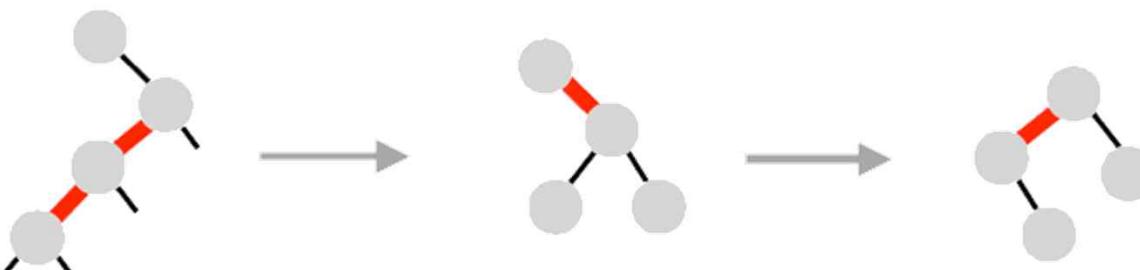
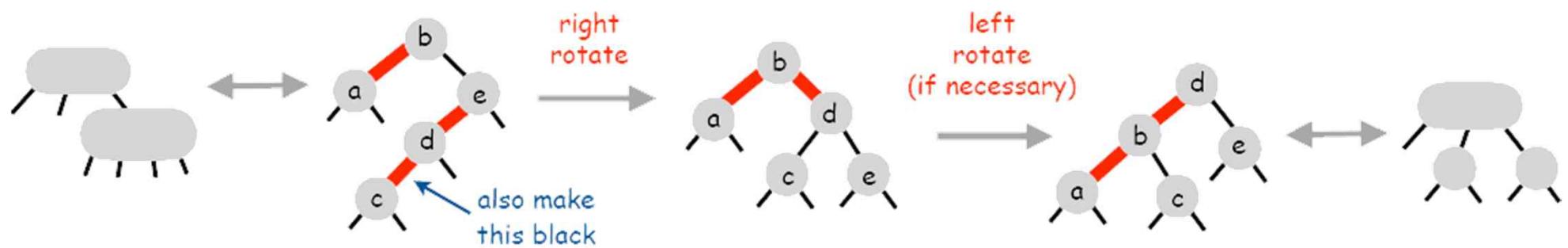
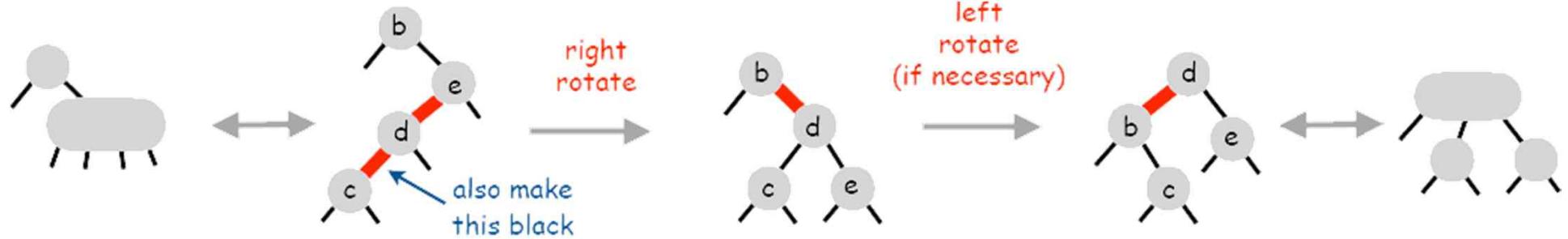
Splitting a 4-node below a 3-node in a left-leaning red-black tree

- Maintain correspondence with 2-3-4 trees



Splitting 4-nodes a left-leaning red-black tree

- The two transformations are the same



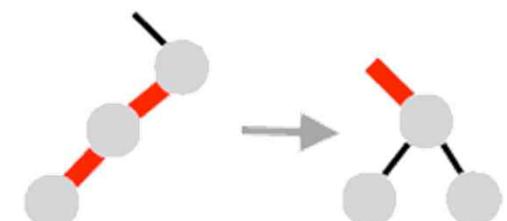
Insert implementation for left-leaning red-black trees (strategy revisited)

- Basic idea: maintain 1-1 correspondence with 2-3-4 trees



- Search as usual

- ◆ if key found reset value, as usual
- ◆ if key not found **insert a new red node at the bottom** [might be right-leaning red link]



- Split 4-nodes on the way DOWN the tree.

- ◆ right-rotate and flip color
- ◆ might leave right-leaning link higher up in the tree

- **NEW TRICK:** enforce left-leaning condition on the way UP the tree.

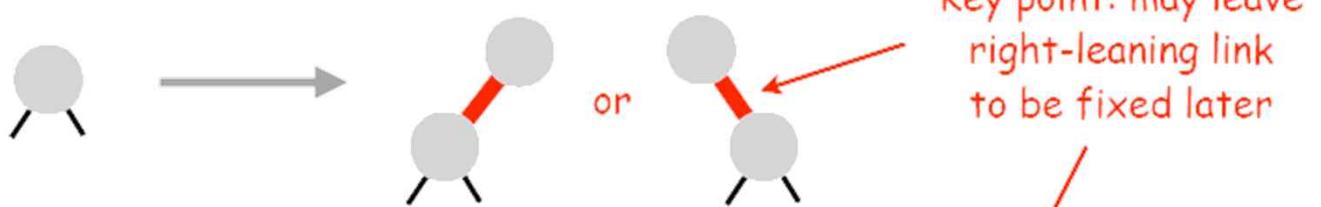
- ◆ left-rotate any right-leaning link on search path
- ◆ trivial with recursion (do it after recursive calls)
- ◆ no other right-leaning links elsewhere



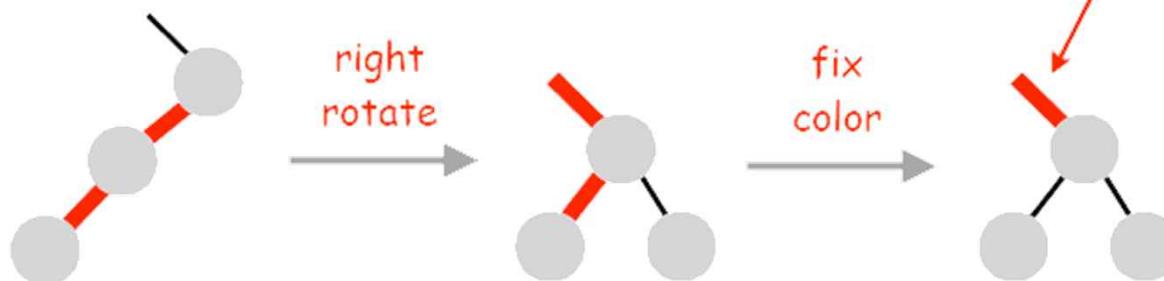
Note: nonrecursive top-down implementation possible, but requires keeping track of great-grandparent on search path (!) and lots of cases.

Insert implementation for left-leaning red-black trees (basic operations)

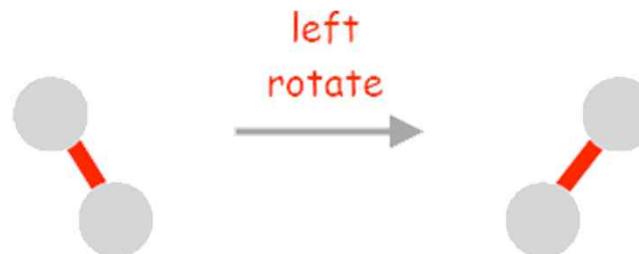
- Insert a new node at bottom



- Split a 4-node



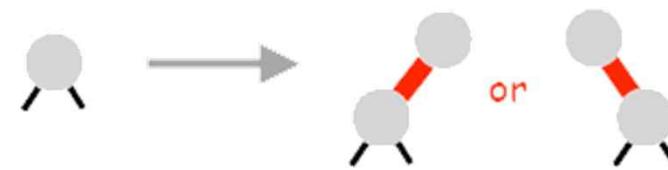
- Enforce left-leaning condition



Insert implementation for left-leaning red-black trees (code for basic operations)

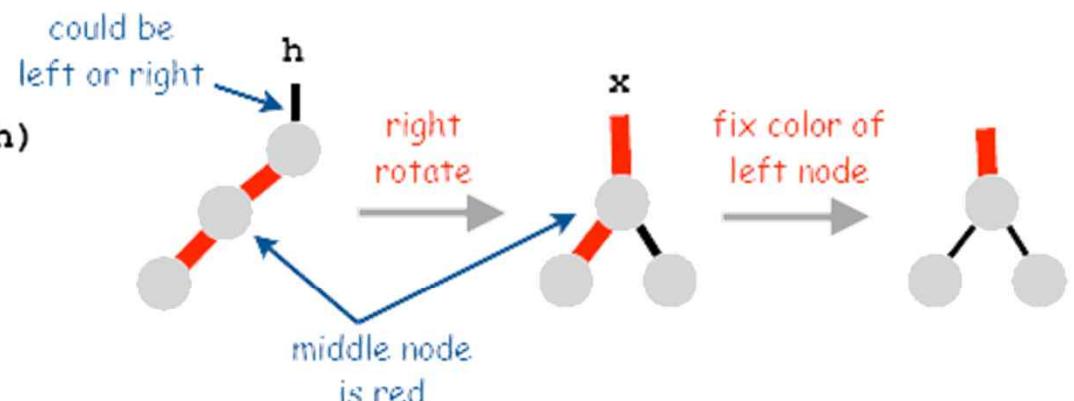
□ Insert a new node at bottom

```
if (h == null)
    return new Node(key, value, RED);
```



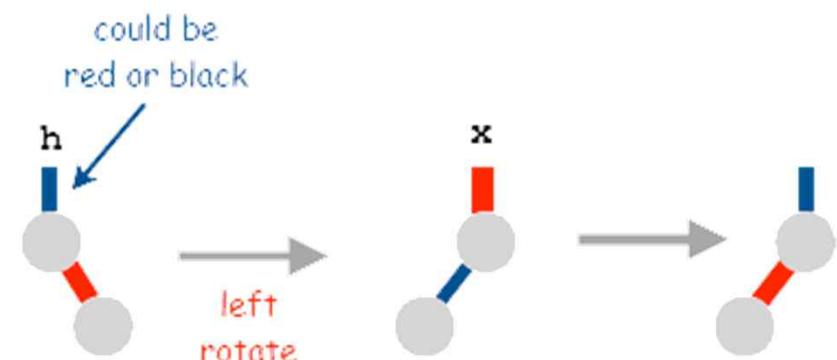
□ Split a 4-node

```
private Node splitFourNode(Node h)
{
    x = rotR(h);
    x.left.color = BLACK;
    return x;
}
```



□ Enforce left-leaning condition

```
private Node leanLeft(Node h)
{
    x = rotL(h);
    x.color      = x.left.color;
    x.left.color = RED;
    return x;
}
```



Insert implementation for left-leaning red-black trees (code)

```
private Node insert(Node h, Key key, Value val)
{
    if (h == null)
        return new Node(key, val, RED);

    if (isRed(h.left))
        if (isRed(h.left.left))
            h = splitFourNode(h);

    int cmp = key.compareTo(h.key);
    if (cmp == 0) h.val = val;
    else if (cmp < 0)
        h.left = insert(h.left, key, val);
    else
        h.right = insert(h.right, key, val);

    if (isRed(h.right))
        h = leanLeft(h);
}

return h;
```

← insert new node at bottom

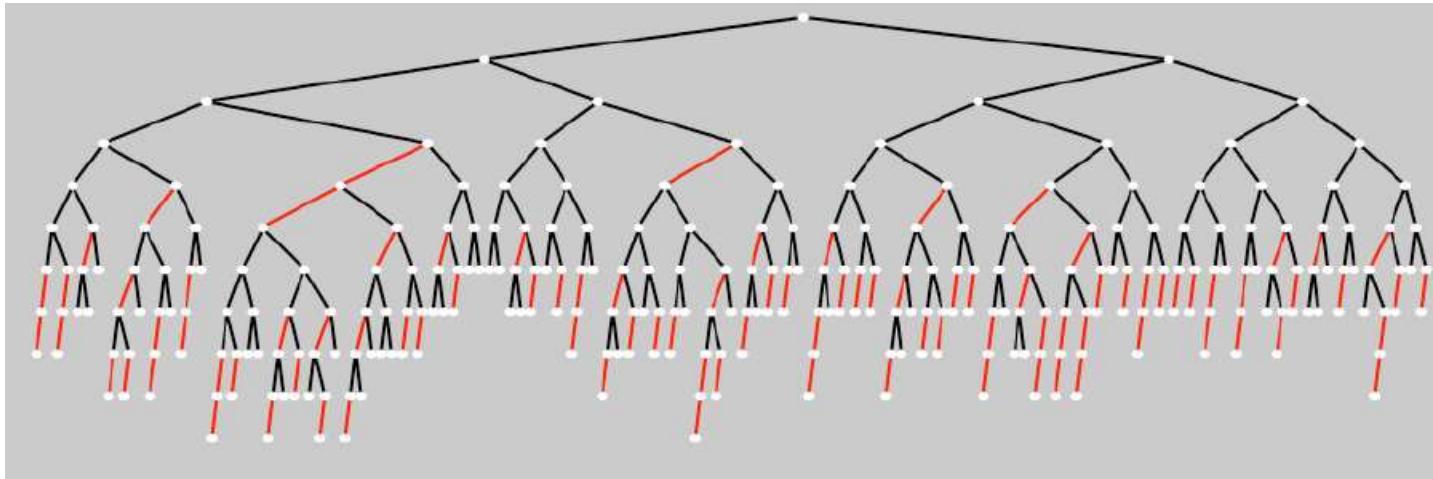
← split 4-nodes on the way down

← search

← enforce left-leaning condition
on the way back up

Balance in left-leaning red-black trees

- Proposition A. Every path from root to leaf has same number of black links.
- Proposition B. Never three red links in-a-row.
- Proposition C. Height of tree is less than $3 \lg N + 2$ in the worst case.



- Property D. Height of tree is $\sim \lg N$ in typical applications.
- Property E. Nearly all 4-nodes are on the bottom in the typical applications.

Why left-leaning trees?

□ Take your pick:

old code (that students had to learn in the past)

```
private Node insert(Node x, Key key, Value val, boolean sw)
{
    if (x == null)
        return new Node(key, value, RED);
    int cmp = key.compareTo(x.key);

    if (isRed(x.left) && isRed(x.right))
    {
        x.color = RED;
        x.left.color = BLACK;
        x.right.color = BLACK;
    }
    if (cmp == 0) x.val = val;
    else if (cmp < 0)
    {
        x.left = insert(x.left, key, val, false);
        if (isRed(x) && isRed(x.left) && sw)
            x = rotR(x);
        if (isRed(x.left) && isRed(x.left.left))
        {
            x = rotR(x);
            x.color = BLACK; x.right.color = RED;
        }
    }
    else // if (cmp > 0)
    {
        x.right = insert(x.right, key, val, true);
        if (isRed(h) && isRed(x.right) && !sw)
            x = rotL(x);
        if (isRed(h.right) && isRed(h.right.right))
        {
            x = rotL(x);
            x.color = BLACK; x.left.color = RED;
        }
    }
    return x;
}
```



new code (that you have to learn)

```
private Node insert(Node h, Key key, Value val)
{
    int cmp = key.compareTo(h.key);
    if (h == null)
        return new Node(key, val, RED);
    if (isRed(h.left))
        if (isRed(h.left.left))
        {
            h = rotR(h);
            h.left.color = BLACK;
        }
    if (cmp == 0) x.val = val;
    else if (cmp < 0)
        h.left = insert(h.left, key, val);
    else
        h.right = insert(h.right, key, val);
    if (isRed(h.right))
    {
        h = rotL(h);
        h.color = h.left.color;
        h.left.color = RED;
    }
    return h;
}
```



↑

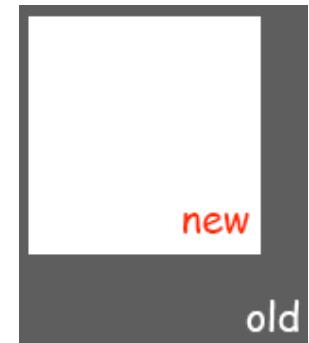
straightforward
(if you've paid attention)

extremely tricky

Why left-leaning trees?

□ Simplified code

- ◆ left-leaning restriction reduces number of cases
- ◆ recursion gives two (easy) chances to fix each node
- ◆ short inner loop



□ Same ideas simplify implementation of other operations

- ◆ delete min
- ◆ delete max
- ◆ delete

□ Built on the shoulders of many, many old balanced tree algorithms

- ◆ AVL trees
- ◆ 2-3 trees
- ◆ 2-3-4 trees
- ◆ skip lists

Bottom line: Left-leaning red-black trees are the **simplest to implement**

↑
and at least as efficient
Algorithms

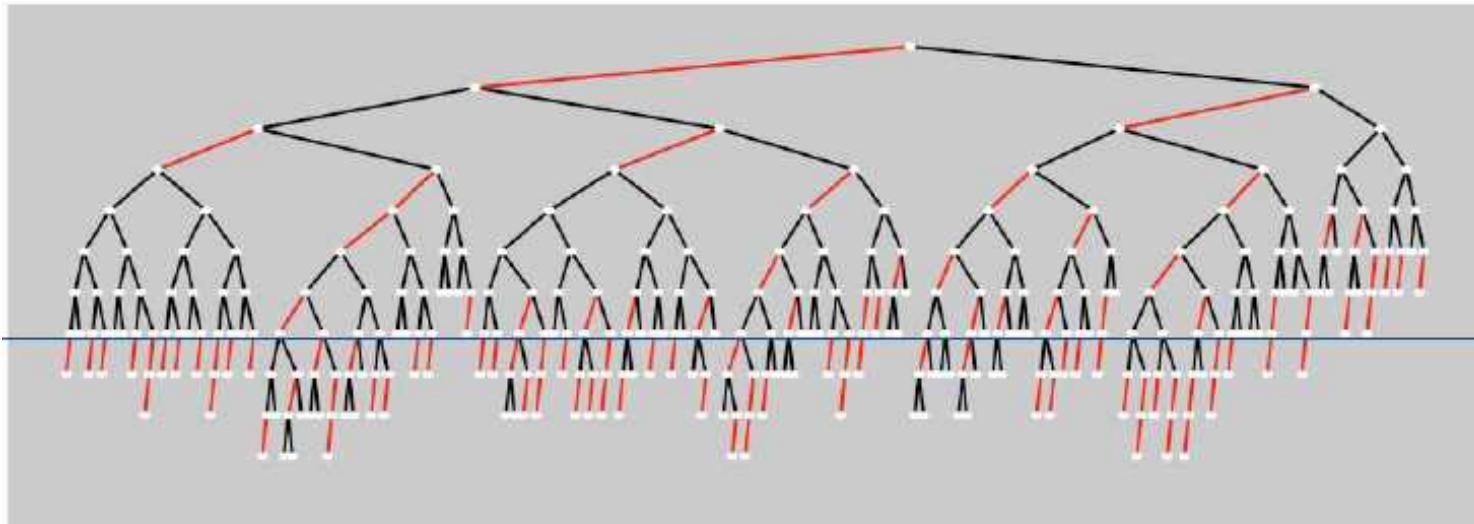
Summary of symbol-table implementations

implementation	guarantee			average case			ordered iteration?
	search	insert	delete	search	insert	delete	
unordered array	N	N	N	N/2	N/2	N/2	no
ordered array	$\lg N$	N	N	$\lg N$	N/2	N/2	yes
unordered list	N	N	N	N/2	N	N/2	no
ordered list	N	N	N	N/2	N/2	N/2	yes
BST	N	N	N	$1.38 \lg N$	$1.38 \lg N$?	yes
randomized BST	$7 \lg N$	$7 \lg N$	$7 \lg N$	$1.38 \lg N$	$1.38 \lg N$	$1.38 \lg N$	yes
2-3-4 tree	$c \lg N$	$c \lg N$		$c \lg N$	$c \lg N$		yes
red-black tree	$3 \lg N$	$3 \lg N$	$3 \lg N$	$\lg N$	$\lg N$	$\lg N$	yes

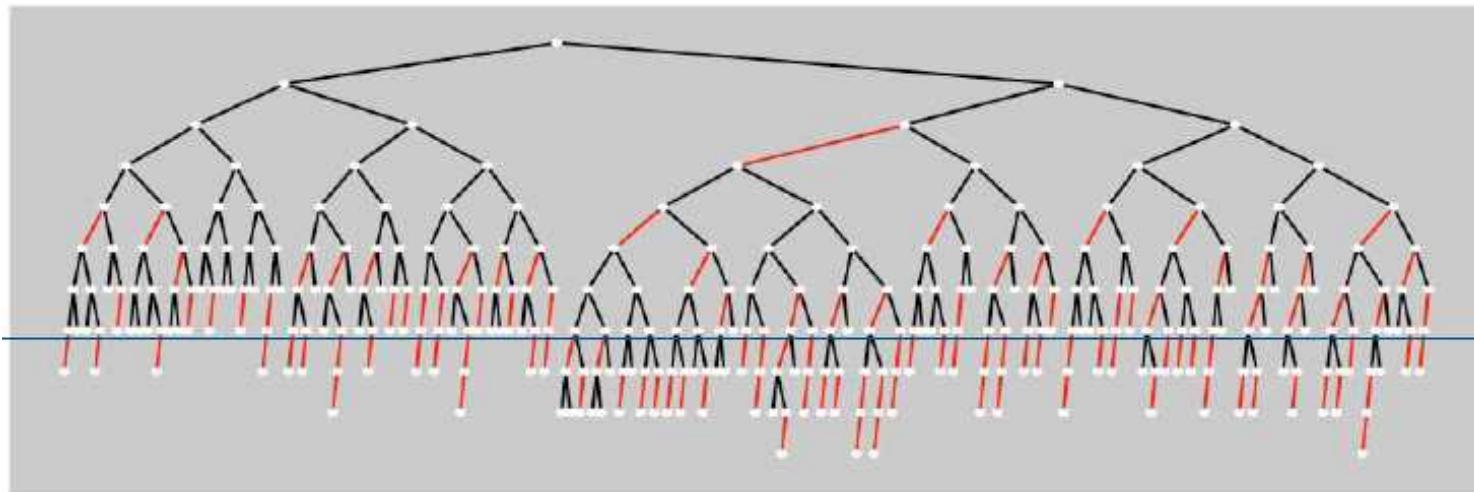
exact value of coefficient unknown
but extremely close to 1

Typical random left-leaning red-black trees

$N = 500$
 $\lg N \approx 9$



average node depth



8. Balanced Trees

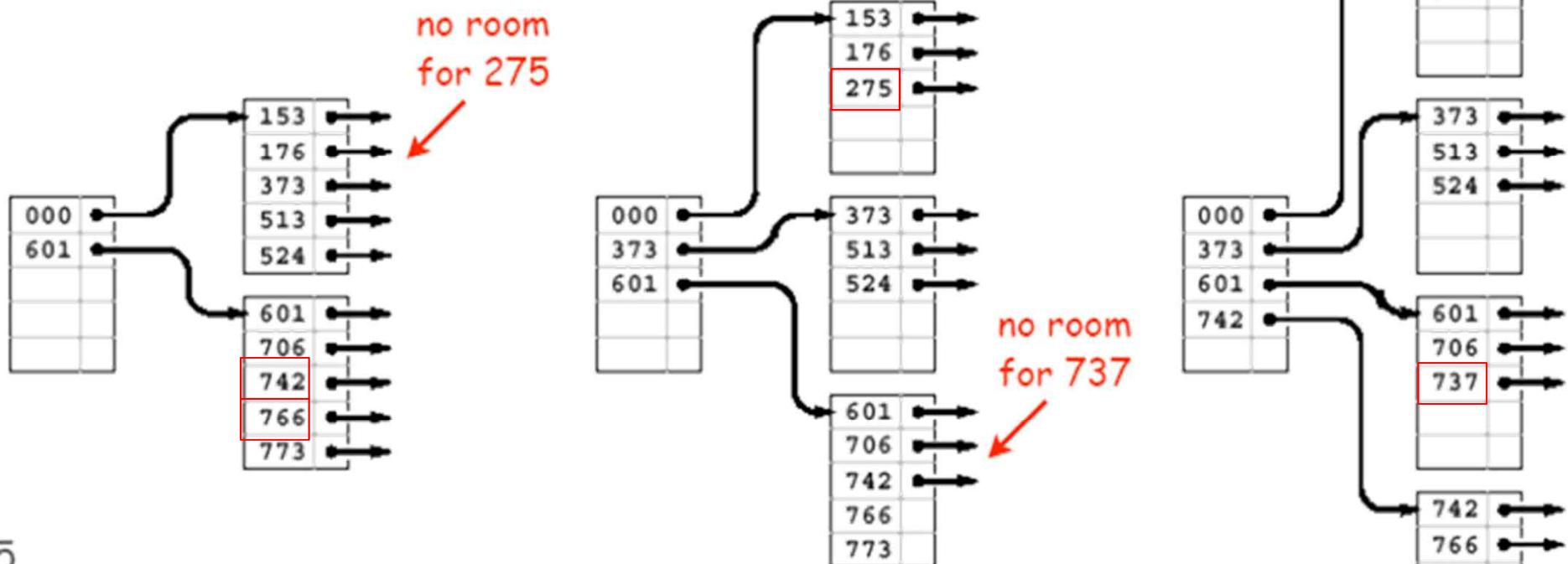
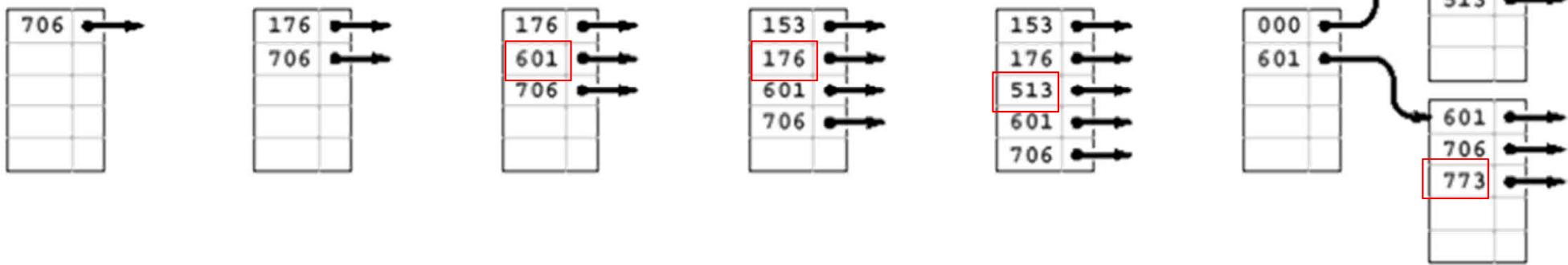
- 2-3-4 trees
- red-black trees
- B-trees

B-trees (Bayer-McCreight, 1972)

- B-Tree. Generalizes 2-3-4 trees by allowing up to M links per node.
- Main application: file systems.
 - ◆ Reading a page into memory from disk is expensive.
 - ◆ Accessing info on a page in memory is free.
 - ◆ Goal: minimize # page accesses.
 - ◆ Node size M = page size.
- Space-time tradeoff.
 - ◆ M large \rightarrow only a few levels in tree.
 - ◆ M small \rightarrow less wasted space.
 - ◆ Typical $M = 1000$, $N < 1$ trillion.
- Bottom line. Number of page accesses is $\log_M N$ per op.

\uparrow
in practice: 3 or 4 (!)

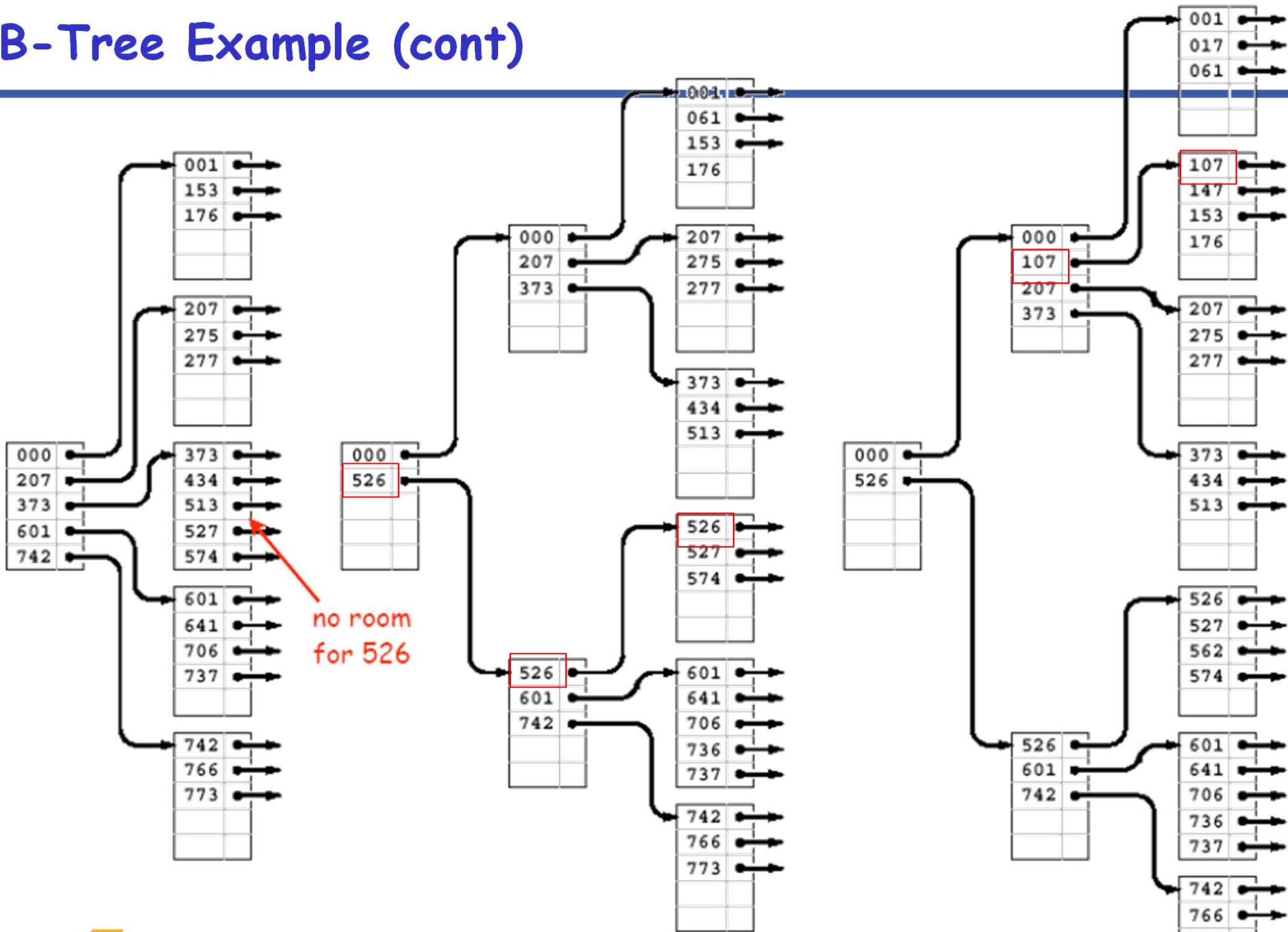
B-Tree Example



$$M = 5$$



B-Tree Example (cont)



Summary of symbol-table implementations

implementation	guarantee			average case			ordered iteration?
	search	insert	delete	search	insert	delete	
unordered array	N	N	N	N/2	N/2	N/2	no
ordered array	$\lg N$	N	N	$\lg N$	N/2	N/2	yes
unordered list	N	N	N	N/2	N	N/2	no
ordered list	N	N	N	N/2	N/2	N/2	yes
BST	N	N	N	$1.44 \lg N$	$1.44 \lg N$?	yes
randomized BST	$7 \lg N$	$7 \lg N$	$7 \lg N$	$1.44 \lg N$	$1.44 \lg N$	$1.44 \lg N$	yes
2-3-4 tree	$c \lg N$	$c \lg N$		$c \lg N$	$c \lg N$		yes
red-black tree	$2 \lg N$	$2 \lg N$	$2 \lg N$	$\lg N$	$\lg N$	$\lg N$	yes
B-tree	1	1	1	1	1	1	yes

- B-Tree. Number of page accesses is $\log_M N$ per op.

Balanced trees in the wild

- Red-black trees: widely used as system symbol tables
 - ◆ Java: java.util.TreeMap, java.util.TreeSet.
 - ◆ C++ STL: map, multimap, multiset.
 - ◆ Linux kernel: linux/rbtree.h.
- B-Trees: widely used for file systems and databases
 - ◆ Windows: HPFS.
 - ◆ Mac: HFS, HFS+.
 - ◆ Linux: ReiserFS, XFS, Ext3FS, JFS.
 - ◆ Databases: ORACLE, DB2, INGRES, SQL, PostgreSQL
- Bottom line: ST implementation with $\lg N$ guarantee for all ops.
 - ◆ Algorithms are variations on a theme: rotations when inserting.
 - ◆ Easiest to implement, optimal, fastest in practice: LLRB trees
 - ◆ Abstraction extends to give search algorithms for huge files: B-trees
- After the break: Can we do better??



Red-black trees in the wild



Common sense. Sixth sense.
Together they're the FBI's newest team.

ACT FOUR

FADE IN:

48 INT. FBI HQ - NIGHT

48

Antonio is at THE COMPUTER as Jess explains herself to Nicole and Pollock. The CONFERENCE TABLE is covered with OPEN REFERENCE BOOKS, TOURIST GUIDES, MAPS and REAMS OF PRINTOUTS.

JESS
It was the red door again.

POLLOCK
I thought the red door was the storage container.

JESS
But it wasn't red anymore. It was black.

ANTONIO
So red turning to black means... what?

POLLOCK
Budget deficits? Red ink, black ink?

NICOLE
Yes. I'm sure that's what it is.
But maybe we should come up with a couple other options, just in case.

Antonio refers to his COMPUTER SCREEN, which is filled with mathematical equations.

Red-black trees in the wild



Common sense. Sixth sense.
Together they're the FBI's newest team.

ACT FOUR

FADE IN:

48 INT. FBI HQ - NIGHT

48

Antonio is at THE COMPUTER as Jess explains herself to Nicole and Pollock. The CONFERENCE TABLE is covered with OPEN REFERENCE BOOKS, TOURIST GUIDES, MAPS and REAMS OF PRINTOUTS.

JESS
It was the red door again.

POLLOCK
I thought the red door was the storage container.

JESS
But it wasn't red anymore. It was black.

ANTONIO
So red turning to black means... what?

POLLOCK
Budget deficits? Red ink, black ink?

NICOLE
Yes. I'm sure that's what it is. But maybe we should come up with a couple other options, just in case.

Antonio refers to his COMPUTER SCREEN, which is filled with mathematical equations.

!!

ANTONIO
It could be an algorithm from a binary search tree. A **red-black tree** tracks every simple path from a node to a descendant leaf with the same number of black nodes.

Red-black trees in the wild



Common sense. Sixth sense.
Together they're the FBI's newest team.

ACT FOUR

FADE IN:

48 INT. FBI HQ - NIGHT

48

Antonio is at THE COMPUTER as Jess explains herself to Nicole and Pollock. The CONFERENCE TABLE is covered with OPEN REFERENCE BOOKS, TOURIST GUIDES, MAPS and REAMS OF PRINTOUTS.

JESS
It was the red door again.

POLLOCK
I thought the red door was the storage container.

JESS
But it wasn't red anymore. It was black.

ANTONIO
So red turning to black means... what?

POLLOCK
Budget deficits? Red ink, black ink?

NICOLE
Yes. I'm sure that's what it is. But maybe we should come up with a couple other options, just in case.

Antonio refers to his COMPUTER SCREEN, which is filled with mathematical equations.

!!

ANTONIO
It could be an algorithm from a binary search tree. A **red-black tree** tracks every simple path from a node to a descendant leaf with the same number of black nodes.

JESS
Does that help you with girls?

Nicole is tapping away at a computer keyboard. She finds something.

Algorithms

Data Structures & Algorithms

9. Hashing

- hash functions
- collision resolution
- applications



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

Summary of symbol-table implementations

implementation	guarantee			average case			ordered iteration?
	search	insert	delete	search	insert	delete	
unordered array	N	N	N	N/2	N/2	N/2	no
ordered array	$\lg N$	N	N	$\lg N$	N/2	N/2	yes
unordered list	N	N	N	N/2	N	N/2	no
ordered list	N	N	N	N/2	N/2	N/2	yes
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$?	yes
randomized BST	$7 \lg N$	$7 \lg N$	$7 \lg N$	$1.39 \lg N$	$1.39 \lg N$	$1.39 \lg N$	yes
red-black tree	$3 \lg N$	$3 \lg N$	$3 \lg N$	$\lg N$	$\lg N$	$\lg N$	yes

Can we do better?

Optimize Judiciously

More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity. - William A. Wulf

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. - Donald E. Knuth

We follow two rules in the matter of optimization:

Rule 1: Don't do it.

Rule 2 (for experts only). Don't do it yet - that is, not until you have a perfectly clear and unoptimized solution.

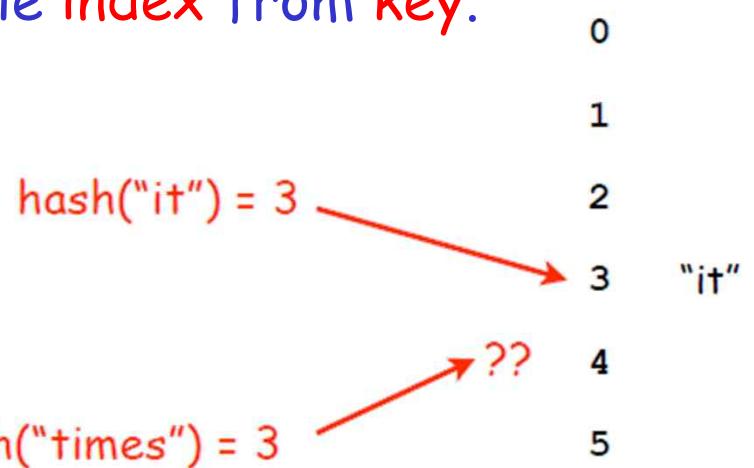
- M. A. Jackson

Reference: Effective Java by Joshua Bloch.



Hashing: basic plan

- Save items in a key-indexed table (index is a function of the key).
- Hash function. Method for computing table **index** from **key**.



- Issues
 - ◆ Computing the hash function
 - ◆ Collision resolution: Algorithm and data structure to handle two keys that hash to the same index.
 - ◆ Equality test: Method for checking whether two keys are equal.
- Classic space-time tradeoff.
 - ◆ No **space** limitation: trivial hash function with key as **address**.
 - ◆ No **time** limitation: trivial collision resolution with **sequential search**.
 - ◆ Limitations on both time and space: **hashing (the real world)**.

9. Hashing

- hash functions
- collision resolution
- applications

Computing the hash function

- Idealistic goal: scramble the keys uniformly
 - ◆ Efficiently computable.
 - ◆ Each table position **equally likely** for each key.


thoroughly researched problem,
still problematic in practical applications
- Practical challenge: need different approach for each **type** of key
- Ex: Social Security numbers.
 - ◆ Bad: first three digits.
 - ◆ Better: last three digits.
- Ex: date of birth.
 - ◆ Bad: birth year.
 - ◆ Better: birthday.
- Ex: phone numbers.
 - ◆ Bad: first three digits.
 - ◆ Better: last three digits.



Hash Codes and Hash Functions

- Java convention: all classes implement `hashCode()`
- `hashCode()` returns a 32-bit int (between -2147483648 and 2147483647)
- **Hash function.** An int between 0 and M-1 (for use as an array index)

First try:

```
String s = "call";
int code = s.hashCode();
int hash = code % M;
```

3045982
7121 8191

- **Bug.** Don't use $(code \% M)$ as array index
- **1-in-a billion bug.** Don't use $(\text{Math.abs}(code) \% M)$ as array index.
- **OK.** Safe to use $((code \& 0xffffffff) \% M)$ as array index.

hex literal 31-bit mask

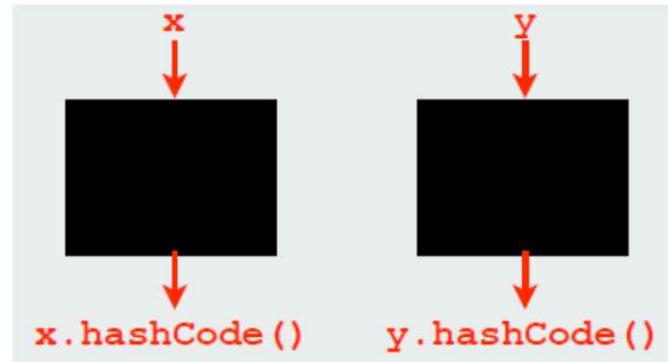
Java's hashCode() convention

□ Theoretical advantages

- ◆ Ensures hashing can be used for every type of object
- ◆ Allows expert implementations suited to each type

□ Requirements:

- ◆ If $x.equals(y)$ then x and y must have the same hash code.
- ◆ Repeated calls to $x.hashCode()$ must return the same value.



□ Practical realities

- ◆ True randomness is hard to achieve
- ◆ Cost is an important consideration

□ Available implementations

- ◆ default (inherited from Object): Memory address of x (!!!)
- ◆ customized Java implementations: String, URL, Integer, Date.
- ◆ User-defined types: users are on their own

← that's you!

A typical type

- Assumption when using hashing in Java:
 - ◆ Key type has reasonable implementation of hashCode() and equals()
- Ex. Phone numbers: (609) 867-5309.

↑
exchange extension

```
public final class PhoneNumber
{
    private final int area, exch, ext;
    public PhoneNumber(int area, int exch, int ext)
    {
        this.area = area;
        this.exch = exch;
        this.ext = ext;
    }
    public boolean equals(Object y) { // as before }
    public int hashCode()
    { return 10007 * (area + 1009 * exch) + ext; }
}
```

← sufficiently
random?

Fundamental problem:

Need a **theorem** for each data type to ensure reliability.

A decent hash code design

- Java 1.5 string library [see also Program 14.2 in Algs in Java].

```
public int hashCode()
{
    int hash = 0;
    for (int i = 0; i < length(); i++)
        hash = s[i] + (31 * hash);
    return hash;
}
```

ith character of s

char	Unicode
...	...
'a'	97
'b'	98
'c'	99
...	...

- Equivalent to $h = 31^{L-1} \cdot s_0 + \dots + 31^2 \cdot s_{L-3} + 31 \cdot s_{L-2} + s_{L-1}$.
- Horner's method to hash string of length L: L multiplies/adds

- Ex

```
String s = "call";
int code = s.hashCode();
```

3045982 = 99 · 31³ + 97 · 31² + 108 · 31¹ + 108 · 31⁰

$$= 108 + 31 \cdot (108 + 31 \cdot (99 + 31 \cdot (97)))$$

Provably random? Well, no.

A poor hash code design

□ Java 1.1 string library.

- ◆ For long strings: only examines 8-9 evenly spaced characters.
- ◆ Saves time in performing arithmetic...

```
public int hashCode()
{
    int hash = 0;
    int skip = Math.max(1, length() / 8);
    for (int i = 0; i < length(); i += skip)
        hash = (37 * hash) + s[i];
    return hash;
}
```

□ but great potential for bad collision patterns.

6 = 52/8

<http://www.cs.princeton.edu/introcs/13loop>Hello.java>
<http://www.cs.princeton.edu/introcs/13loop>Hello.class>
<http://www.cs.princeton.edu/introcs/13loop>Hello.html>
<http://www.cs.princeton.edu/introcs/13loop/index.html>
<http://www.cs.princeton.edu/introcs/12type/index.html>

Basic rule: need to use the whole key.

Digression: using a hash function for data mining

- Use content to characterize documents.
- Applications
 - ◆ Search documents on the web for documents similar to a given one.
 - ◆ Determine whether a new document belongs in one set or another

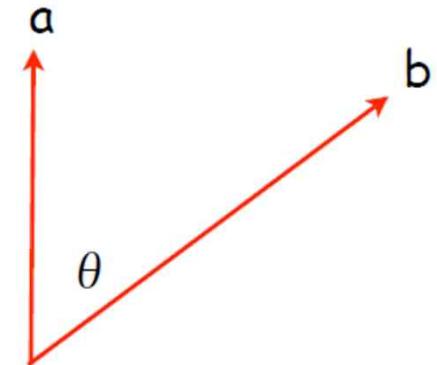
<http://formulas.tutorvista.com/math/angle-between-two-vectors-formula.html>

- Approach
 - ◆ Fix order k and dimension d
 - ◆ Compute $\text{hashCode()} \% d$ for all k -grams in the document
 - ◆ Result: d -dimensional vector profile of each document
 - ◆ To compare documents:

Consider angle θ separating vectors

$\cos \theta$ close to 0: not similar

$\cos \theta$ close to 1: similar



$$\cos \theta = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|}$$

Digression: using a hash function for data mining

```
% more tale.txt  
it was the best of times  
it was the worst of times  
it was the age of wisdom  
it was the age of  
foolishness  
...
```

```
% more genome.txt  
CTTCGGTTGGAAACC  
GAAGCCGCGCGTCT  
TGTCTGCTGCAGC  
ATCGTTTC  
...
```

$$k = 10$$
$$d = 65536$$

tale.txt		genome.txt		
i	10-grams with hashcode() i	freq	10-grams with hashcode() i	freq
0		0		0
1		0		0
2		0		0
435	best of ti foolishnes	2	TTTCGGTTTG TGTCTGCTGC	2
8999	it was the	8		0
...				
12122		0	CTTCGGTTT	3
...				
34543	t was the b	5	ATGCGGTCGA	4
...				
65535				
65536				

profiles

$\cos \theta$ small: **not similar**



Digression: using a hash function to profile a document for data mining

```
public class Document
{
    private String name;
    private double[] profile;
    public Document(String name, int k, int d)
    {
        this.name = name;
        String doc = (new In(name)).readAll();
        int N = doc.length();
        profile = new double[d];
        for (int i = 0; i < N-k; i++)
        {
            int h = doc.substring(i, i+k).hashCode();
            profile[Math.abs(h % d)] += 1;
        }
    }
    public double simTo(Document other)
    {
        // compute dot product and divide by magnitudes
    }
}
```

$$\cos \theta = \frac{a \cdot b}{|a||b|}$$

Digression: using a hash function to compare documents

```
public class CompareAll
{
    public static void main(String args[])
    {
        int k = Integer.parseInt(args[0]);
        int d = Integer.parseInt(args[1]);
        int N = StdIn.readInt();
        Document[] a = new Document[N];
        for (int i = 0; i < N; i++)
            a[i] = new Document(StdIn.readString(), k, d);
        System.out.print("      ");
        for (int j = 0; j < N; j++)
            System.out.printf("      %.4s", a[j].name());
        System.out.println();
        for (int i = 0; i < N; i++)
        {
            System.out.printf("%.4s  ", a[i].name());
            for (int j = 0; j < N; j++)
                System.out.printf("%8.2f", a[i].simTo(a[j]));
            System.out.println();
        }
    }
}
```

Digression: using a hash function to compare documents

Cons	US Constitution
TomS	"Tom Sawyer"
Huck	"Huckleberry Finn"
Prej	"Pride and Prejudice"
Pict	a photograph
DJIA	financial data
Amaz	Amazon.com website .html source
ACTG	genome

```
% java CompareAll 5 1000 < docs.txt
```

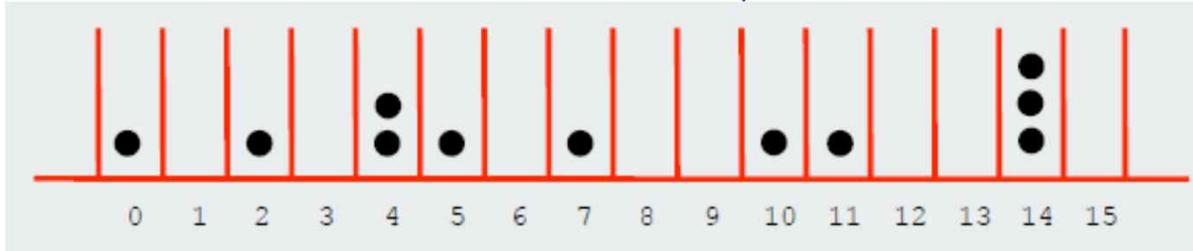
	Cons	TomS	Huck	Prej	Pict	DJIA	Amaz	ACTG
Cons	1.00	0.89	0.87	0.88	0.35	0.70	0.63	0.58
TomS	0.89	1.00	0.98	0.96	0.34	0.75	0.66	0.62
Huck	0.87	0.98	1.00	0.94	0.32	0.74	0.65	0.61
Prej	0.88	0.96	0.94	1.00	0.34	0.76	0.67	0.63
Pict	0.35	0.34	0.32	0.34	1.00	0.29	0.48	0.24
DJIA	0.70	0.75	0.74	0.76	0.29	1.00	0.62	0.58
Amaz	0.63	0.66	0.65	0.67	0.48	0.62	1.00	0.45
ACTG	0.58	0.62	0.61	0.63	0.24	0.58	0.45	1.00

9. Hashing

- hash functions
- collision resolution
- applications

Helpful results from probability theory

- Bins and balls. Throw balls uniformly at random into M bins.



- Birthday problem.

Expect two balls in the same bin after $\sqrt{\pi M / 2}$ tosses.

- Coupon collector.

Expect every bin has ≥ 1 ball after $\Theta(M \ln M)$ tosses.

- Load balancing.

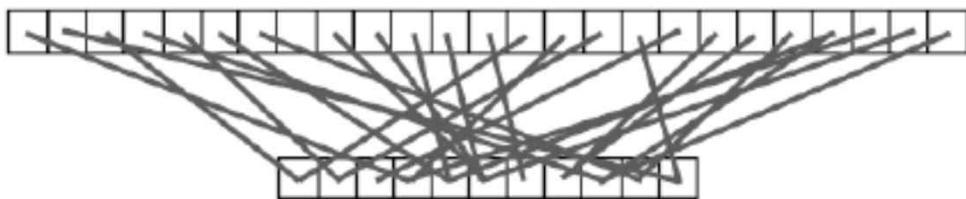
After M tosses, expect most loaded bin has $\Theta(\log M / \log \log M)$ balls.

Collisions

- ❑ Collision. Two distinct keys hashing to same index.
- ❑ Conclusion. Birthday problem → can't avoid collisions unless you have a ridiculous amount of memory.
- ❑ Challenge. Deal with collisions efficiently.

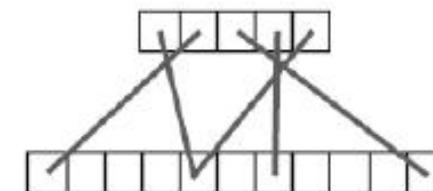
Approach 1:
accept multiple collisions

25 items, 11 table positions
~2 items per table position



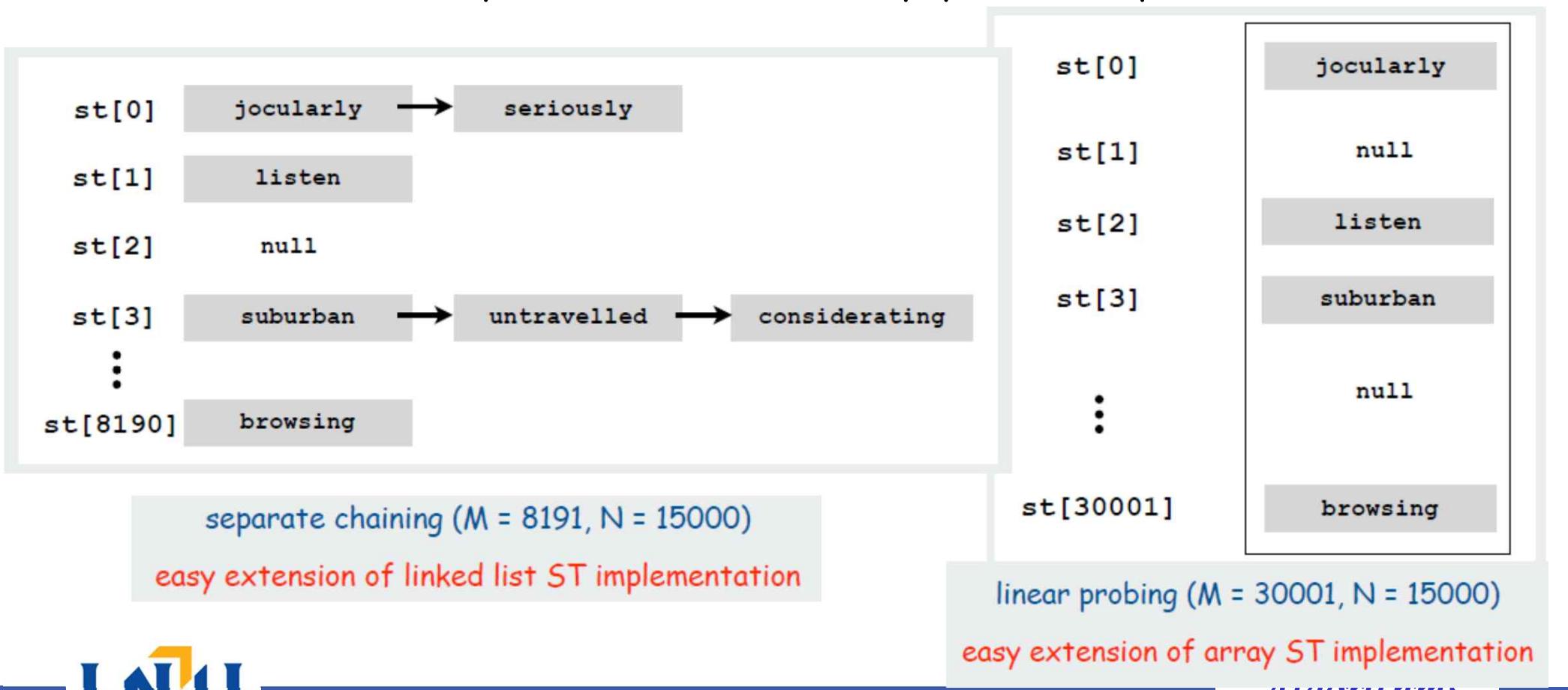
Approach 2:
minimize collisions

5 items, 11 table positions
~ .5 items per table position



Collision resolution: two approaches

- Separate chaining. [H. P. Luhn, IBM 1953]
 - ◆ Put keys that collide in a list associated with index.
- Open addressing. [Amdahl-Boehme-Rochester-Samuel, IBM 1953]
 - ◆ When a new key collides, find next empty slot, and put it there.

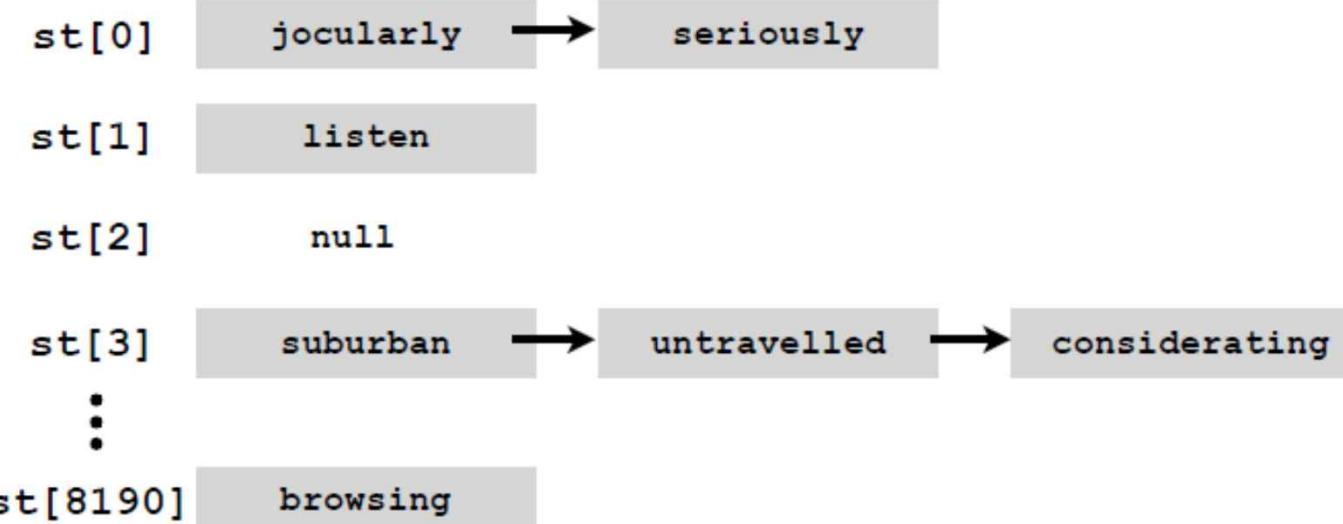


Collision resolution approach 1: separate chaining

- Use an array of $M < N$ linked lists.

← good choice: $M \approx N/10$

- ◆ Hash: map key to integer i between 0 and $M-1$.
- ◆ Insert: put at front of i^{th} chain (if not already there).
- ◆ Search: only need to search i^{th} chain.



key	hash
call	7121
me	3480
ishmael	5017
seriously	0
untravelled	3
suburban	3
...	...

Separate chaining ST implementation (skeleton)

```
public class ListHashST<Key, Value>
{
    private int M = 8191;
    private Node[] st = new Node[M];

    private class Node
    {
        Object key;
        Object val;
        Node next;
        Node(Key key, Value val, Node next)
        {
            this.key    = key;
            this.val    = val;
            this.next   = next;
        }
    }

    private int hash(Key key)
    {   return (key.hashCode() & 0x7fffffff) % M;   }

    public void put(Key key, Value val)
    // see next slide

    public Val get(Key key)
    // see next slide
}
```

could use
doubling

no generics in
arrays in Java

compare with
linked lists

Separate chaining ST implementation (put and get)

```
public void put(Key key, Value val)
{
    int i = hash(key);
    for (Node x = st[i]; x != null; x = x.next)
        if (key.equals(x.key))
            { x.val = val; return; }
    st[i] = new Node(key, value, first);
}

public Value get(Key key)
{
    int i = hash(key);
    for (Node x = st[i]; x != null; x = x.next)
        if (key.equals(x.key))
            return (Value) x.val;
    return null;
}
```

Identical to linked-list code, except hash to pick a list.

Analysis of separate chaining

□ Separate chaining performance.

- ◆ Cost is proportional to length of list.
- ◆ Average length = N / M .
- ◆ Worst case: all keys hash to same list.

Theorem. Let $\alpha = N / M > 1$ be average length of list. For any $t > 1$, probability that list length $> t \alpha$ is exponentially small in t .

depends on hash map being random map

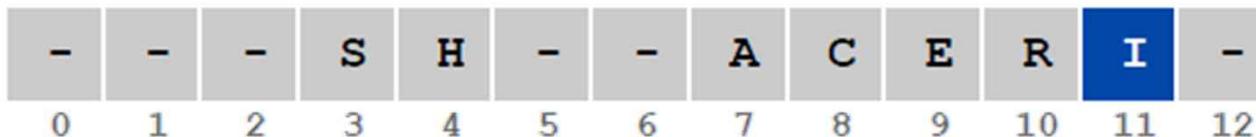
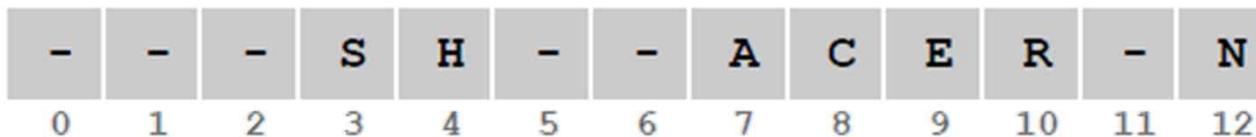


□ Parameters.

- ◆ M too large \Rightarrow too many empty chains.
- ◆ M too small \Rightarrow chains too long.
- ◆ Typical choice: $\alpha = N / M \approx 10 \Rightarrow$ constant-time ops.

Collision resolution approach 2: open addressing

- Use an array of size $M \gg N$.
 - ◆ Hash: map key to integer i between 0 and $M-1$.
- Linear probing:
 - ◆ Insert: put in slot i if free; if not try $i+1, i+2, \dots$, etc.
 - ◆ Search: search slot i ; if occupied but no match, try $i+1, i+2, \dots$, etc.



insert I
hash(I) = 11



insert N
hash(N) = 8

Linear probing ST implementation

```
public class ArrayHashST<Key, Value>
{
    private int M = 30001;           ← standard ugly casts
    private Value[] vals = (Value[]) new Object[M];
    private Key[] keys = (Key[]) new Object[M];

    private int hash(Key key) // as before

    public void put(Key key, Value val)
    {
        int i;
        for (i = hash(key); keys[i] != null; i = (i+1) % M)
            if (key.equals(keys[i]))
                break;
        vals[i] = val;
        keys[i] = key;
    }

    public Value get(Key key)
    {
        for (int i = hash(key); keys[i] != null; i = (i+1) % M)
            if (key.equals(keys[i]))
                return vals[i];
        return null;
    }
}
```

compare with
elementary
unordered array
implementation

standard
array doubling
code omitted
(double when
half full)

Clustering

- Cluster. A contiguous block of items.
- Observation. New keys likely to hash into middle of big clusters.



cluster

- Knuth's parking problem. Cars arrive at one-way street with M parking spaces. Each desires a random space i : if space i is taken, try $i+1, i+2, \dots$. What is mean displacement of a car?



- Empty. With $M/2$ cars, mean displacement is about $3/2$.
- Full. Mean displacement for the last car is about $\sqrt{\pi M / 2}$

Analysis of linear probing

□ Linear probing performance.

- ◆ Insert and search cost depend on length of cluster.
- ◆ Average length of cluster = $\alpha = N / M$.
- ◆ Worst case: all keys hash to same cluster.

□ Theorem. [Knuth 1962] Let $\alpha = N / M < 1$ be the load factor.

Average probes for insert/search miss

$$\frac{1}{2} \left(1 + \frac{1}{(1 - \alpha)^2} \right) = (1 + \alpha + 2\alpha^2 + 3\alpha^3 + 4\alpha^4 + \dots) / 2$$

Average probes for search hit

$$\frac{1}{2} \left(1 + \frac{1}{(1 - \alpha)} \right) = 1 + (\alpha + \alpha^2 + \alpha^3 + \alpha^4 + \dots) / 2$$

□ Parameters.

- ◆ Load factor too small \Rightarrow too many empty array entries.
- ◆ Load factor too large \Rightarrow clusters coalesce.
- ◆ Typical choice: $M \approx 2N \Rightarrow$ constant-time ops.

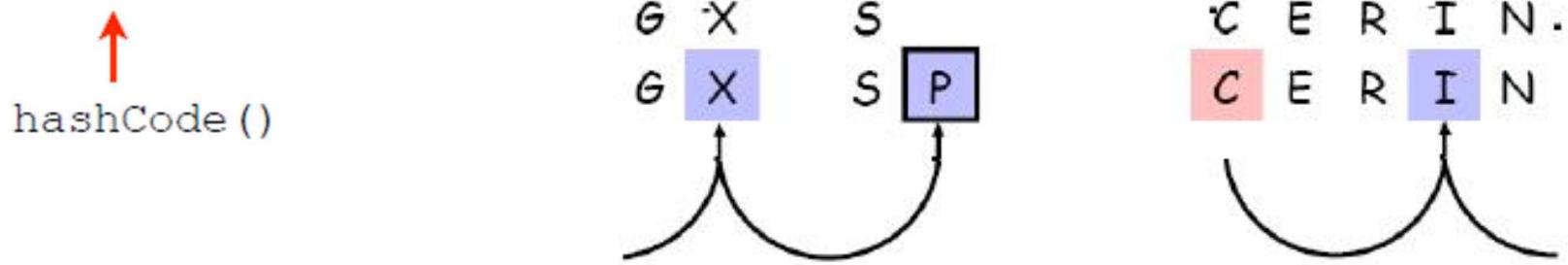
Hashing: variations on the theme

- Many improved versions have been studied:
- Ex: Two-probe hashing
 - ◆ hash to two positions, put key in shorter of the two lists
 - ◆ reduces average length of the longest list to $\log \log N$
- Ex: Double hashing
 - ◆ use linear probing, but skip a variable amount, not just 1 each time
 - ◆ effectively eliminates clustering
 - ◆ can allow table to become nearly full



Double hashing

- Idea Avoid clustering by using second hash to compute skip for search.
- Hash. Map key to integer i between 0 and $M-1$.
- Second hash. Map key to nonzero skip value k .
- Ex: $k = 1 + (v \bmod 97)$.



- Effect. Skip values give different search paths for keys that collide.
- Best practices. Make k and M relatively prime.

Double Hashing Performance

- Theorem. [Guibas-Szemerédi] Let $\alpha = N / M < 1$ be average length of list.

Average probes for insert/search miss

$$\frac{1}{(1 - \alpha)} = 1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \dots$$

Average probes for search hit

$$\frac{1}{\alpha} \ln \frac{1}{(1 - \alpha)} = 1 + \alpha/2 + \alpha^2/3 + \alpha^3/4 + \alpha^4/5 + \dots$$

- Parameters. Typical choice: $\alpha \approx 1.2 \Rightarrow$ constant-time ops.
- Disadvantage. Delete cumbersome to implement.

Hashing Tradeoffs

- Separate chaining vs. linear probing/double hashing.
 - ◆ Space for links vs. empty table slots.
 - ◆ Small table + linked allocation vs. big coherent array.
- Linear probing vs. double hashing.

		load factor α			
		50%	66%	75%	90%
linear probing	get	1.5	2.0	3.0	5.5
	put	2.5	5.0	8.5	55.5
double hashing	get	1.4	1.6	1.8	2.6
	put	1.5	2.0	3.0	5.5

number of probes

Summary of symbol-table implementations

implementation	guarantee			average case			ordered iteration?	operations on keys
	search	insert	delete	search	insert	delete		
unordered array	N	N	N	N/2	N/2	N/2	no	<code>equals()</code>
ordered array	$\lg N$	N	N	$\lg N$	N/2	N/2	yes	<code>compareTo()</code>
unordered list	N	N	N	N/2	N	N/2	no	<code>equals()</code>
ordered list	N	N	N	N/2	N/2	N/2	yes	<code>compareTo()</code>
BST	N	N	N	$1.38 \lg N$	$1.38 \lg N$?	yes	<code>compareTo()</code>
randomized BST	$7 \lg N$	$7 \lg N$	$7 \lg N$	$1.38 \lg N$	$1.38 \lg N$	$1.38 \lg N$	yes	<code>compareTo()</code>
red-black tree	$2 \lg N$	$2 \lg N$	$2 \lg N$	$\lg N$	$\lg N$	$\lg N$	yes	<code>compareTo()</code>
hashing	1*	1*	1*	1*	1*	1*	no	<code>equals()</code> <code>hashCode()</code>

* assumes random hash code



Hashing versus balanced trees

- Hashing
 - ◆ simpler to code
 - ◆ no effective alternative for unordered keys
 - ◆ faster for simple keys (a few arithmetic ops versus $\lg N$ compares)
 - ◆ (Java) better system support for strings [cached hashCode]
 - ◆ does your hash function produce random values for your key type??
- Balanced trees
 - ◆ stronger performance guarantee
 - ◆ can support many more operations for ordered keys
 - ◆ easier to implement compareTo() correctly than equals() and hashCode()
- Java system includes both
 - ◆ red-black trees: java.util.TreeMap, java.util.TreeSet
 - ◆ hashing: java.util.HashMap, java.util.IdentityHashMap

Typical “full” ST API

public class *ST<Key extends Comparable<Key>, Value>	
*ST()	create a symbol table
void put(Key key, Value val)	put key-value pair into the table
Value get(Key key)	return value paired with key (null if key is not in table)
boolean contains(Key key)	is there a value paired with key?
Key min()	smallest key
Key max()	largest key
Key next(Key key)	next largest key (null if key is max)
Key prev(Key key)	next smallest key (null if key is min)
void remove(Key key)	remove key-value pair from table
Iterator<Key> iterator()	iterator through keys in table

Hashing is **not** suitable for implementing such an API (no order)

BSTs are **easy** to extend to support such an API (basic tree ops)

Ex: Can use LLRB trees implement priority queues for distinct keys

9. Hashing

- hash functions
- collision resolution
- applications

Set ADT (Abstract Data Type)

- Set. Collection of distinct keys.

```
public class *SET<Key extends Comparable<Key>, Value>
```

SET()	create a set
void add(Key key)	put key into the set
boolean contains(Key key)	is there a value paired with key?
void remove(Key key)	remove key from the set
Iterator<Key> iterator()	iterator through all keys in the set

- Normal mathematical assumption: collection is unordered
- Typical (eventual) client expectation: ordered iteration

Q. How to implement?

A0. Hashing (our ST code [value removed] or `java.util.HashSet`)

A1. Red-black BST (our ST code [value removed] or `java.util.TreeSet`)

unordered iterator
 $O(1)$ search

ordered iterator
 $O(\log N)$ search

SET client example 1: dedup filter

No iterator needed.
Output is in same order
as input with
dups removed.

- Remove duplicates from strings in standard input

- ◆ Read a key.
- ◆ If key is not in set, insert and print it.

```
public class DeDup
{
    public static void main(String[] args)
    {
        SET<String> set = new SET<String>();
        while (!StdIn.isEmpty())
        {
            String key = StdIn.readString();
            if (!set.contains(key))
            {
                set.add(key);
                StdOut.println(key);
            }
        }
    }
}
```

```
% more tale.txt
it was the best of times
it was the worst of times
it was the age of wisdom
it was the age of
foolishness
...
```

```
% java Dedup < tale.txt
it
was
the
best
of
times
worst
age
wisdom
foolishness
...
```

Simplified version of FrequencyCount (no iterator needed)

Algorithms

SET client example 2A: lookup filter

- Print words from standard input that are found in a list
 - ◆ Read in a list of words from one file.
 - ◆ Print out all words from standard input that are in the list.

```
public class LookupFilter
{
    public static void main(String[] args)
    {
        SET<String> set = new SET<String>();

        In in = new In(args[0]);
        while (!in.isEmpty())
            set.add(in.readString());

        while (!StdIn.isEmpty())
        {
            String word = StdIn.readString();
            if (set.contains(word))
                StdOut.println(word);
        }
    }
}
```

← create SET
← process list
← print words that
are ~~not~~ in list

SET client example 2B: exception filter

- Print words from standard input that are not found in a list
 - ◆ Read in a list of words from one file.
 - ◆ Print out all words from standard input that are not in the list.

```
public class LookupFilter
{
    public static void main(String[] args)
    {
        SET<String> set = new SET<String>();

        In in = new In(args[0]);
        while (!in.isEmpty())
            set.add(in.readString());

        while (!StdIn.isEmpty())
        {
            String word = StdIn.readString();
            if (!set.contains(word))
                StdOut.println(word);
        }
    }
}
```

← create SET
← process list
← print words that
are not in list

SET filter applications

application	purpose	key	type	in list	not in list
dedup	eliminate duplicates		dedup	duplicates	unique keys
spell checker	find misspelled words	word	exception	dictionary	misspelled words
browser	mark visited pages	URL	lookup	visited pages	
chess	detect draw	board	lookup	positions	
spam filter	eliminate spam	IP addr	exception	spam	good mail
trusty filter	allow trusted mail	URL	lookup	good mail	
credit cards	check for stolen cards	number	exception	stolen cards	good cards

Searching challenge:

Problem: Index for a PC or the web

Assumptions: 1 billion++ words to index

Which searching method to use?

- 1) hashing implementation of SET
- 2) hashing implementation of ST
- 3) red-black-tree implementation of ST
- 4) red-black-tree implementation of SET
- 5) doesn't matter much

The screenshot shows a 'Spotlight' search interface with the query 'searching challenge'. The results are organized into sections: 'Top Hit' (10Hashing), 'Documents' (moby dick.txt, movies.txt, Papers/Abstracts, score.card.txt, Requests), 'Mail Messages' (Re: Draft of lecture on symb..., SODA 07 Final Accepts, SODA 07 Summary, Got-it, No Subject), 'PDF Documents' (08BinarySearchTrees.pdf, 07SymbolTables.pdf, 07SymbolTables.pdf, 06PriorityQueues.pdf, 06PriorityQueues.pdf), and 'Presentations' (10Hashing, 07SymbolTables, 06PriorityQueues).



Index for search in a PC

```
ST<String, SET<File>> st = new ST<String, SET<File>>();  
for (File f: filesystem)  
{  
    In in = new In(f);  
    String[] words = in.readAll().split("\\s+"); ← build index  
    for (int i = 0; i < words.length; i++)  
    {  
        String s = words[i];  
        if (!st.contains(s))  
            st.put(s, new SET<File>());  
        SET<File> files = st.get(s);  
        files.add(f);  
    }  
}  
  
SET<File> files = st.get(s); ← process  
for (File f: files) ... ← lookup  
                           request
```

Searching challenge:

Problem: Index for a book

Assumptions: book has 100,000+ words

Which searching method to use?

- 1) hashing implementation of SET
- 2) hashing implementation of ST
- 3) red-black-tree implementation of ST
- 4) red-black-tree implementation of SET
- 5) doesn't matter much

Index

Abstract data type (ADT), 127-195
abstract classes, 163
classes, 129-136
collections of items, 137-139
creating, 137-164
defined, 128
duplicate items, 173-176
equivalence-relations, 159-162
FIFO queues, 165-171
first-class, 177-186
generic operations, 273
index items, 177
insert/remove operations, 138-139
modular programming, 135
polynomial, 188-192
priority queues, 375-376
pushdown stack, 138-156
subx, 135
symbol table, 497-506
ADT interfaces
array (*myArray*), 274
complex number (*Complex*), 181
existence table (*ET*), 663
full priority queue (*PQfull*), 397
indirect priority queue (*PQ1*), 403
item (*myItem*), 273, 498
key (*myKey*), 498
polynomial (*Poly*), 189
point (*Point*), 134
priority queue (*PQ*), 375
queue of int (*intQueue*), 166
stack of int (*intStack*), 140
symbol table (ST), 503
text index (TI), 525
union-find (UF), 159
Abstract in-place merging, 351-353
Abstract operation, 10
Access control state, 131
Actual data, 31
Adapter class, 135-157
Adaptive sort, 268
Address, 84-85
Adjacency list, 120-123
 depth-first search, 251-256
Adjacency matrix, 120-122
Ajtai, M., 464
Algorithm, 4-6, 27-64
 abstract operations, 10, 31, 34-35
 analysis of, 6
 average-worst-case performance, 35, 60-62
 big-Oh notation, 44-47
 binary search, 56-59
 computational complexity, 62-64
 efficiency, 6, 30, 32
 empirical analysis, 30-32, 58
 exponential-time, 219
 implementation, 28-30
 logarithm function, 40-43
 mathematical analysis, 33-36, 58
 primary parameter, 36
 probabilistic, 351
 recurrences, 49-52, 57
 recursive, 198
 running time, 34-40
 search, 53-56, 498
 steps in, 22-23
 See also Randomized algorithm
Amortization approach, 557, 627
Arithmetic operator, 177-179, 188, 191
Array, 12, 83
 binary search, 57
 dynamic allocation, 87
and linked lists, 92, 94-95
merging, 349-350
multidimensional, 117-118
references, 86-87, 89
sorting, 265-267, 273-276
and strings, 119
two-dimensional, 117-118, 120-124
vectors, 87
visualizations, 295
See also Index, array
Array representation
 binary tree, 381
 FIFO queue, 168-169
 linked lists, 119
 polynomial ADT, 191-192
 priority queue, 377-378, 403, 406
 pushdown stack, 148-150
 random queue, 170
 symbol table, 508, 511-512, 521
Asymptotic expression, 45-46
Average deviation, 80-81
Average-case performance, 35, 60-61
AVL tree, 583
B tree, 584, 692-704
extreme/internal pages, 695
4-5-6-7-8 tree, 693-704
Markov chain, 701
remove, 701-703
search/insert, 697-701
select/kort, 701
Balanced tree, 238, 555-558
B tree, 584
bottom-up, 576, 584-585
height-balanced, 583
indexed sequential access, 690-692
performance, 575-576, 581-582, 595-598
randomized, 559-564
red-black, 577-585
skip lists, 587-594
splay, 566-571

727



Index for a book

```
public class Index
{
    public static void main(String[] args)
    {
        String[] words = StdIn.readAll().split("\s+");
        ST<String, SET<Integer>> st;
        st = new ST<String, SET<Integer>>();

        for (int i = 0; i < words.length; i++)
        {
            String s = words[i];
            if (!st.contains(s))
                st.put(s, new SET<Integer>());
            SET<Integer> pages = st.get(s);
            pages.add(page(i));
        }

        for (String s : st)
            StdOut.println(s + ": " + st.get(s));
    }
}
```

read book and create ST

process all words

print index!



Hashing in the wild: Java implementations

- Java has built-in libraries for hash tables.
 - ◆ `java.util.HashMap` = separate chaining implementation.
 - ◆ `java.util.IdentityHashMap` = linear probing implementation.

```
import java.util.HashMap;
public class HashMapDemo
{
    public static void main(String[] args)
    {
        HashMap<String, String> st = new HashMap <String, String>();
        st.put("www.cs.princeton.edu", "128.112.136.11");
        st.put("www.princeton.edu",      "128.112.128.15");
        StdOut.println(st.get("www.cs.princeton.edu"));
    }
}
```

- Null value policy.
 - ◆ Java `HashMap` allows null values.
 - ◆ Our implementation forbids null values.

Using HashMap

- Implementation of our API with `java.util.HashMap`.

```
import java.util.HashMap;
import java.util.Iterator;

public class ST<Key, Value> implements Iterable<Key>
{
    private HashMap<Key, Value> st = new HashMap<Key, Value>();

    public void put(Key key, Value val)
    {
        if (val == null) st.remove(key);
        else                st.put(key, val);
    }

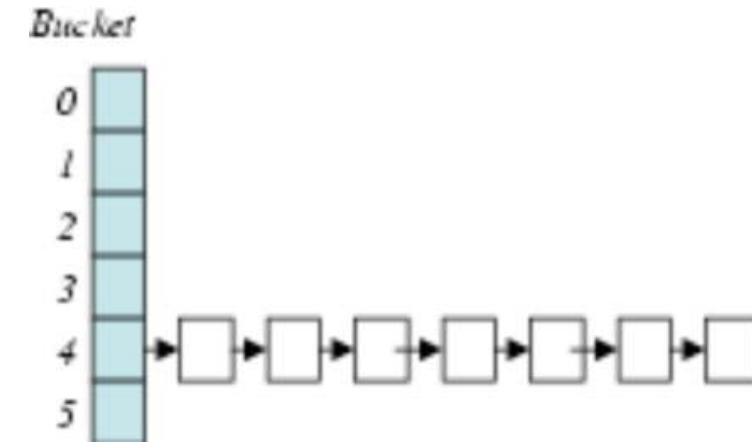
    public Value get(Key key)          { return st.get(key); }
    public Value remove(Key key)       { return st.remove(key); }
    public boolean contains(Key key)   { return st.contains(key); }
    public int size()                 { return st.size(); }
    public Iterator<Key> iterator()   { return st.keySet().iterator(); }
}
```

Hashing in the wild: algorithmic complexity attacks

- Is the random hash map assumption important in practice?
 - ◆ Obvious situations: aircraft control, nuclear reactor, pacemaker.
 - ◆ Surprising situations: **denial-of-service** attacks.



malicious adversary learns **your** ad hoc hash function
(e.g., by reading Java API) and causes a big pile-up in
single address that grinds performance to a halt



- Real-world exploits. [Crosby-Wallach 2003]

- ◆ Bro server: send carefully chosen packets to DOS the server, using less bandwidth than a dial-up modem
- ◆ Perl 5.8.0: insert carefully chosen strings into associative array.
- ◆ Linux 2.4.20 kernel: save files with carefully chosen names.

Reference: <http://www.cs.rice.edu/~scrosby/hash>

Algorithmic complexity attack on the Java Library

- Goal. Find strings with the same hash code.
- Solution. The base-31 hash code is part of Java's string API.

Key	hashCode()	Key	hashCode()
Aa	2112	AaAaAaAa	-540425984
BB	2112	AaAaAaBB	-540425984
		AaAaBBAa	-540425984
		AaAaBBBB	-540425984
		AaBBAaAa	-540425984
		AaBBAaBB	-540425984
		AaBBBBBa	-540425984
		AaBBBBBB	-540425984
		BBAaAaAa	-540425984
		BBAaAaBB	-540425984
		BBAaBBAa	-540425984
		BBAaBBBB	-540425984
		BBBBAaAa	-540425984
		BBBBAaBB	-540425984
		BBBBBBBa	-540425984
		BBBBBBBB	-540425984

Does your hash function produce **random** values for your key type??

2^N strings of length $2N$ that hash to same value!

One-Way Hash Functions

□ One-way hash function.

- ◆ Hard to find a key that will hash to a desired value, or to find two keys that hash to same value.

□ Ex.

MD4, MD5, SHA-0, SHA-1, SHA-2, WHIRLPOOL, RIPEMD-160.

insecure

```
String password = args[0];
MessageDigest sha1 = MessageDigest.getInstance("SHA1");
byte[] bytes = sha1.digest(password);

// prints bytes as hex string
```

□ Applications.

- ◆ Digital fingerprint, message digest, storing passwords.
- ◆ Too expensive for use in ST implementations (use balanced trees)

Data Structures & Algorithms

10. Undirected Graphs

- Graph API
- maze exploration
- depth-first search
- breadth-first search
- connected components
- challenges



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

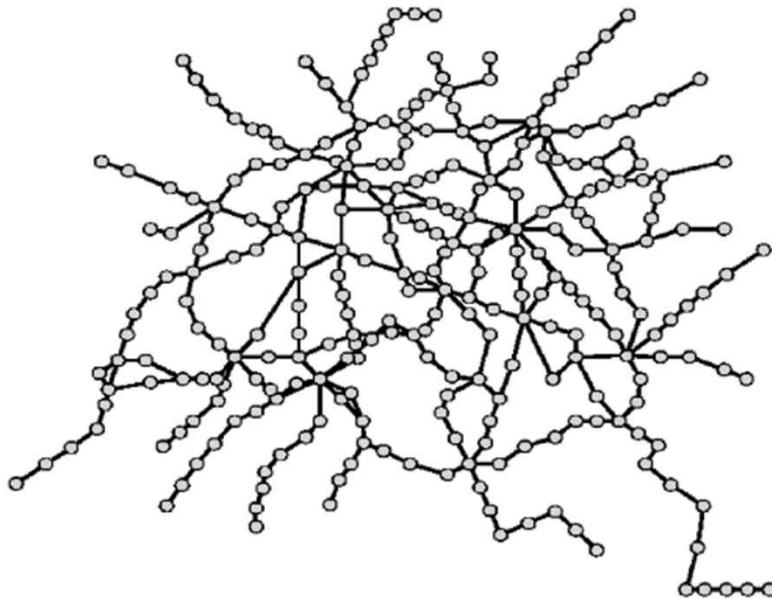
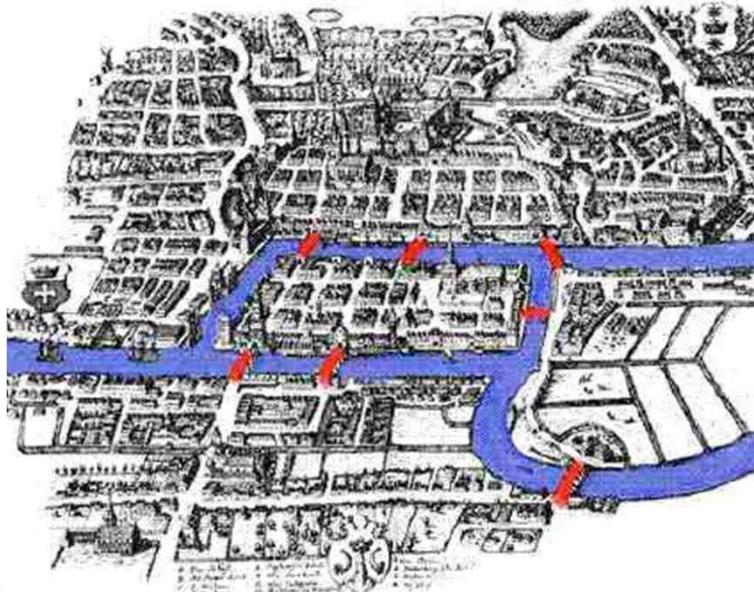
Undirected graphs

□ Graph

- ◆ Set of **vertices** connected pairwise by **edges**.

□ Why study graph algorithms?

- ◆ Interesting and broadly useful **abstraction**.
- ◆ Challenging branch of computer science and discrete math.
- ◆ Hundreds of graph **algorithms** known.
- ◆ Thousands of practical **applications**.

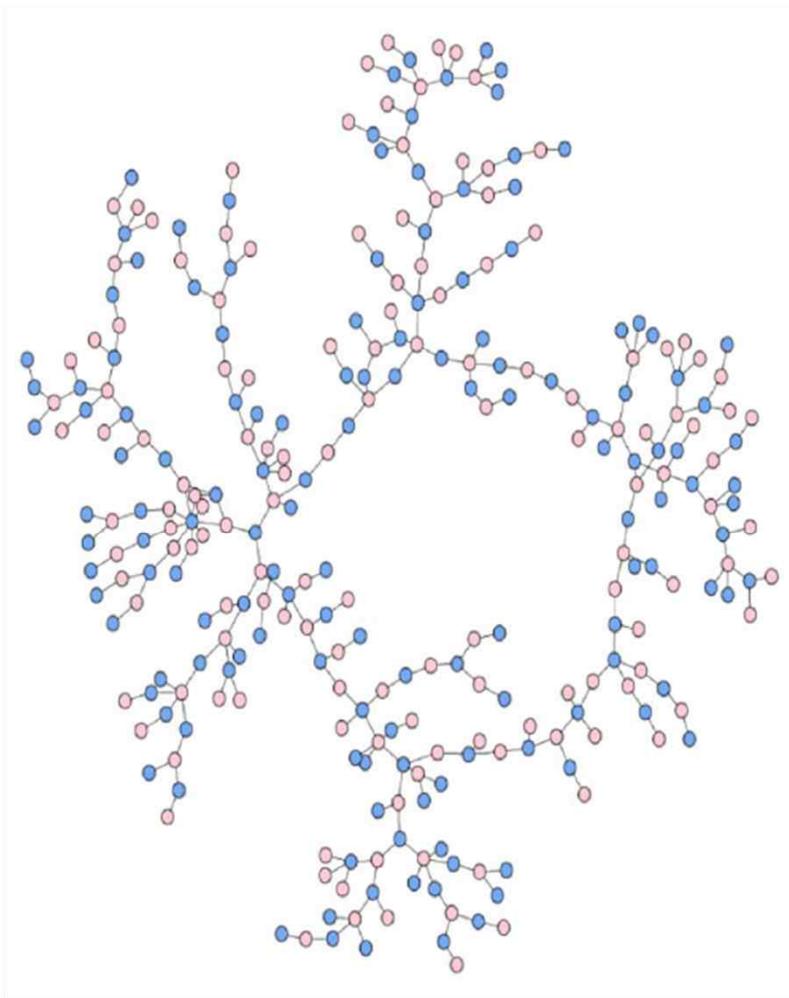


Graph applications

graph	vertices	edges
communication	telephones, computers	fiber optic cables
circuits	gates, registers, processors	wires
mechanical	joints	rods, beams, springs
hydraulic	reservoirs, pumping stations	pipelines
financial	stocks, currency	transactions
transportation	street intersections, airports	highways, airway routes
scheduling	tasks	precedence constraints
software systems	functions	function calls
internet	web pages	hyperlinks
games	board positions	legal moves
social relationship	people, actors	friendships, movie casts
neural networks	neurons	synapses
protein networks	proteins	protein-protein interactions
chemical compounds	molecules	bonds

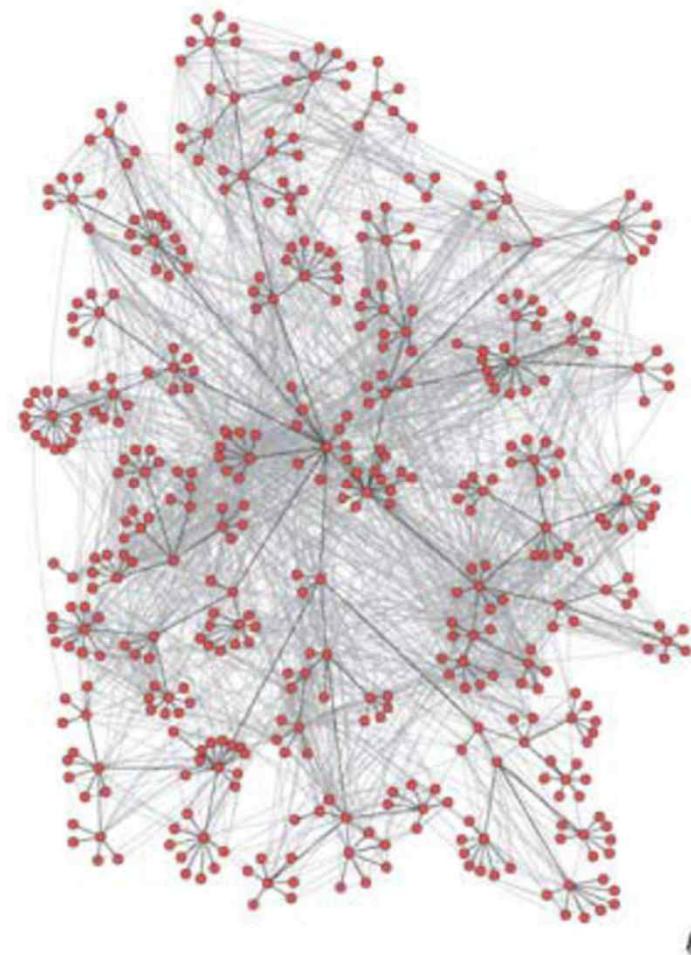
Social networks

high school dating



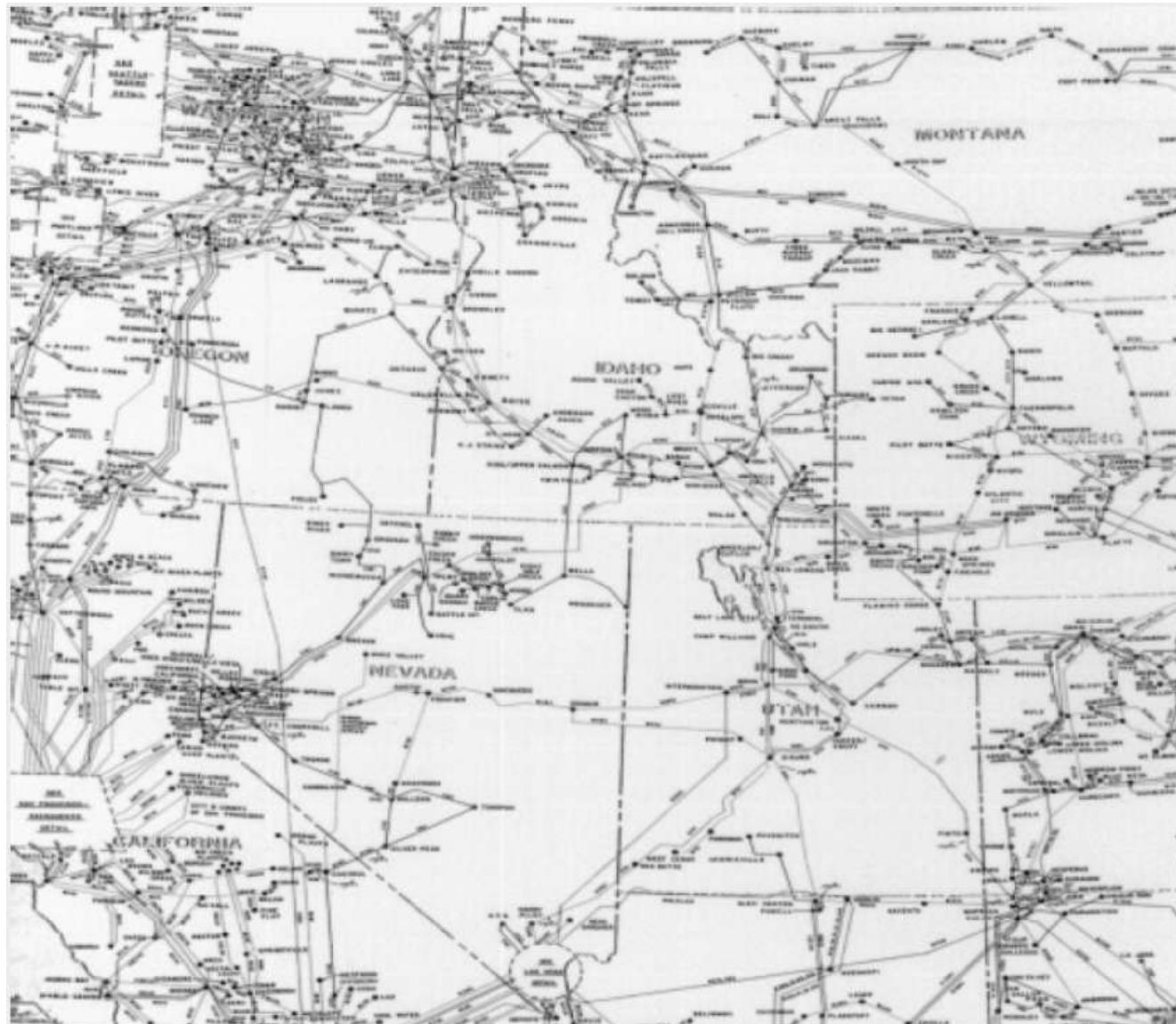
Reference: Bearman, Moody and Stovel, 2004
image by Mark Newman

corporate e-mail



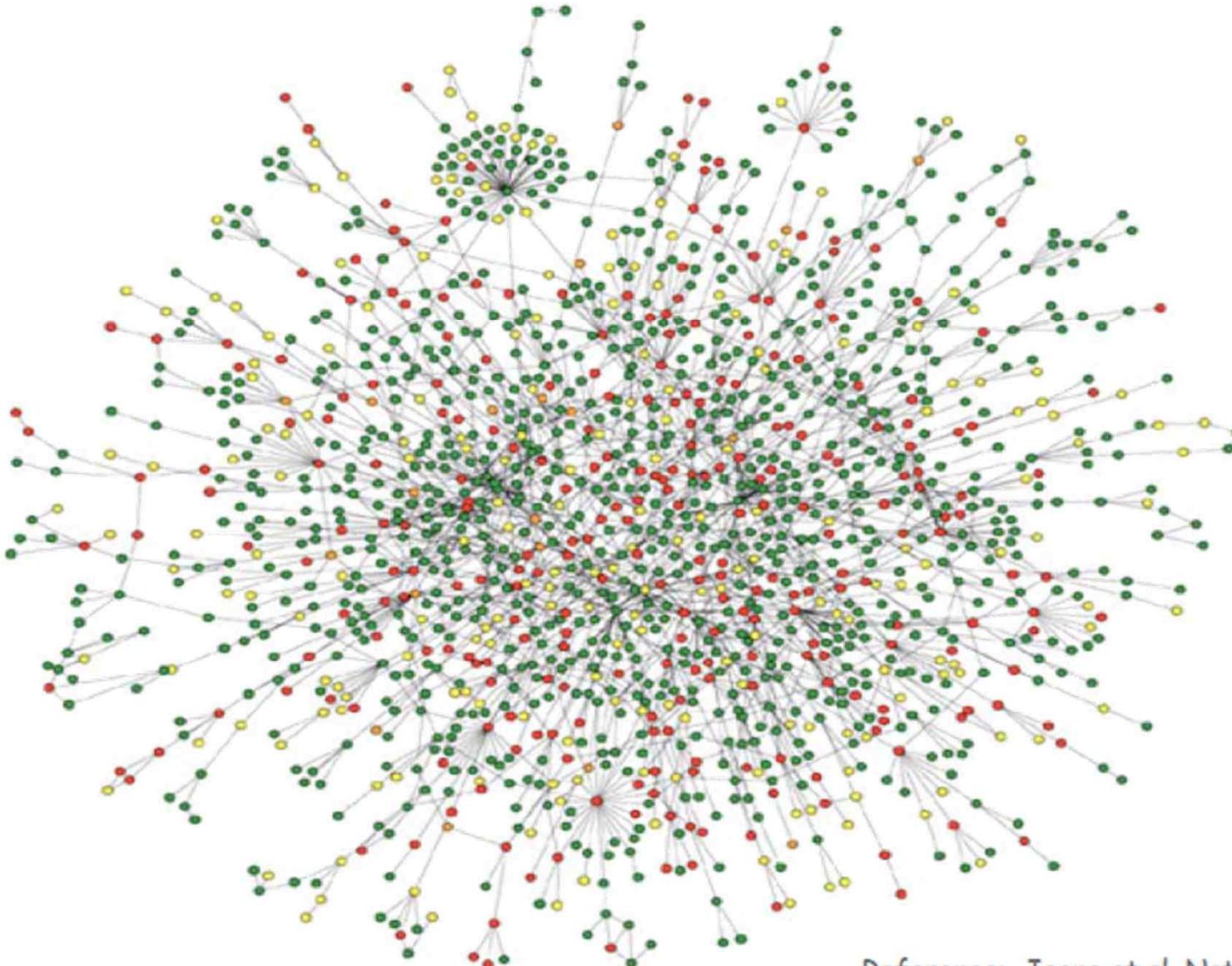
Reference: Adamic and Adar, 2004

Power transmission grid of Western US



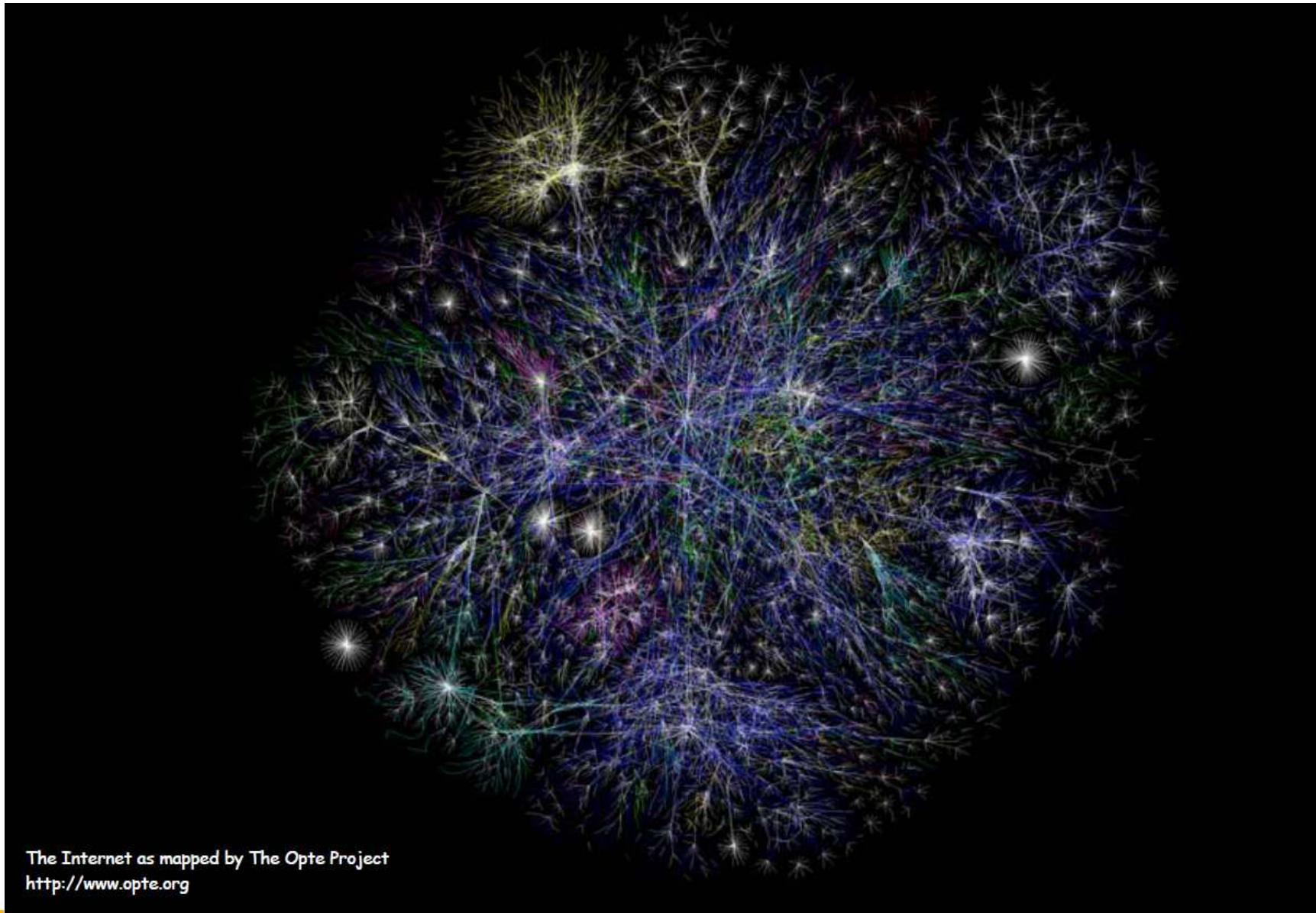
Reference: Duncan Watts

Protein interaction network

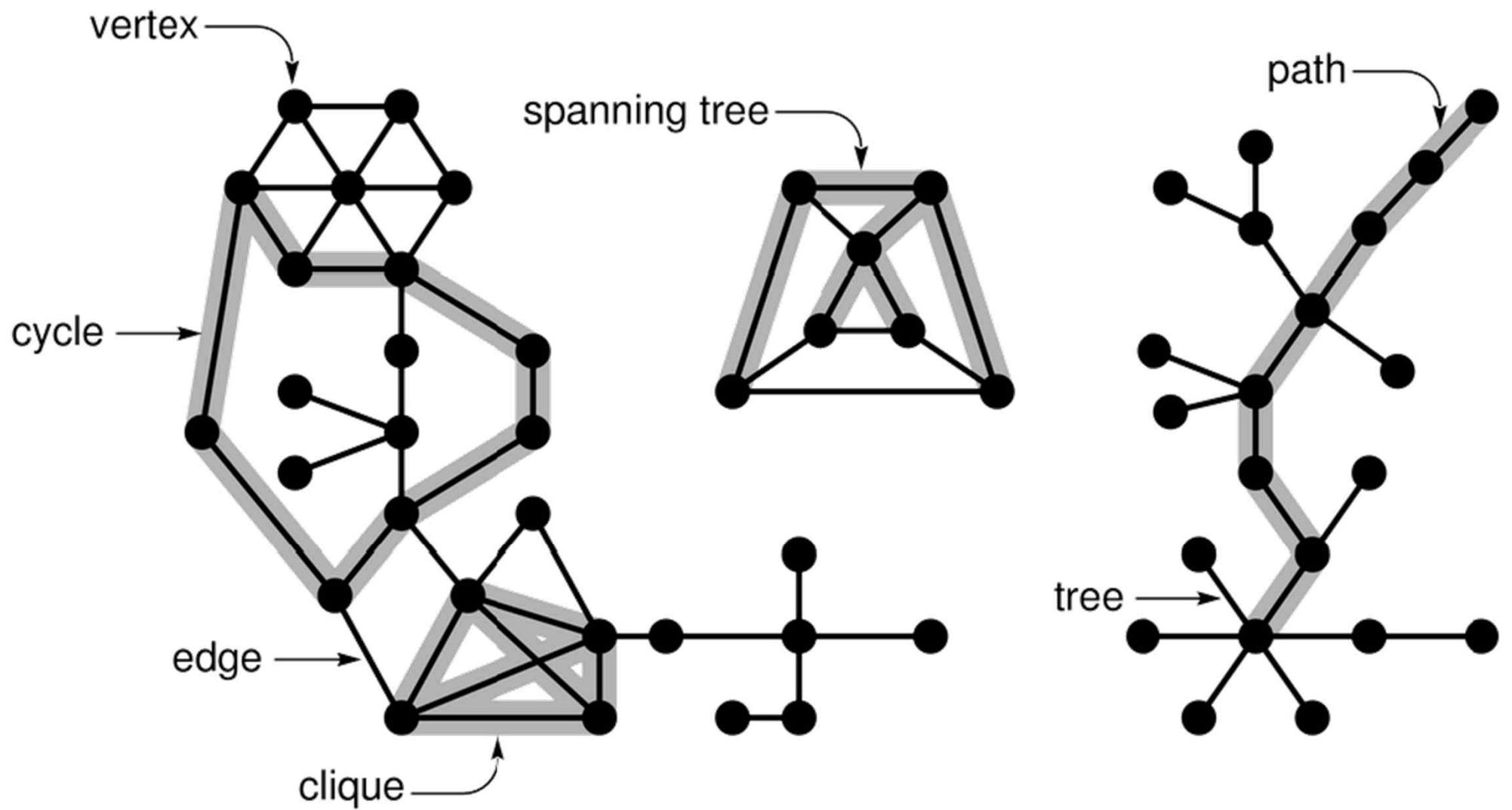


Reference: Jeong et al, Nature Review | Genetics

The Internet



Graph terminology



Some graph-processing problems

- Path : Is there a path between s to t ?
- Shortest path : What is the shortest path between s and t ?
- Longest path : What is the longest simple path between s and t ?
- Cycle : Is there a cycle in the graph?
- Euler tour : Is there a cycle that uses each **edge** exactly once?
- Hamilton tour : Is there a cycle that uses each **vertex** exactly once?
- Connectivity : Is there a way to connect all of the vertices?
- MST : What is the best way to connect all of the vertices?
- Biconnectivity : Is there a vertex whose removal disconnects the graph?
- Planarity : Can you draw the graph in the plane with no crossing edges?

First challenge: Which of these problems is easy? difficult? intractable?

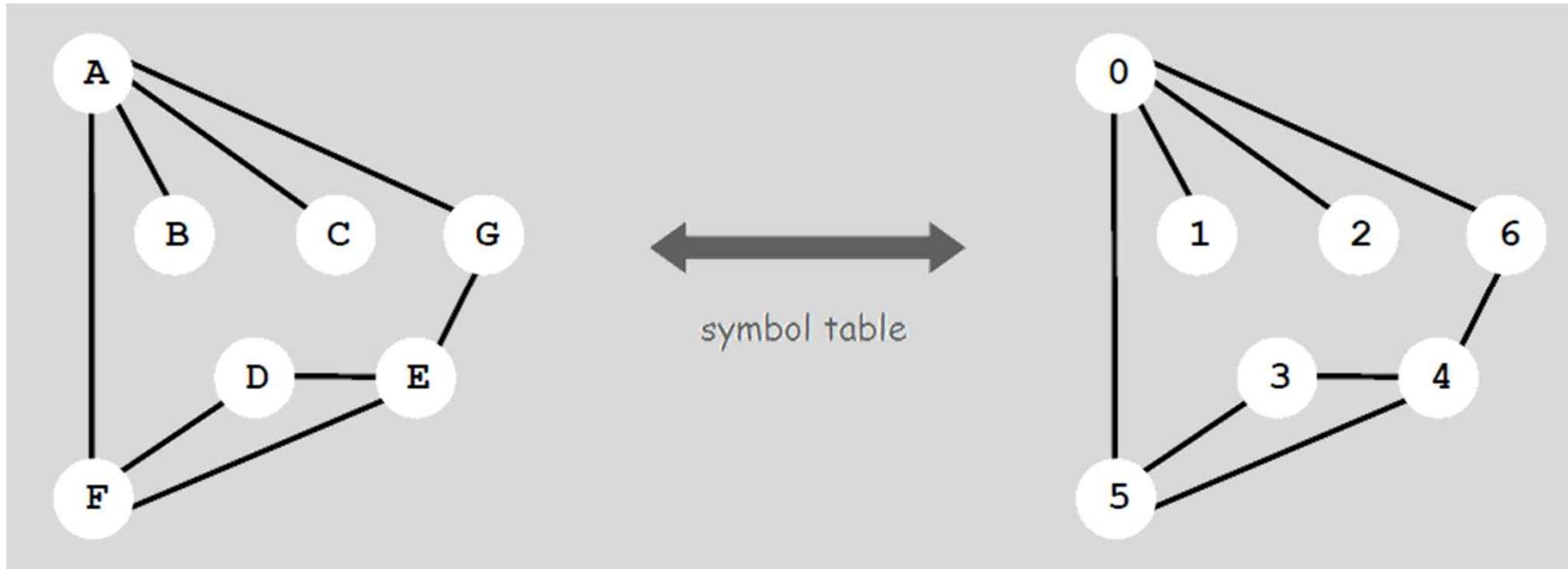


10. Undirected Graphs

- Graph API
- maze exploration
- depth-first search
- breadth-first search
- connected components
- challenges

Graph representation

- Vertex representation.
 - ◆ This lecture: use integers between 0 and $V-1$.
 - ◆ Real world: convert between names and integers with symbol table.



- Other issues. Parallel edges, self-loops.

Graph API

```
public class Graph  (graph data type)
    Graph(int V)           create an empty graph with V vertices
    Graph(int V, int E)    create a random graph with V vertices, E edges
    void addEdge(int v, int w)  add an edge v-w
Iterable<Integer> adj(int v)      return an iterator over the neighbors of v
    int V()                return number of vertices
    String toString()       return a string representation
```

Client that iterates through all edges

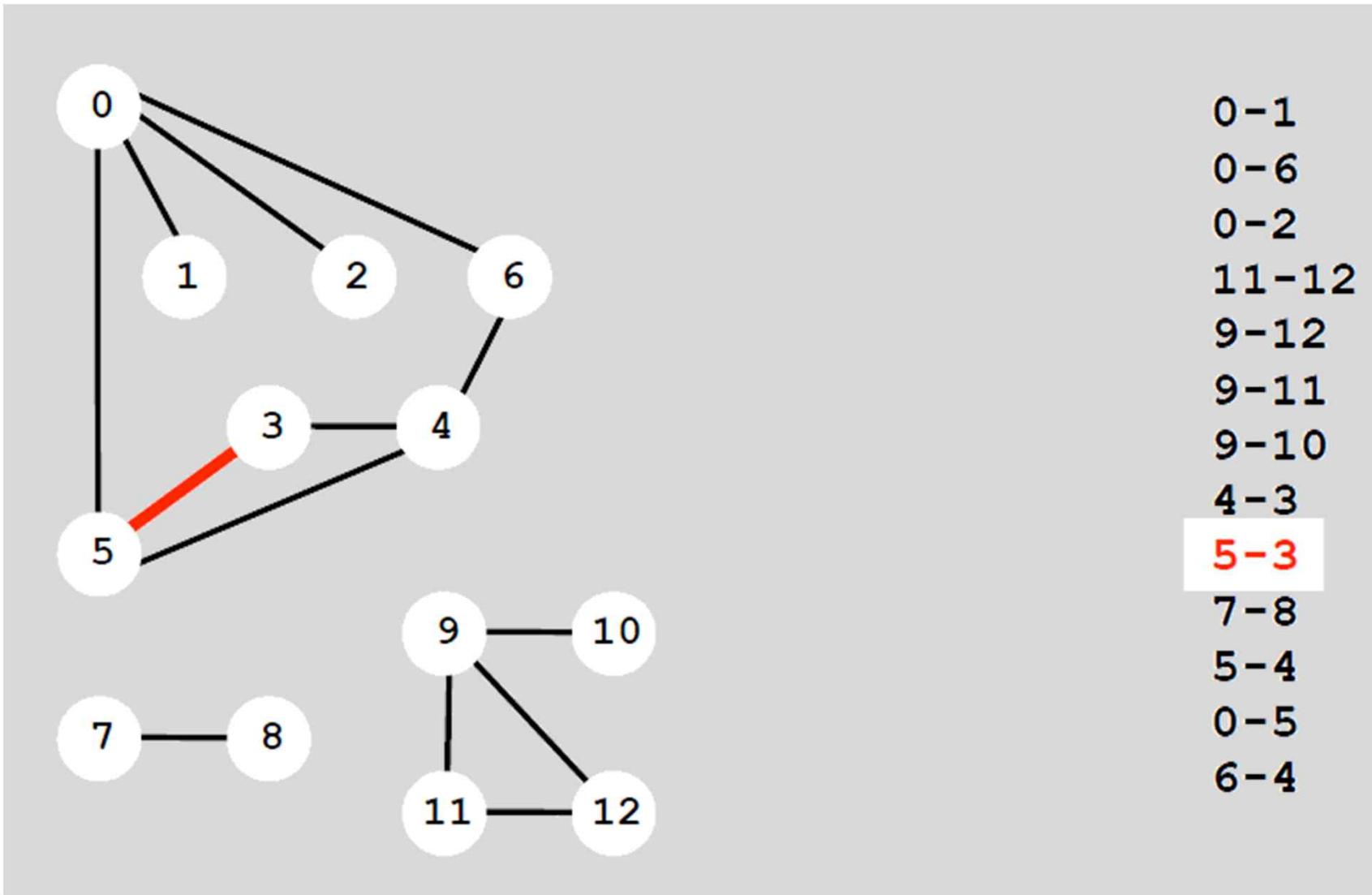
```
Graph G = new Graph(V, E);
StdOut.println(G);
for (int v = 0; v < G.V(); v++)
    for (int w : G.adj(v))
        // process edge v-w
```

processes BOTH
v-w and w-v



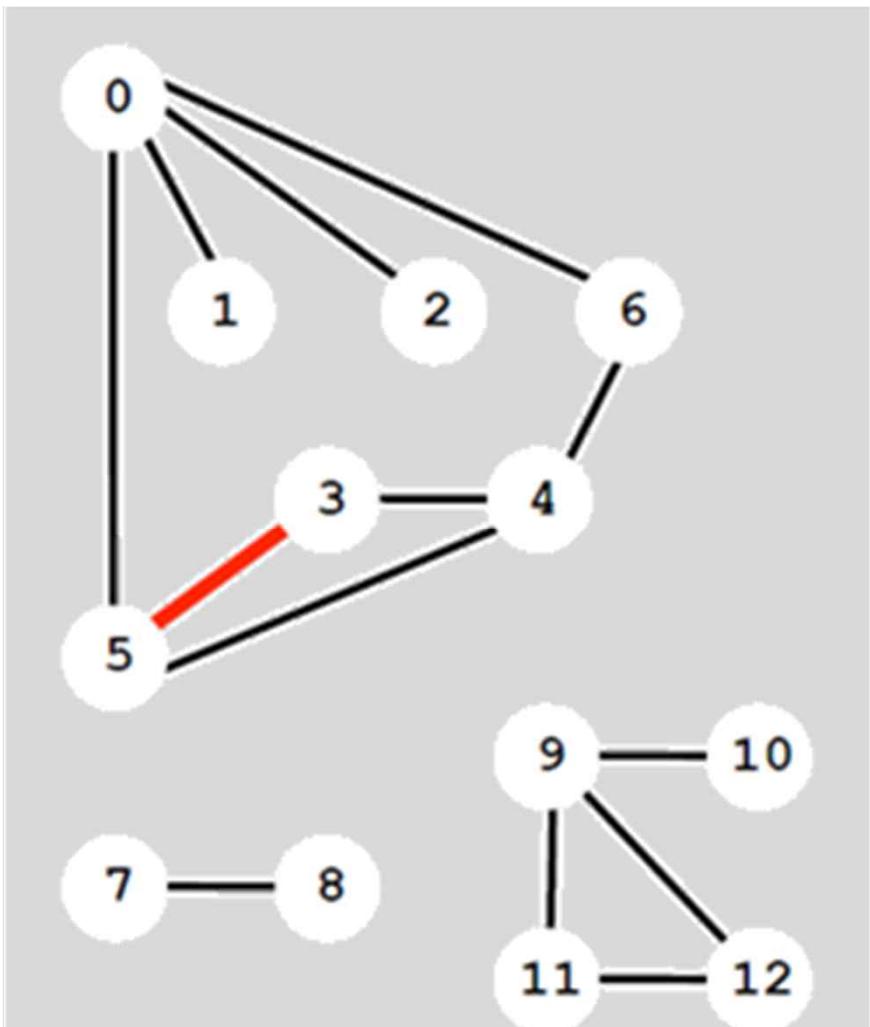
Set of edges representation

- Store a list of the edges (linked list or array)



Adjacency matrix representation

- Maintain a two-dimensional $V \times V$ boolean array.
- For each edge $v-w$ in graph: $\text{adj}[v][w] = \text{adj}[w][v] = \text{true}$.



two entries
for each
edge

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	0	0	1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	1	1	1
10	0	0	0	0	0	0	0	1	0	0	0	0	0
11	0	0	0	0	0	0	0	0	1	0	0	1	0
12	0	0	0	0	0	0	0	0	1	0	1	0	1

Adjacency-matrix graph representation: Java implementation

```
public class Graph
{
    private int V;
    private boolean[][] adj; ← adjacency matrix

    public Graph(int V)
    {
        this.V = V;
        adj = new boolean[V][V]; ← create empty V-vertex graph
    }

    public void addEdge(int v, int w)
    {
        adj[v][w] = true;
        adj[w][v] = true; ← add edge v-w (no parallel edges)
    }

    public Iterable<Integer> adj(int v)
    {
        return new AdjIterator(v); ← iterator for v's neighbors
    }
}
```

Adjacency matrix: iterator for vertex neighbors

```
private class AdjIterator implements Iterator<Integer>,
    Iterable<Integer>
{
    int v, w = 0;
    AdjIterator(int v)
    { this.v = v; }

    public boolean hasNext()
    {
        while (w < V)
        { if (adj[v][w]) return true; w++ }
        return false;
    }

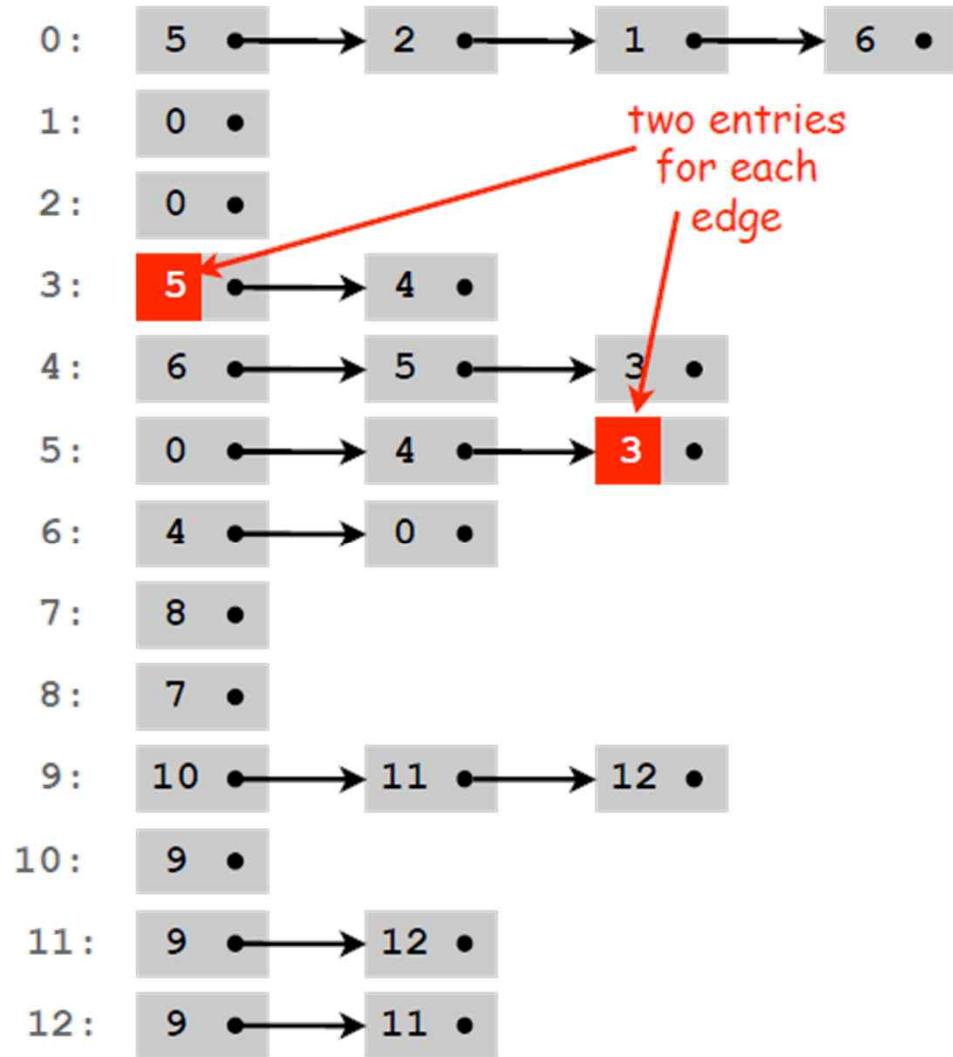
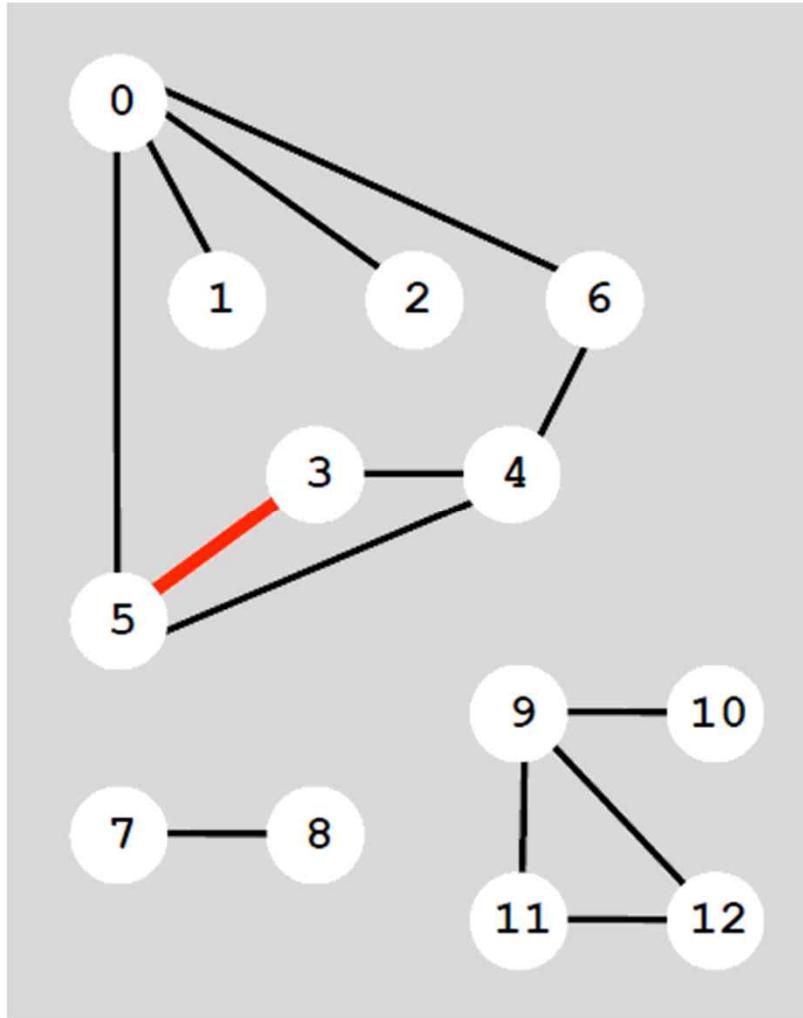
    public int next()
    {
        if (hasNext()) return w++ ;
        else throw new NoSuchElementException();
    }

    public Iterator<Integer> iterator()
    { return this; }
}
```



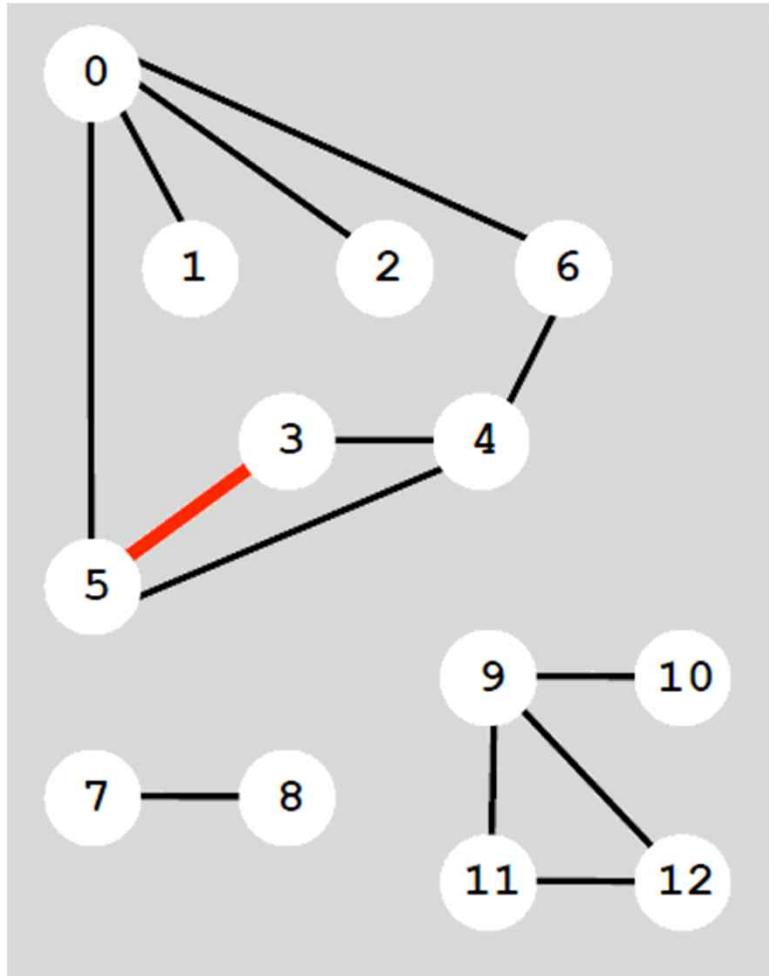
Adjacency-list graph representation

- Maintain vertex-indexed array of lists (implementation omitted)



Adjacency-SET graph representation

- Maintain vertex-indexed array of SETs
(take advantage of balanced-tree or hashing implementations)



0:	{ 1 2 5 6 }
1:	{ 0 }
2:	{ 0 }
3:	{ 4 5 }
4:	{ 3 5 6 }
5:	{ 0 3 4 }
6:	{ 0 4 }
7:	{ 8 }
8:	{ 7 }
9:	{ 10 11 12 }
10:	{ 9 }
11:	{ 9 12 }
12:	{ 9 1 }

two entries for each edge

Adjacency-SET graph representation: Java implementation

```
public class Graph
{
    private int V;
    private SET<Integer>[] adj; ← adjacency sets

    public Graph(int V)
    {
        this.V = V;
        adj = (SET<Integer>[]) new SET[V];
        for (int v = 0; v < V; v++)
            adj[v] = new SET<Integer>(); ← create empty V-vertex graph
    }

    public void addEdge(int v, int w)
    {
        adj[v].add(w);
        adj[w].add(v); ← add edge v-w (no parallel edges)
    }

    public Iterable<Integer> adj(int v)
    {
        return adj[v]; ← iterable SET for v's neighbors
    }
}
```

Graph representations

- Graphs are abstract mathematical objects, BUT
 - ◆ ADT (Abstract Data Type) implementation requires specific representation.
 - ◆ Efficiency depends on matching algorithms to representations.

representation	space	edge between v and w?	iterate over edges incident to v?
list of edges	E	E	E
adjacency matrix	V^2	1	V
adjacency list	$E + V$	$\text{degree}(v)$	$\text{degree}(v)$
adjacency SET	$E + V$	$\log(\text{degree}(v))$	$\text{degree}(v)^*$

- In practice: Use adjacency SET representation

- ◆ Take advantage of proven technology
- ◆ Real-world graphs tend to be "sparse"
[huge number of vertices, small average vertex degree]
- ◆ Algs all based on iterating over edges incident to v.

* easy to also support
ordered iteration and
randomized iteration

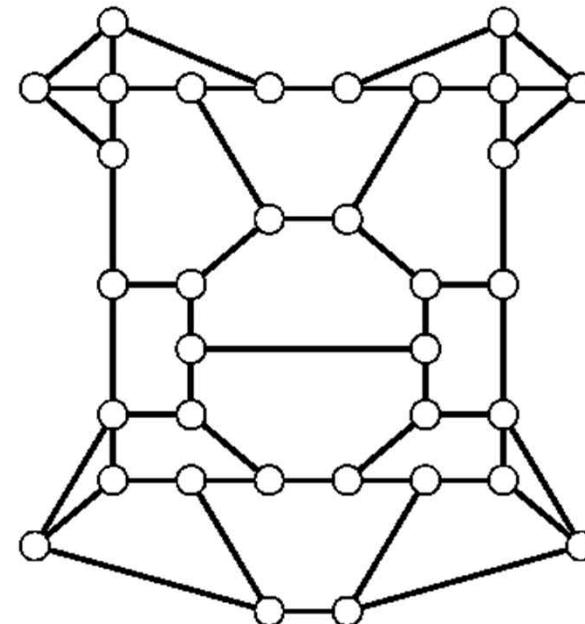
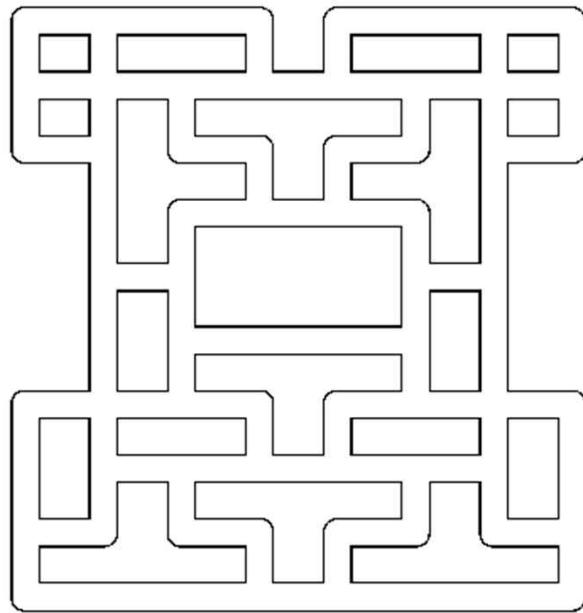
10. Undirected Graphs

- Graph API
- maze exploration
- depth-first search
- breadth-first search
- connected components
- challenges

Maze exploration

□ Maze graphs.

- ◆ Vertex = intersections.
- ◆ Edge = passage.

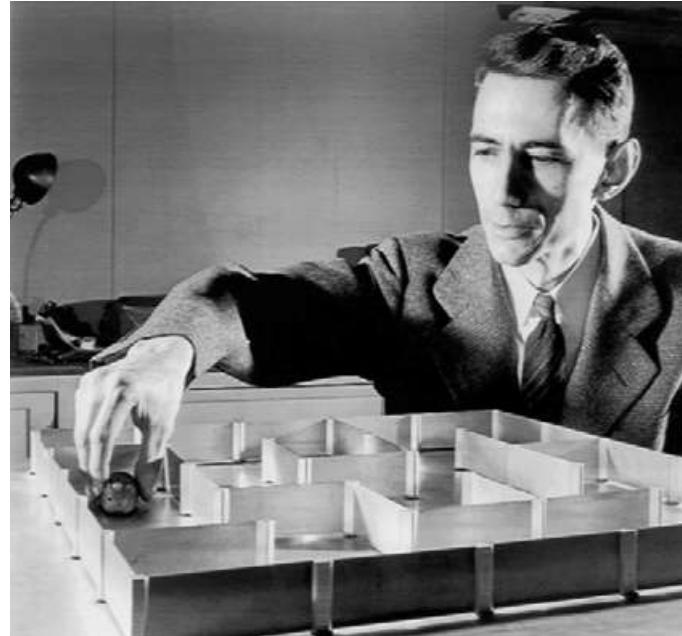


□ Goal

- ◆ Explore every passage in the maze.

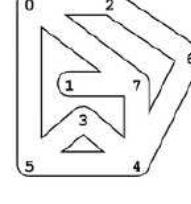
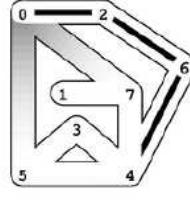
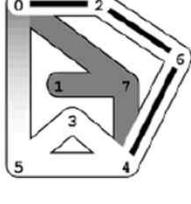
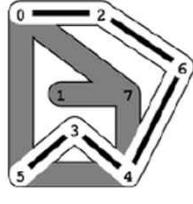
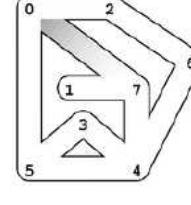
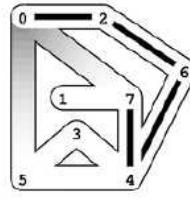
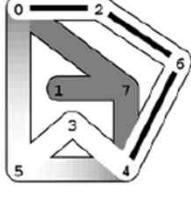
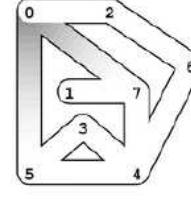
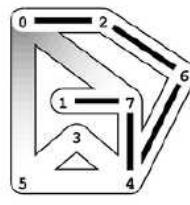
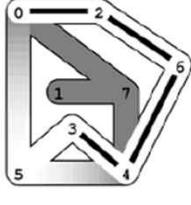
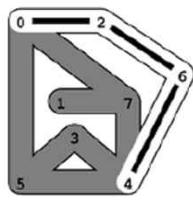
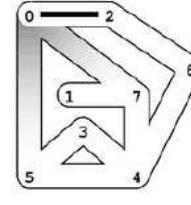
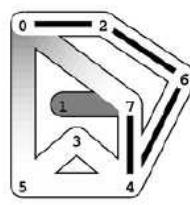
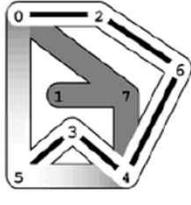
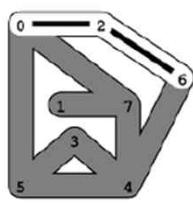
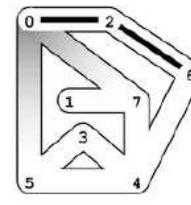
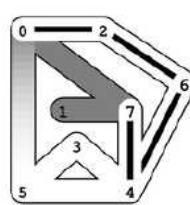
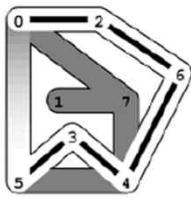
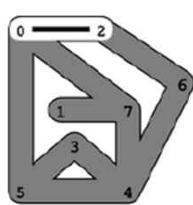
Trémaux Maze Exploration

- Trémaux maze exploration.
 - ◆ Unroll a ball of string behind you.
 - ◆ Mark each visited **intersection** by turning on a light.
 - ◆ Mark each visited **passage** by opening a door
- First use? Theseus entered labyrinth to kill the monstrous Minotaur; Ariadne held ball of string.

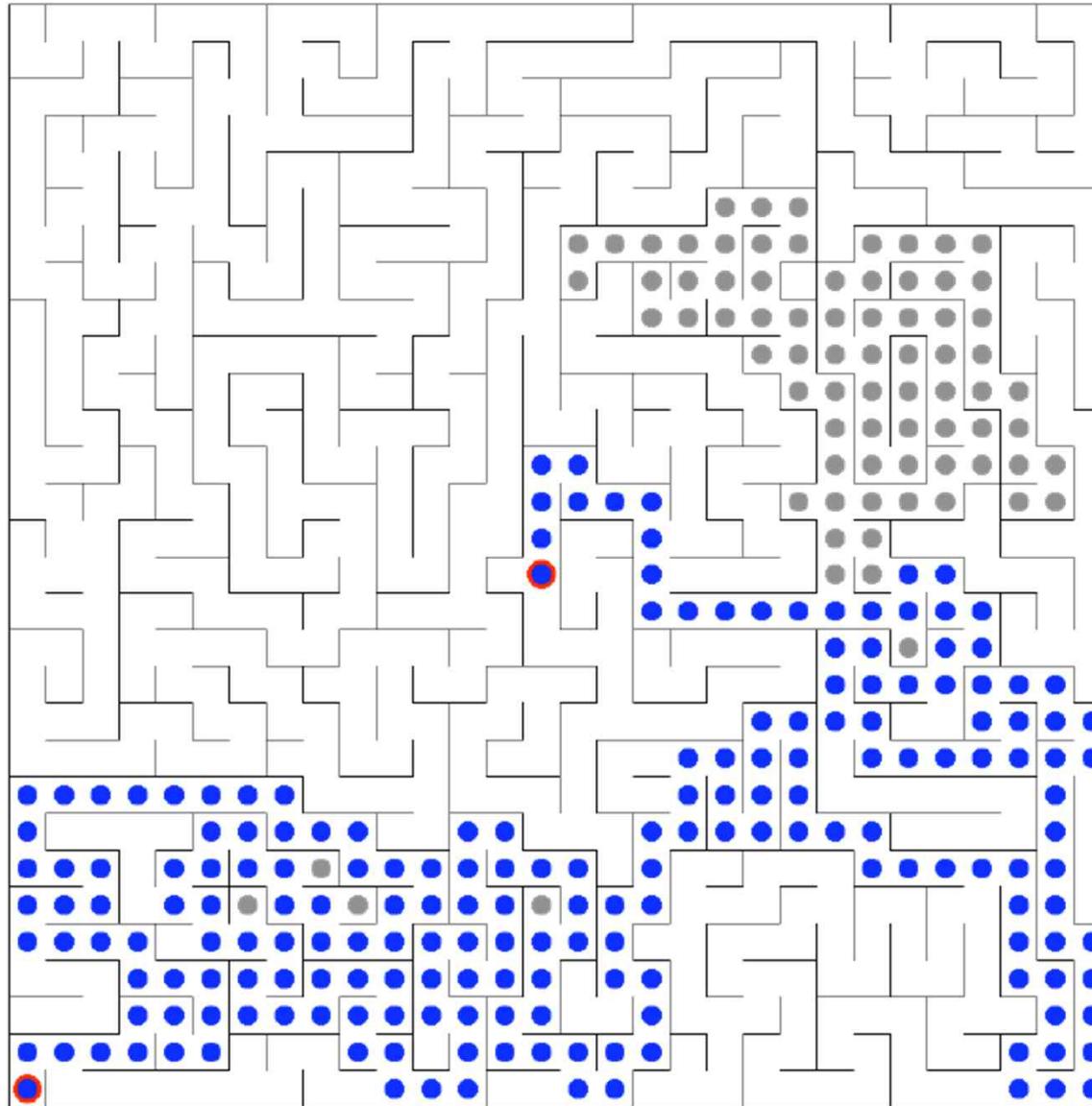


Claude Shannon (with Theseus mouse)

Trémaux Maze Exploration



Maze Exploration



10. Undirected Graphs

- Graph API
- maze exploration
- depth-first search
- breadth-first search
- connected components
- challenges

Flood fill

- Photoshop “magic wand”



- Problem: Flood fill
- Assumptions: picture has millions to billions of pixels

Depth-first search

- **Goal.** Systematically search through a graph.
- **Idea.** Mimic maze exploration.
- **Typical applications.**
 - ◆ find all vertices connected to a given s
 - ◆ find a path from s to t

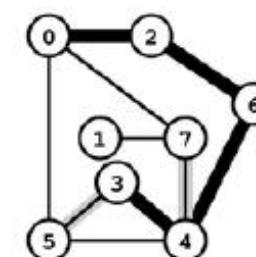
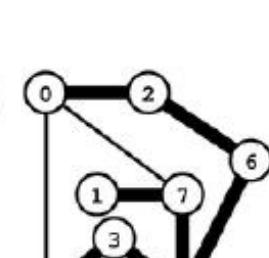
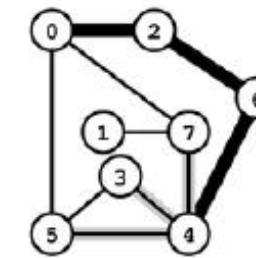
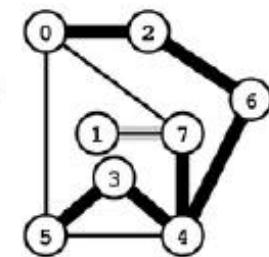
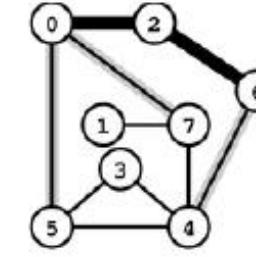
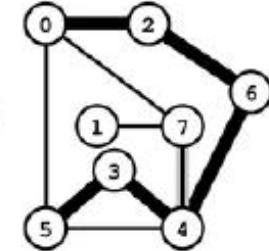
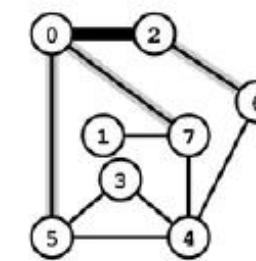
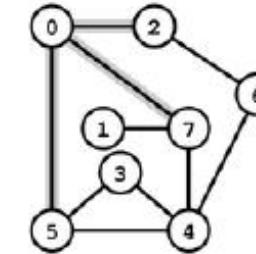
DFS (to visit a vertex s)

Mark s as visited.

Visit all unmarked vertices v adjacent to s .

↑
recursive

- **Running time.**
 - ◆ $O(E)$ since each edge examined at most twice
 - ◆ usually less than V to find paths in real graphs



ithms

Design pattern for graph processing

- Typical client program.
 - ◆ Create a Graph.
 - ◆ Pass the Graph to a graph-processing routine, e.g., DFSearcher.
 - ◆ Query the graph-processing routine for information.

Client that prints all vertices connected to (reachable from) s

```
public static void main(String[] args)
{
    In in = new In(args[0]);
    Graph G = new Graph(in);
    int s = 0;
    DFSearcher dfs = new DFSearcher(G, s);
    for (int v = 0; v < G.V(); v++)
        if (dfs.isConnected(v))
            System.out.println(v);
}
```

Decouple **graph** from **graph processing**.

Depth-first search (connectivity)

```
public class DFSearcher
{
    private boolean[] marked; ← true if connected to s

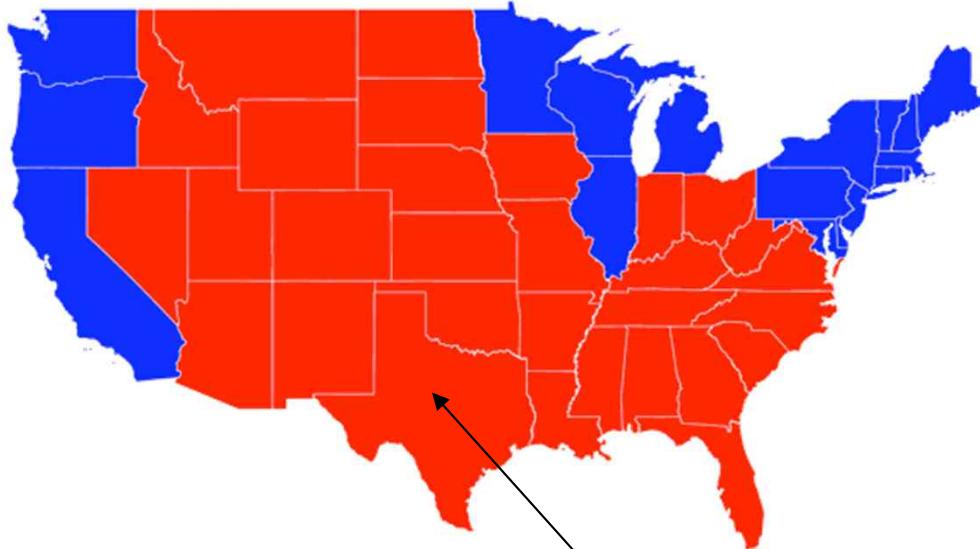
    public DFSearcher(Graph G, int s)
    {
        marked = new boolean[G.V()]; ← constructor marks vertices connected to s
        dfs(G, s);
    }

    private void dfs(Graph G, int v)
    {
        marked[v] = true;
        for (int w : G.adj(v)) ← recursive DFS does the work
            if (!marked[w]) dfs(G, w);
    }

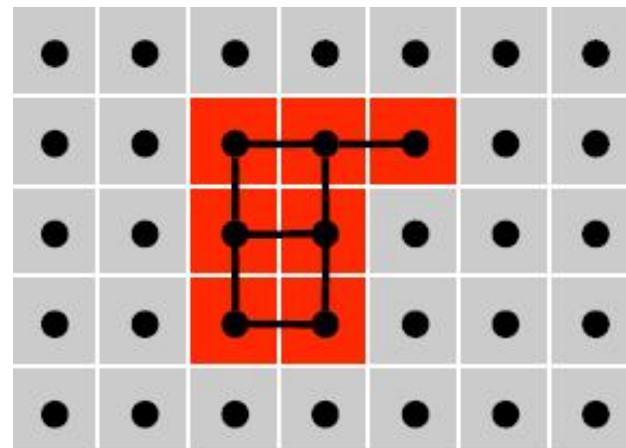
    public boolean isReachable(int v)
    {
        return marked[v]; ← client can ask whether any vertex is connected to s
    }
}
```

Connectivity Application: Flood fill

- Change color of entire blob of neighboring red pixels to blue.
- Build a *grid graph*
 - ◆ vertex: pixel.
 - ◆ edge: between two adjacent red pixels.
 - ◆ blob: all pixels connected to given pixel.

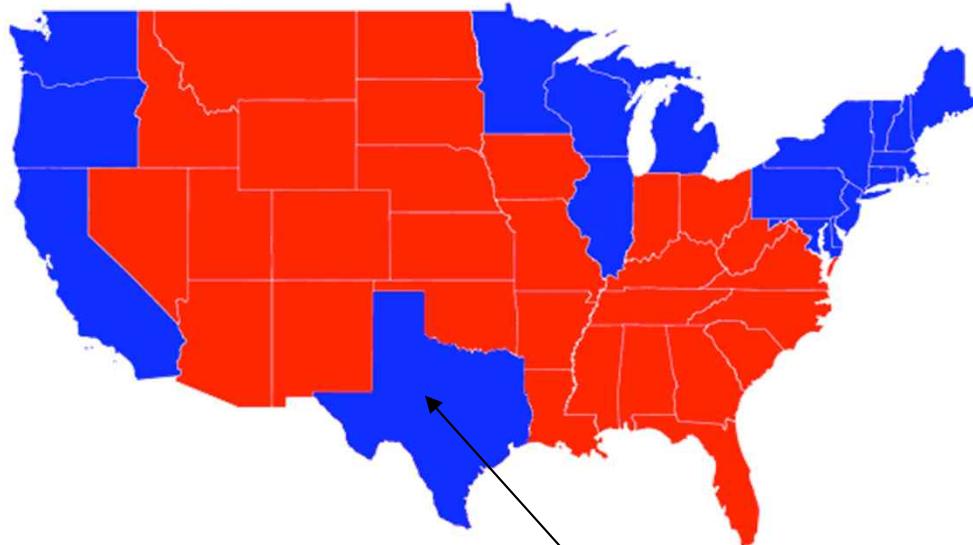


recolor red blob to blue

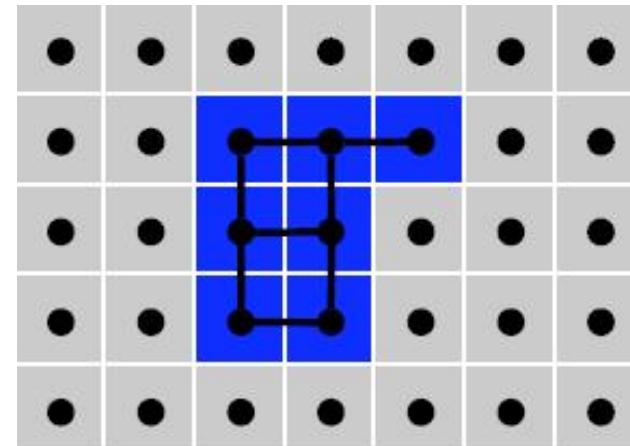


Connectivity Application: Flood Fill

- Change color of entire blob of neighboring **red** pixels to **blue**.
- Build a **grid graph**
 - ◆ vertex: pixel.
 - ◆ edge: between two adjacent red pixels.
 - ◆ blob: all pixels connected to given pixel.

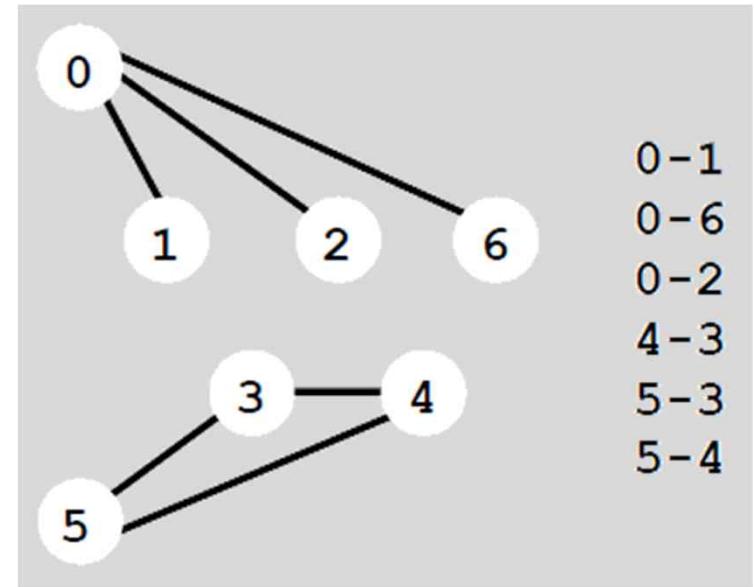
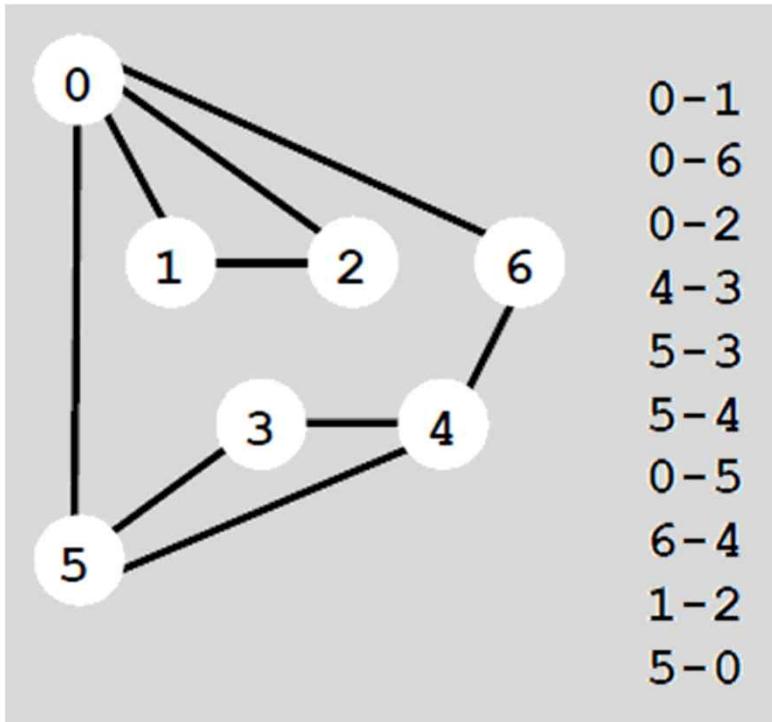


recolor red blob to blue



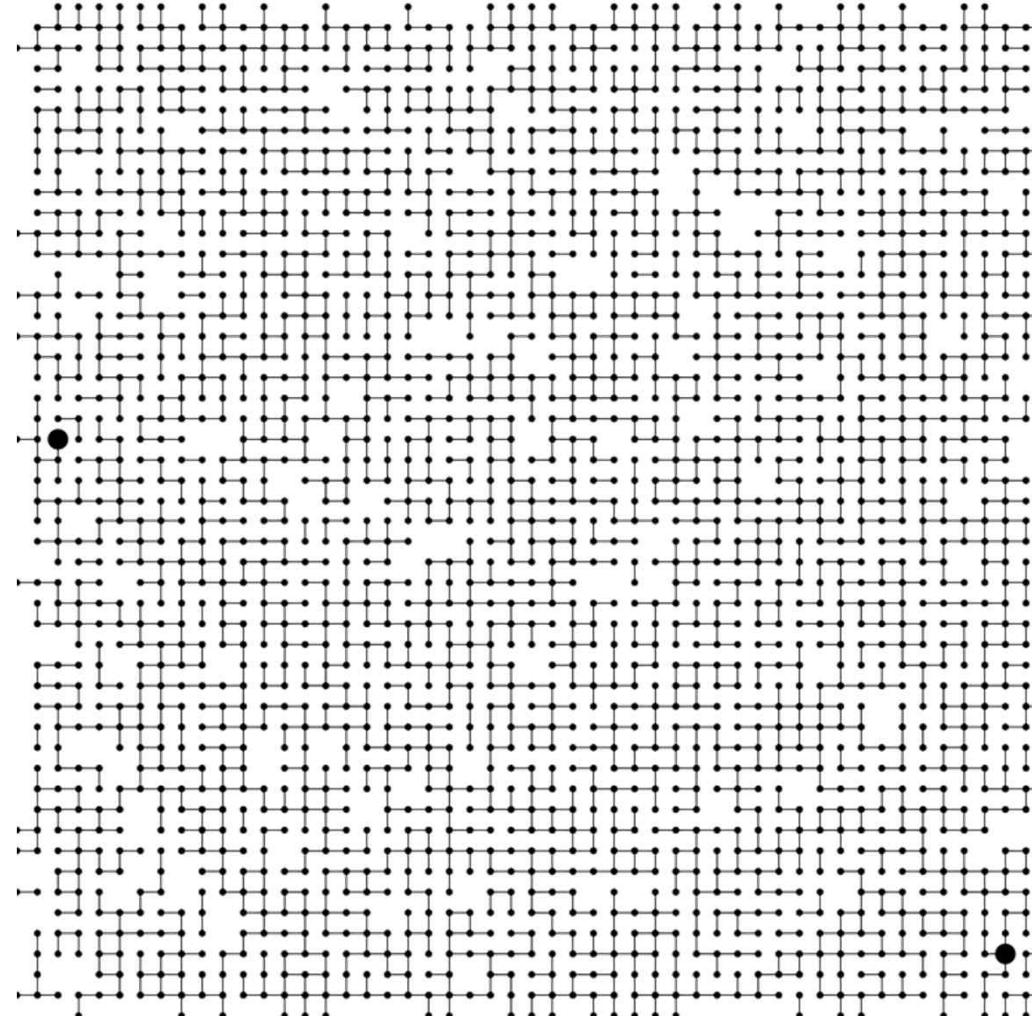
Graph-processing challenges:

- Problem 1: Is there a path from s to t ?
- Problem 2: Find a path from s to t .
- Assumptions: any path will do



Paths in graphs

- Is there a path from s to t ? If so, **find one**.



Paths in graphs

- Is there a path from s to t ?

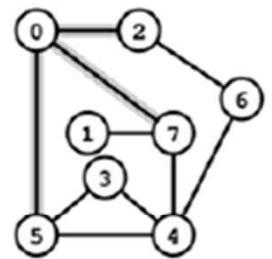
method	preprocess time	query time	space
Union Find	$V + E \log^* V$	$\log^* V$ †	V
DFS	$E + V$	1	$E + V$

† amortized

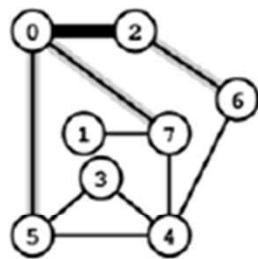
- If so, find one.
 - ◆ Union-Find: no help (use DFS on connected subgraph)
 - ◆ DFS: easy (stay tuned)
- UF advantage. Can intermix queries and edge insertions.
- DFS advantage. Can recover path itself in time proportional to its length.

Keeping track of paths with DFS

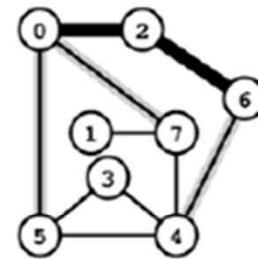
- **DFS tree.** Upon visiting a vertex v for the first time, remember that you came from $\text{pred}[v]$ (parent-link representation).
- **Retrace path.** To find path between s and v , follow pred back from v .



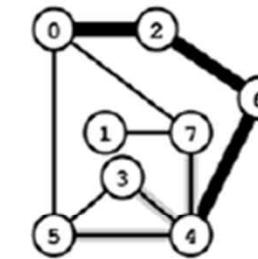
0



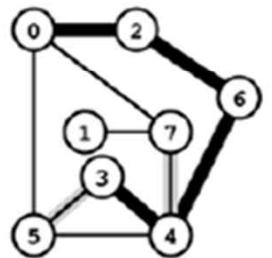
0



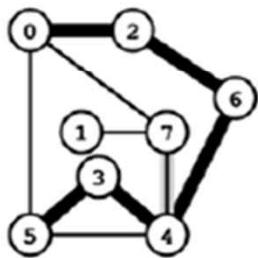
0
2



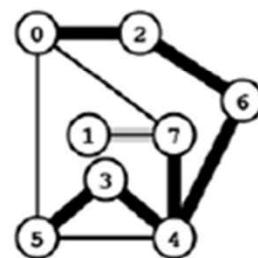
0
2
6



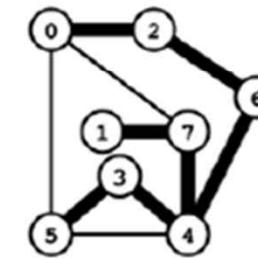
0
2
6
4



0
2
6
4
3



0
2
6
4
3
5



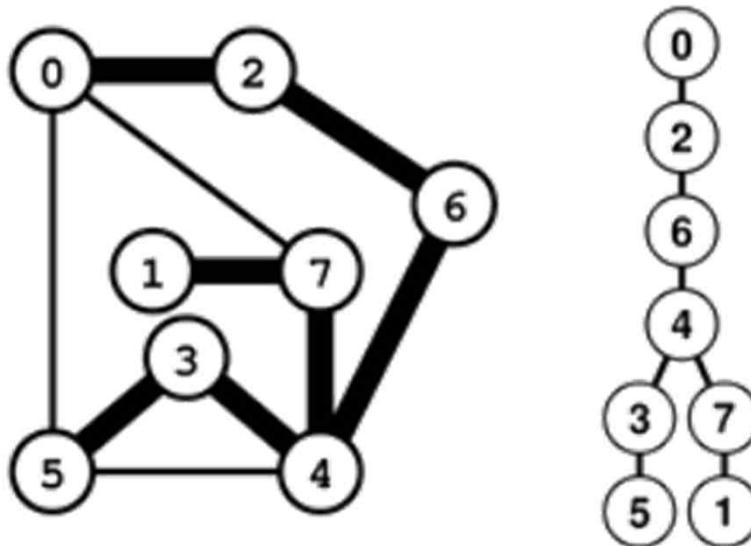
0
2
6
4
3
5
1

Depth-first-search (pathfinding)

```
public class DFSearcher
{
    ...
    private int[] pred; ← add instance variable for
    public DFSearcher(Graph G, int s) parent-link representation
    { of DFS tree
        ...
        pred = new int[G.V()];
        for (int v = 0; v < G.V(); v++) ← initialize it in the
            pred[v] = -1; constructor
        ...
    }
    private void dfs(Graph G, int v)
    {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w])
            {
                pred[w] = v; ← set parent link
                dfs(G, w);
            }
    }
    public Iterable<Integer> path(int v) ← add method for client
    { // next slide } to iterate through path
}
```

Depth-first-search (pathfinding iterator)

```
public Iterable<Integer> path(int v)
{
    Stack<Integer> path = new Stack<Integer>();
    while (v != -1 && marked[v])
    {
        list.push(v);
        v = pred[v];
    }
    return path;
}
```



DFS summary

- Enables direct solution of simple graph problems.
 - ◆ Find path from s to t . ✓
 - ◆ Connected components (stay tuned).
 - ◆ Euler tour (see book).
 - ◆ Cycle detection (simple exercise).
 - ◆ Bipartiteness checking (see book).

- Basis for solving more difficult graph problems.
 - ◆ Biconnected components (see book).
 - ◆ Planarity testing (beyond scope).

10. Undirected Graphs

- Graph API
- maze exploration
- depth-first search
- breadth-first search
- connected components
- challenges

Breadth First Search

- Depth-first search. Put unvisited vertices on a **stack**.
- Breadth-first search. Put unvisited vertices on a **queue**.

- Shortest path. Find path from s to t that uses fewest number of edges.

BFS (from source vertex s)

Put s onto a FIFO queue.

Repeat until the queue is empty:

- remove the least recently added vertex v
 - add each of v 's unvisited neighbors to the queue,
and mark them as visited.
-

- Property. BFS examines vertices in **increasing distance** from s .

Breadth-first search scaffolding

```
public class BFSearcher
{
    private int[] dist; ← distances from s

    public BFSearcher(Graph G, int s)
    {
        dist = new int[G.V()];
        for (int v = 0; v < G.V(); v++)
            dist[v] = G.V() + 1; ← initialize distances
        dist[s] = 0;

        bfs(G, s); ← compute distances
    }

    public int distance(int v)
    { return dist[v]; } ← answer client query

    private void bfs(Graph G, int s)
    { // See next slide. }

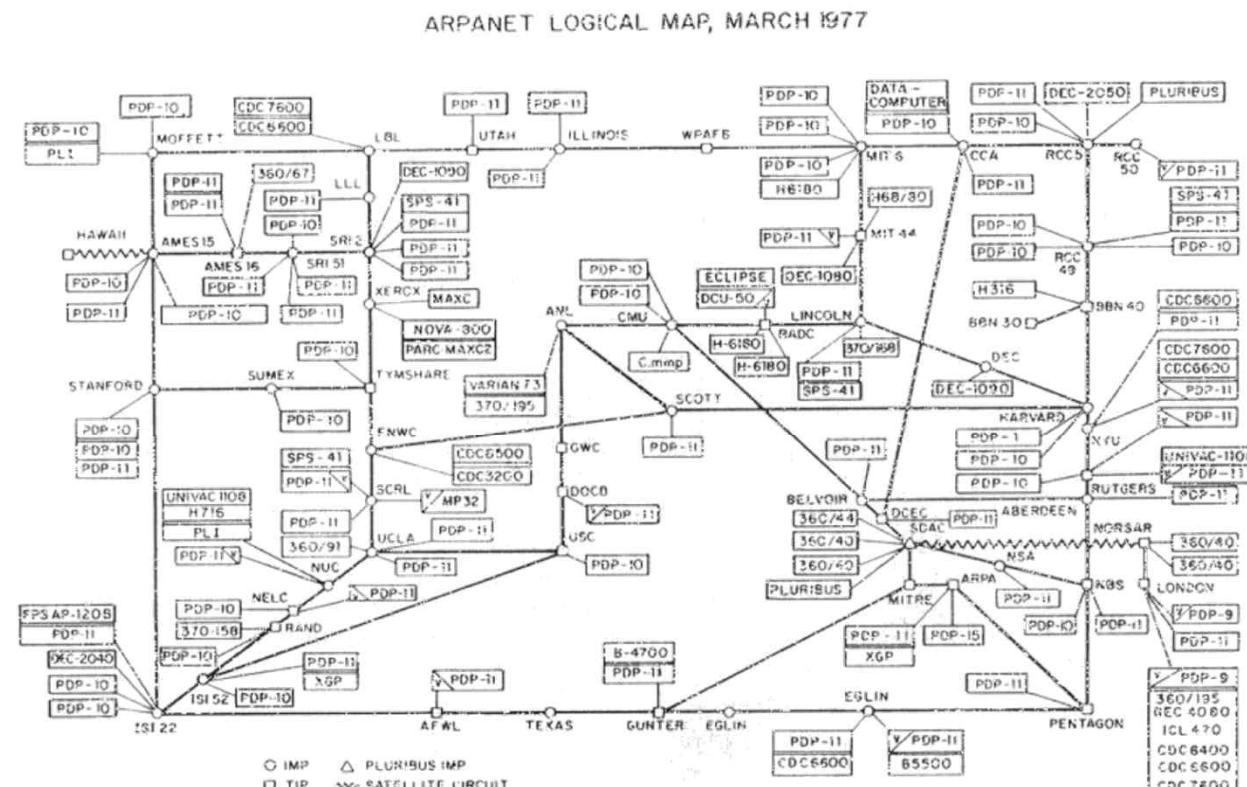
}
```

Breadth-first search (compute shortest-path distances)

```
private void bfs(Graph G, int s)
{
    Queue<Integer> q = new Queue<Integer>();
    q.enqueue(s);
    while (!q.isEmpty())
    {
        int v = q.dequeue();
        for (int w : G.adj(v))
        {
            if (dist[w] > G.V())
            {
                q.enqueue(w);
                dist[w] = dist[v] + 1;
            }
        }
    }
}
```

BFS Application

- Kevin Bacon numbers - six degree of separation
- Facebook.
- Fewest number of hops in a communication network.

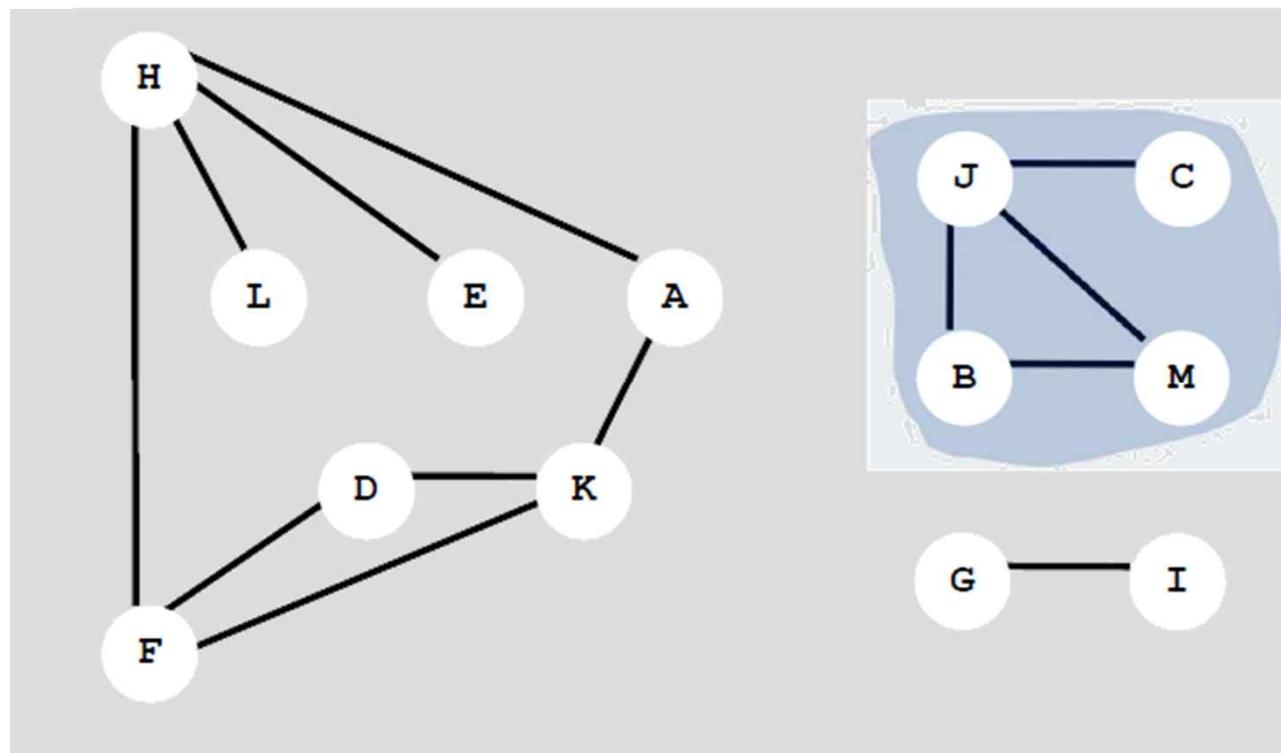


10. Undirected Graphs

- Graph API
- maze exploration
- depth-first search
- breadth-first search
- connected components
- challenges

Connectivity Queries

- Def. Vertices v and w are **connected** if there is a path between them.
- Def. A connected component is a maximal set of connected vertices.
- Goal. Preprocess graph to answer queries: is v connected to w ?
in **constant time**



Vertex	Component
A	0
B	1
C	1
D	0
E	0
F	0
G	2
H	0
I	2
J	1
K	0
L	0
M	1

Union-Find? not quite

Connected Components

- Goal. Partition vertices into connected components.

Connected components

Initialize all vertices v as unmarked.

For each unmarked vertex v , run DFS and identify all vertices discovered as part of the same connected component.

preprocess Time	query Time	extra Space
$E + V$	1	V

Depth-first search for connected components

```
public class CCFinder
{
    private final static int UNMARKED = -1;
    private int components;
    private int[] cc;
    public CCFinder(Graph G)
    {
        cc = new int[G.V()];
        for (int v = 0; v < G.V(); v++)
            cc[v] = UNMARKED;
        for (int v = 0; v < G.V(); v++)
            if (cc[v] == UNMARKED)
                { dfs(G, v); components++; }
    }
    private void dfs(Graph G, int v)
    {
        cc[v] = components;
        for (int w : G.adj(v))
            if (cc[w] == UNMARKED) dfs(G, w);
    }
    public int connected(int v, int w)
    { return cc[v] == cc[w]; }
}
```

component labels

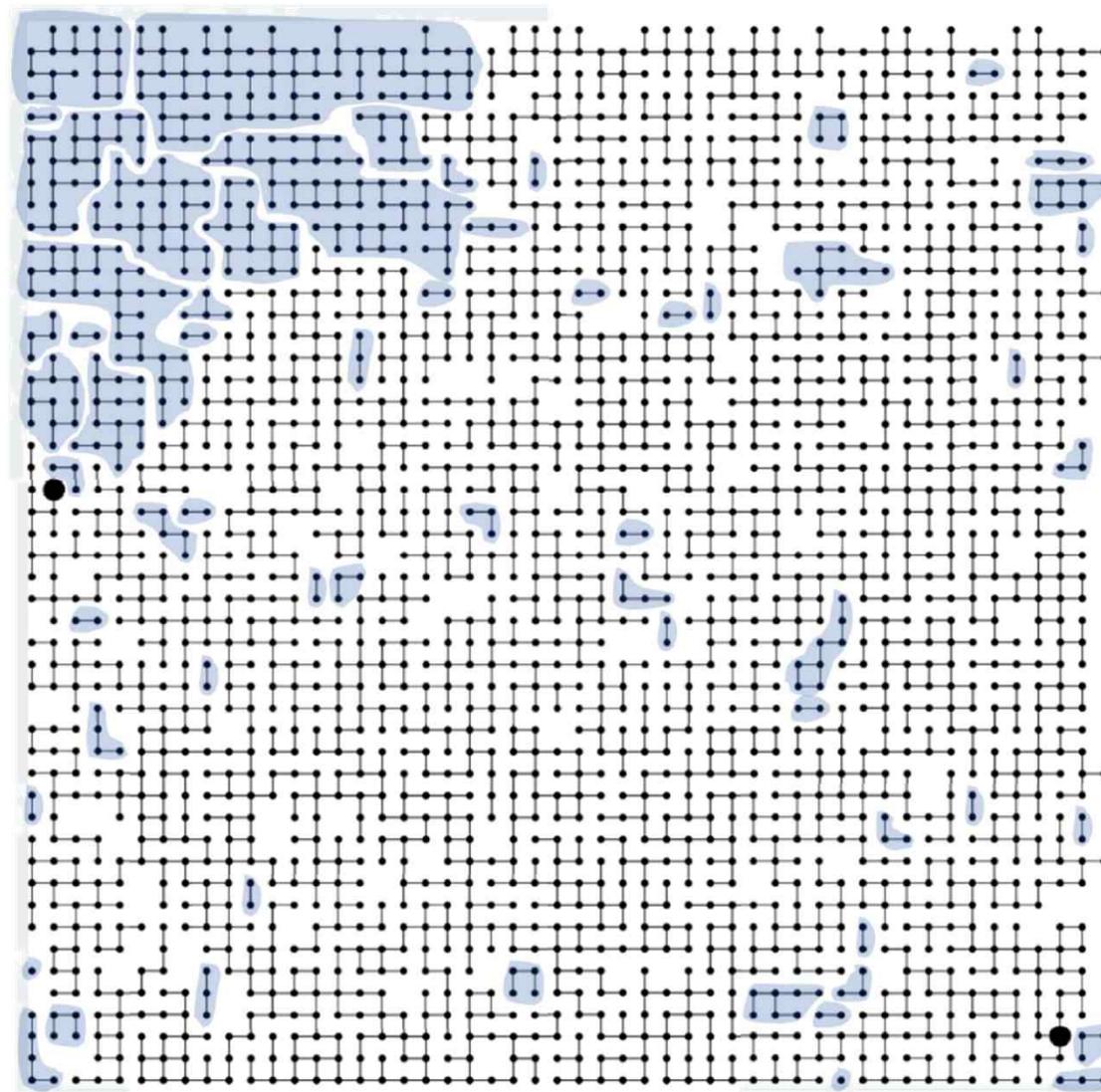
DFS for each component

standard DFS

constant-time connectivity query

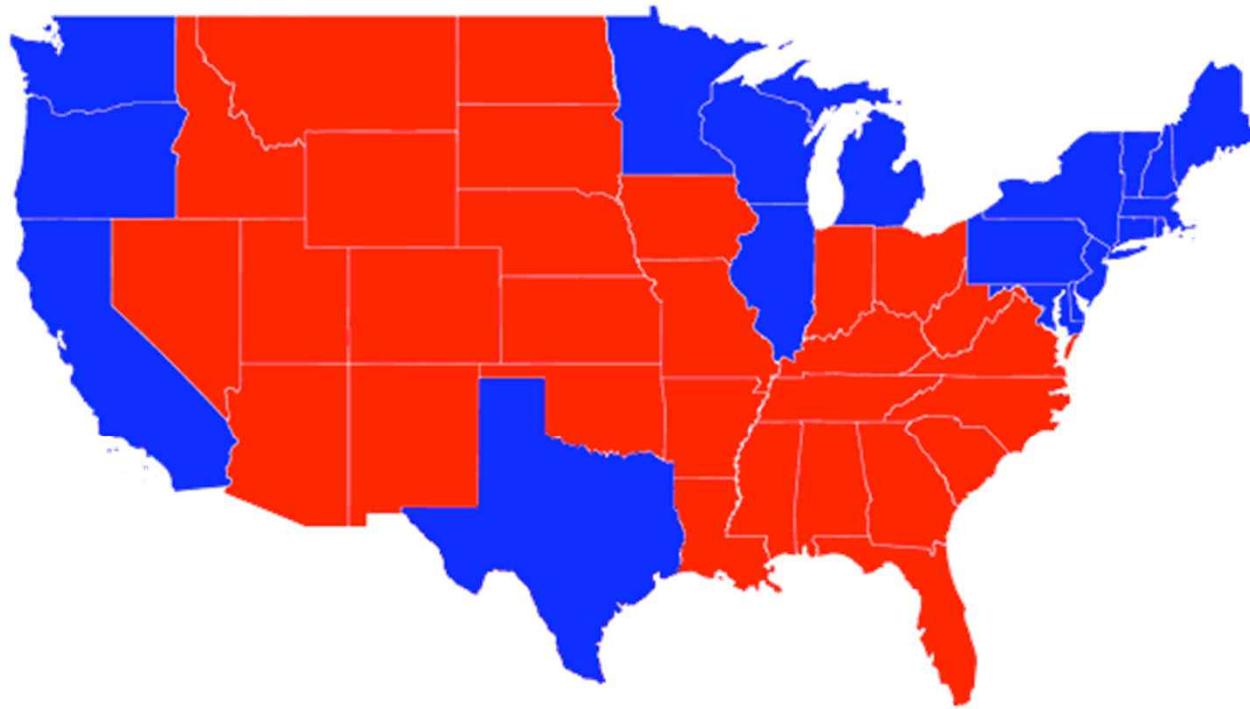
Connected Components

63 components



Connected components application: Image processing

- ❑ Goal. Read in a 2D color image and find regions of connected pixels that have the same color.

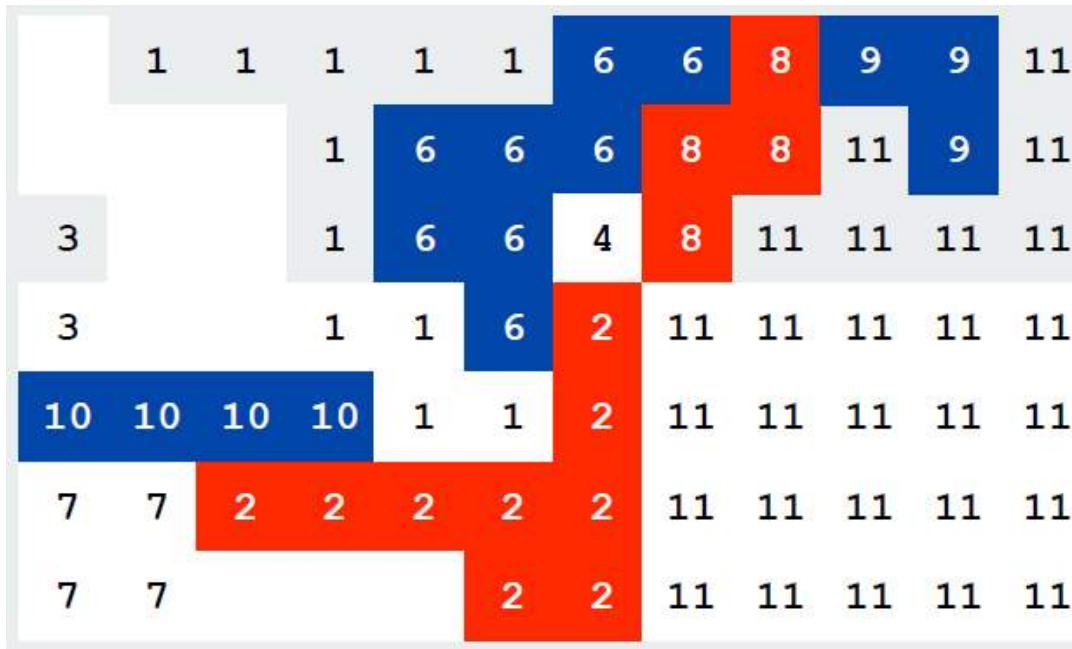


Input: scanned image

Output: number of red and blue states

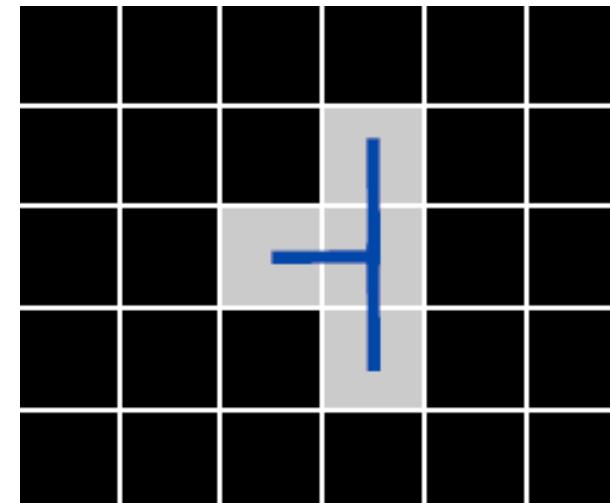
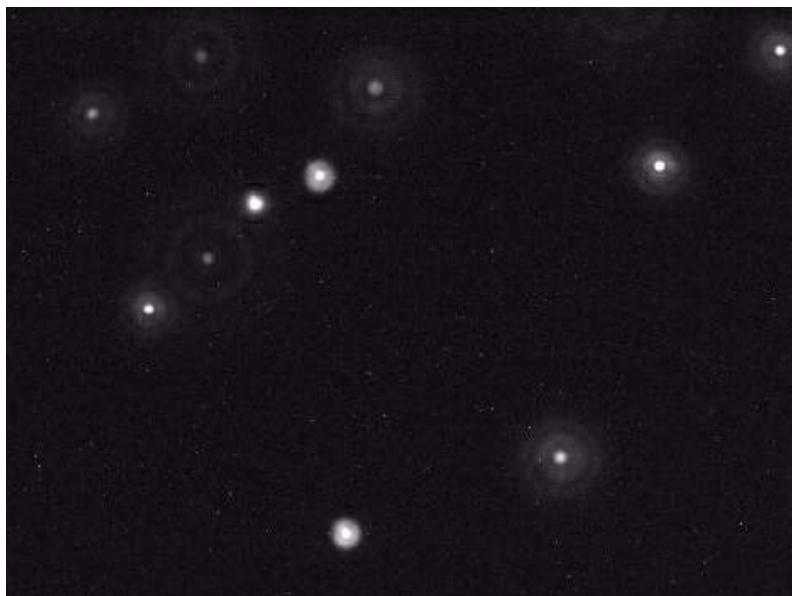
Connected components application: Image Processing

- **Goal.** Read in a 2D color image and find regions of connected pixels that have the same color.
- **Efficient algorithm.**
 - ◆ Connect each pixel to neighboring pixel if same color.
 - ◆ Find connected components in resulting graph.



Connected components application: Particle detection

- Particle detection. Given grayscale image of particles, identify "blobs."
 - ◆ Vertex: pixel.
 - ◆ Edge: between two adjacent pixels with grayscale value ≥ 70 .
 - ◆ Blob: connected component of 20-30 pixels.



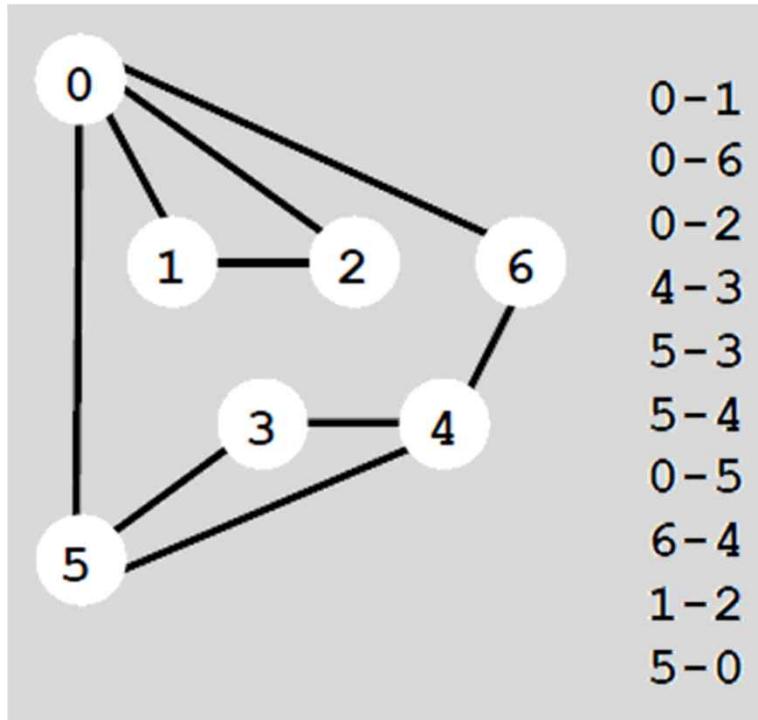
- Particle tracking. Track moving particles over time.

10. Undirected Graphs

- Graph API
- maze exploration
- depth-first search
- breadth-first search
- connected components
- challenges

Graph-processing challenges:

- Problem 4: Find a path from s to t that uses every edge
- Assumptions: need to use each edge exactly once

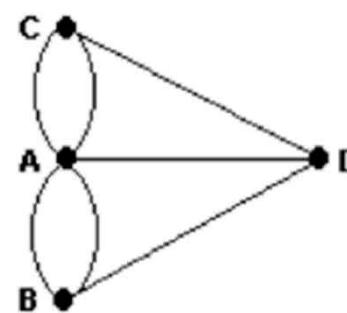
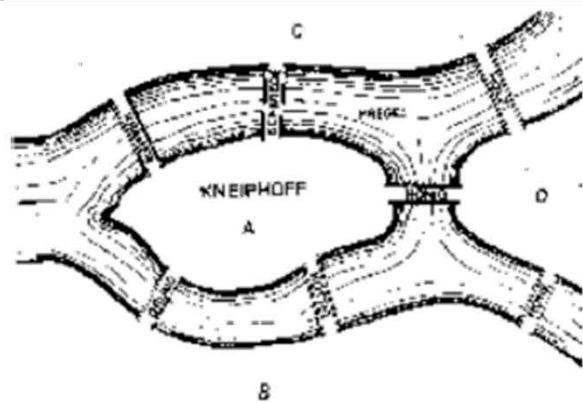


Bridges of Königsberg

□ The Seven Bridges of Königsberg. [Leonhard Euler 1736]

"... in Königsberg in Prussia, there is an island A, called the Kneiphof; the river which surrounds it is divided into two branches ... and these branches are crossed by seven bridges. Concerning these bridges, it was asked whether anyone could arrange a route in such a way that he could cross each bridge once and only once..."

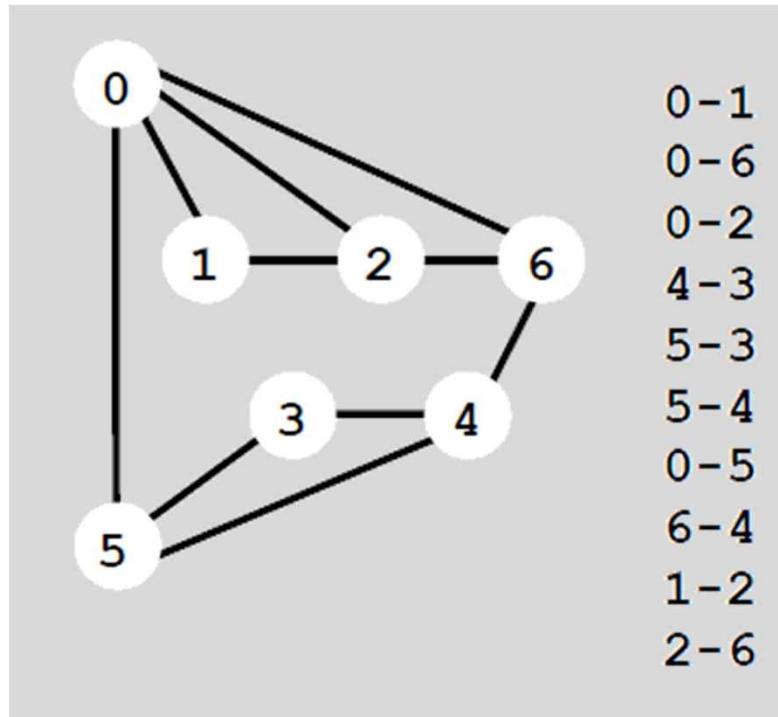
earliest application of graph theory or topology



- Euler tour. Is there a cyclic path that uses each edge exactly once?
- Answer. Yes iff connected and **all** vertices have **even degree**.
Tricky DFS-based algorithm to find path (see Algs in Java).

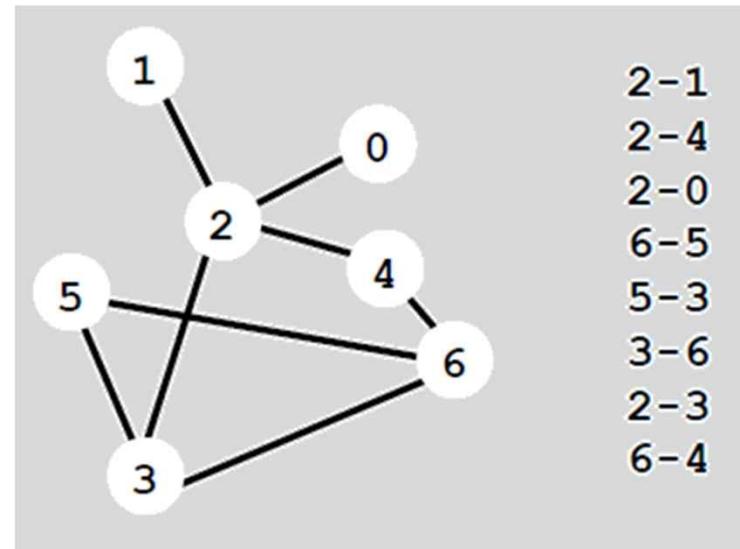
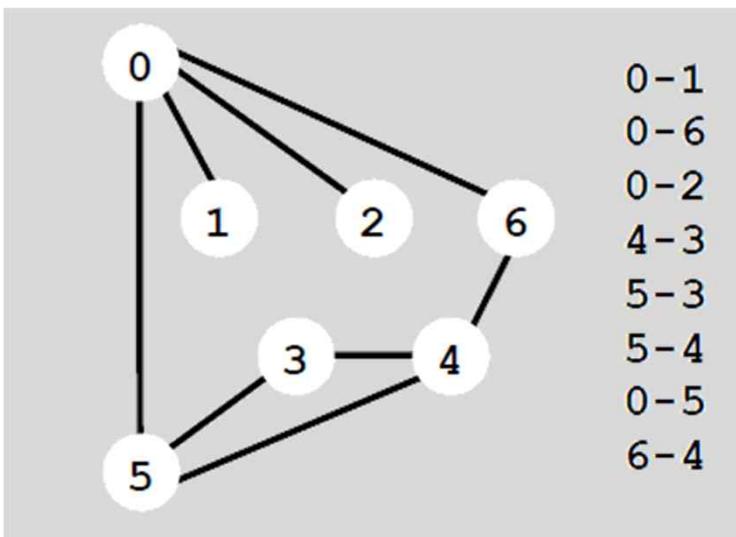
Graph-processing challenges:

- Problem 5 Find a path from s to t that visits every vertex
- Assumptions: need to visit each vertex exactly once



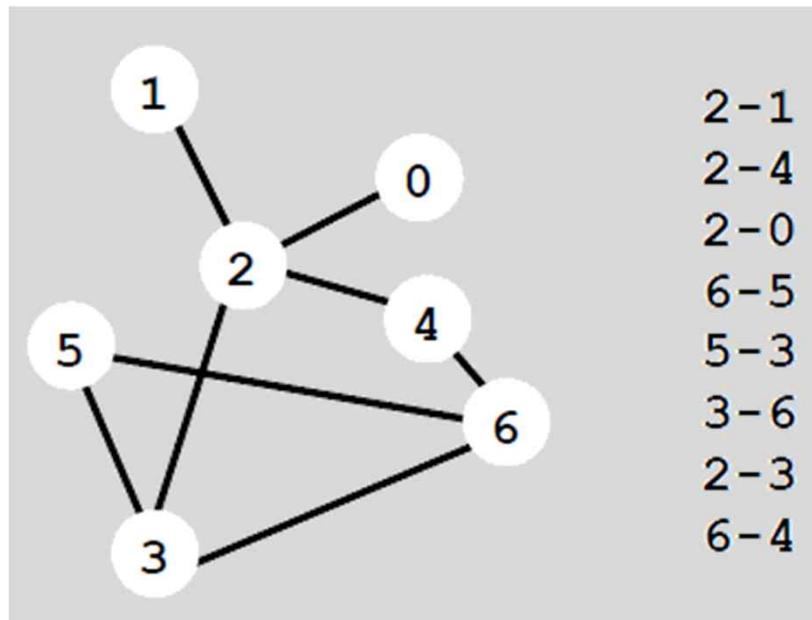
Graph-processing challenges:

- Problem 6: Are two graphs identical except for vertex names?



Graph-processing challenges:

- Problem 7: Can you lay out a graph in the plane without crossing edges?



Data Structures & Algorithms

11. Directed Graphs

- digraph search
- transitive closure
- topological sort
- strong components



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

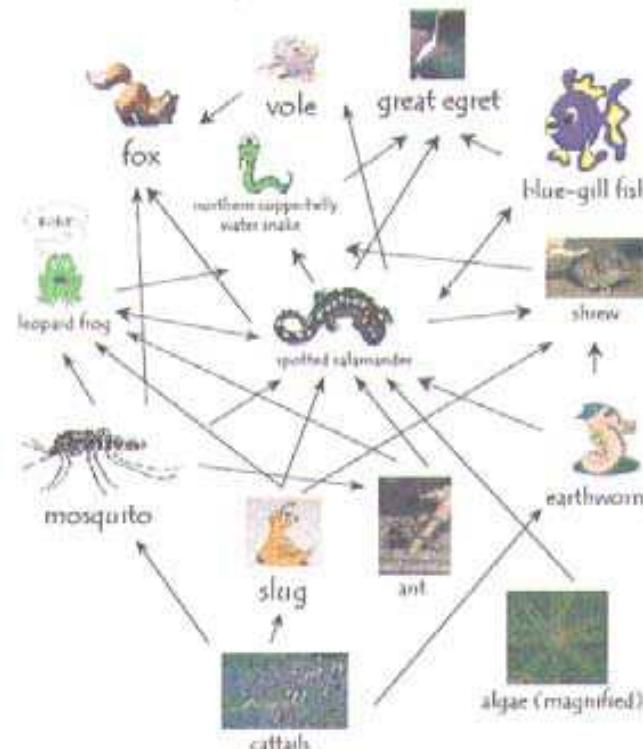
Directed Graphs (Digraphs)

- Set of objects with oriented pairwise connections.

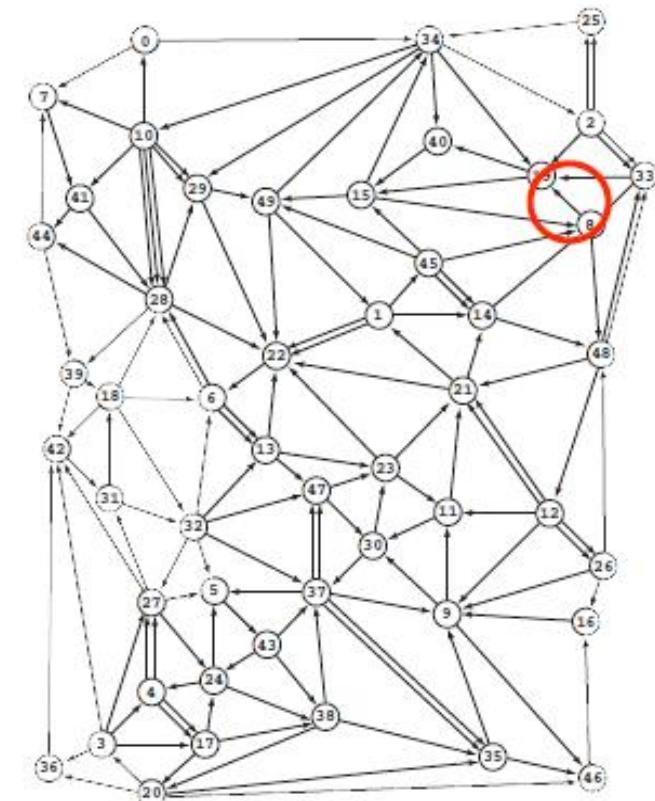
one-way streets in a map



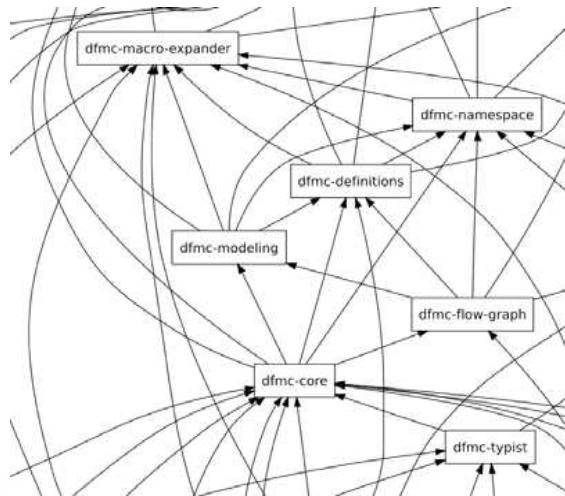
prey-predator relationships



hyperlinks connecting web pages



dependencies in software modules

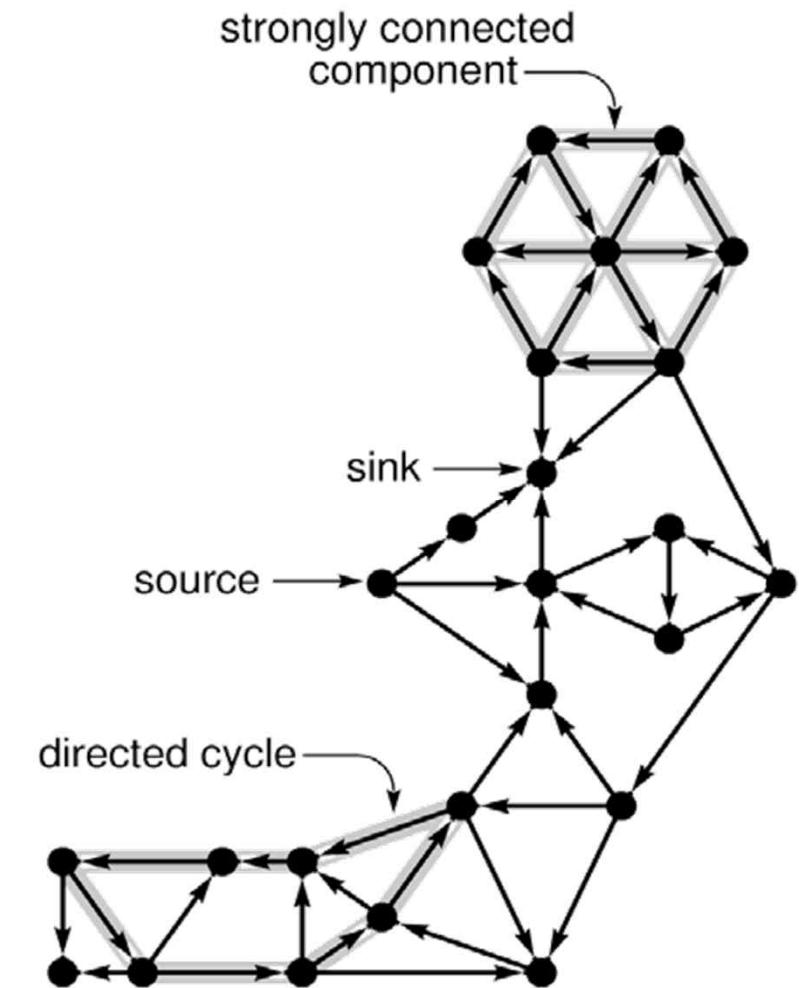


Digraph Applications

digraph	vertex	edge
financial	stock, currency	transaction
transportation	street intersection, airport	highway, airway route
scheduling	task	precedence constraint
WordNet	synset	hypernym
Web	web page	hyperlink
game	board position	legal move
telephone	person	placed call
food web	species	predator-prey relation
infectious disease	person	infection
citation	journal article	citation
object graph	object	pointer
inheritance hierarchy	class	inherits from
control flow	code block	jump

Some Digraph Problems

- **Transitive closure.** Is there a directed path from v to w ?
- **Strong connectivity.** Are all vertices mutually reachable?
- **Topological sort.** Can you draw the digraph so that all edges point from left to right?
- **PERT/CPM.** Given a set of tasks with precedence constraints, how can we best complete them all?
- **Shortest path.** Find best route from s to t in a weighted digraph
- **PageRank.** What is the importance of a web page?



Digraph Representations

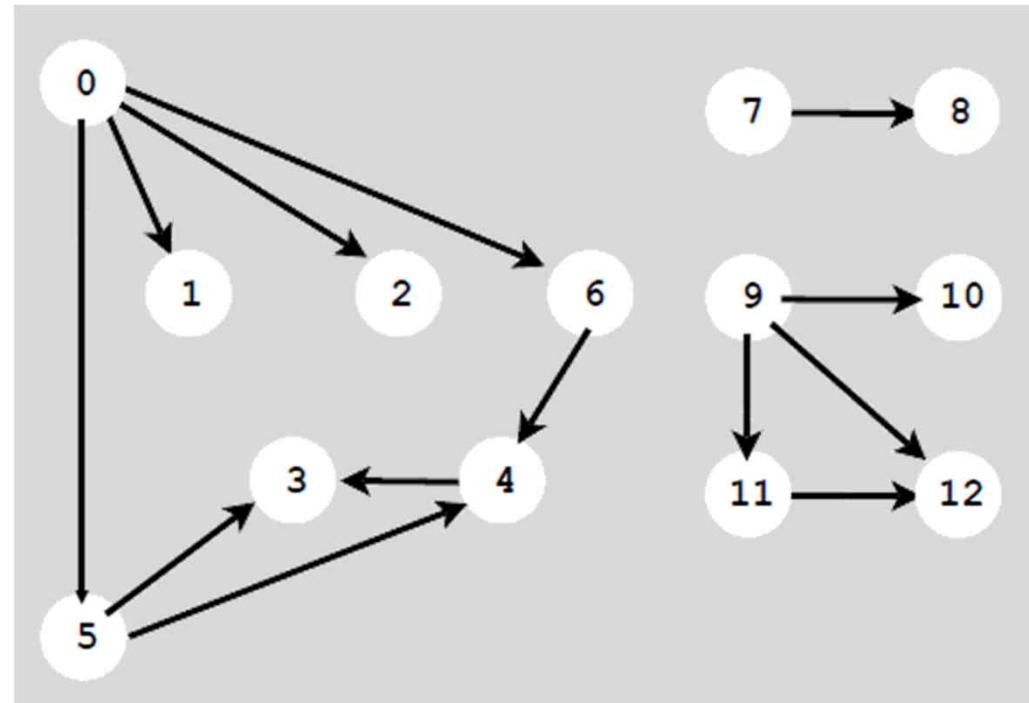
□ Vertices

- ◆ this lecture: use integers between 0 and V-1.
- ◆ real world: convert between names and integers with symbol table.

□ Edges: four easy options

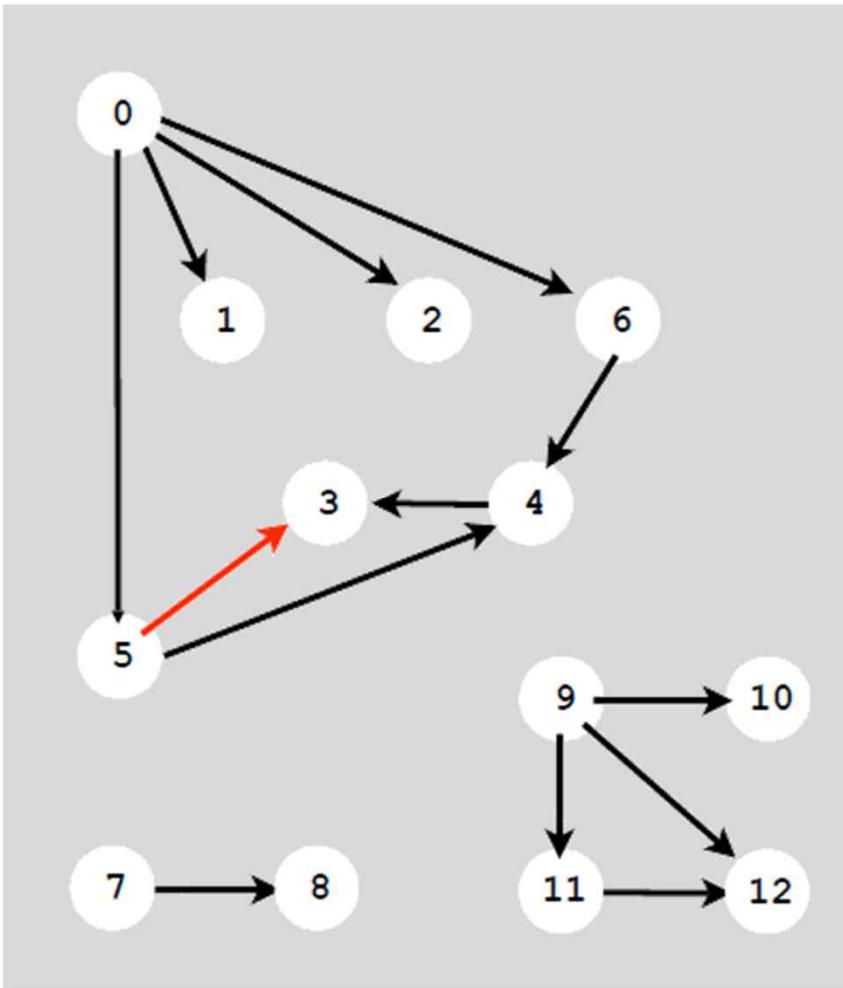
- ◆ list of vertex pairs
- ◆ vertex-indexed adjacency arrays (adjacency matrix)
- ◆ vertex-indexed adjacency lists
- ◆ vertex-indexed adjacency SETs

Same as undirected graph
BUT
orientation of edges is significant.



Adjacency Matrix Digraph Representation

- Maintain a two-dimensional $V \times V$ boolean array.
- For each edge $v \rightarrow w$ in graph: $\text{adj}[v][w] = \text{true}$.

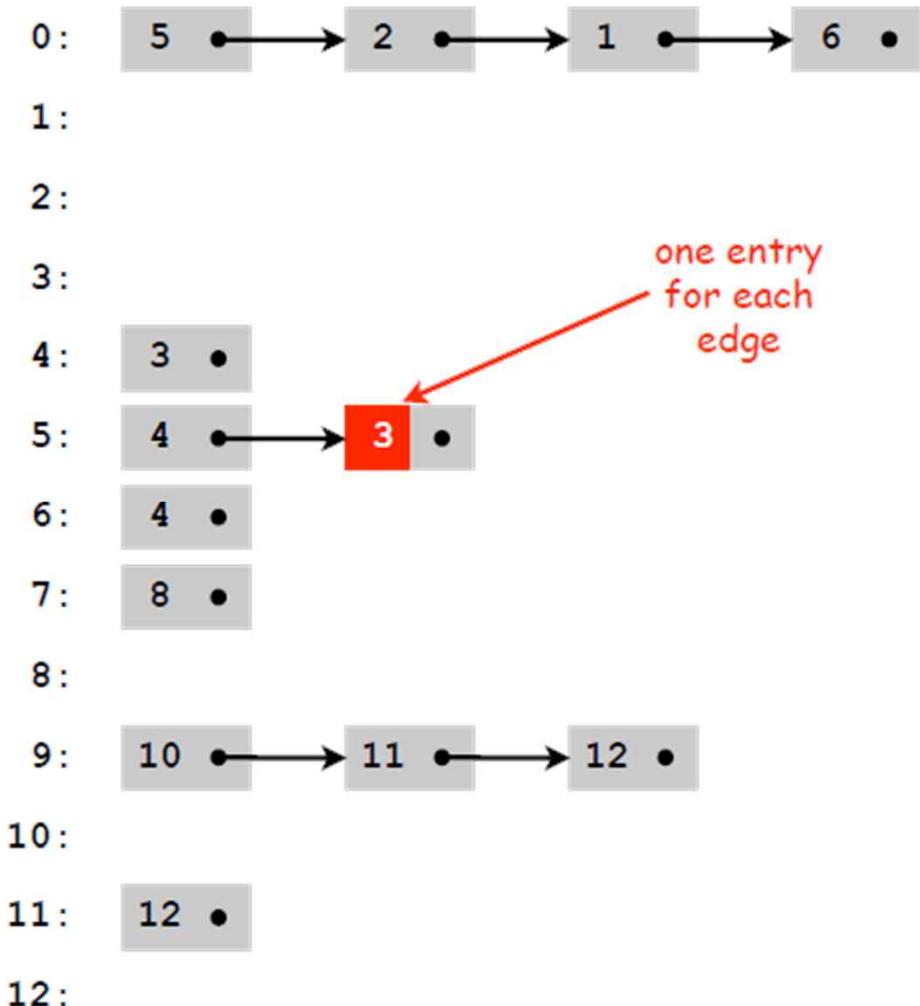
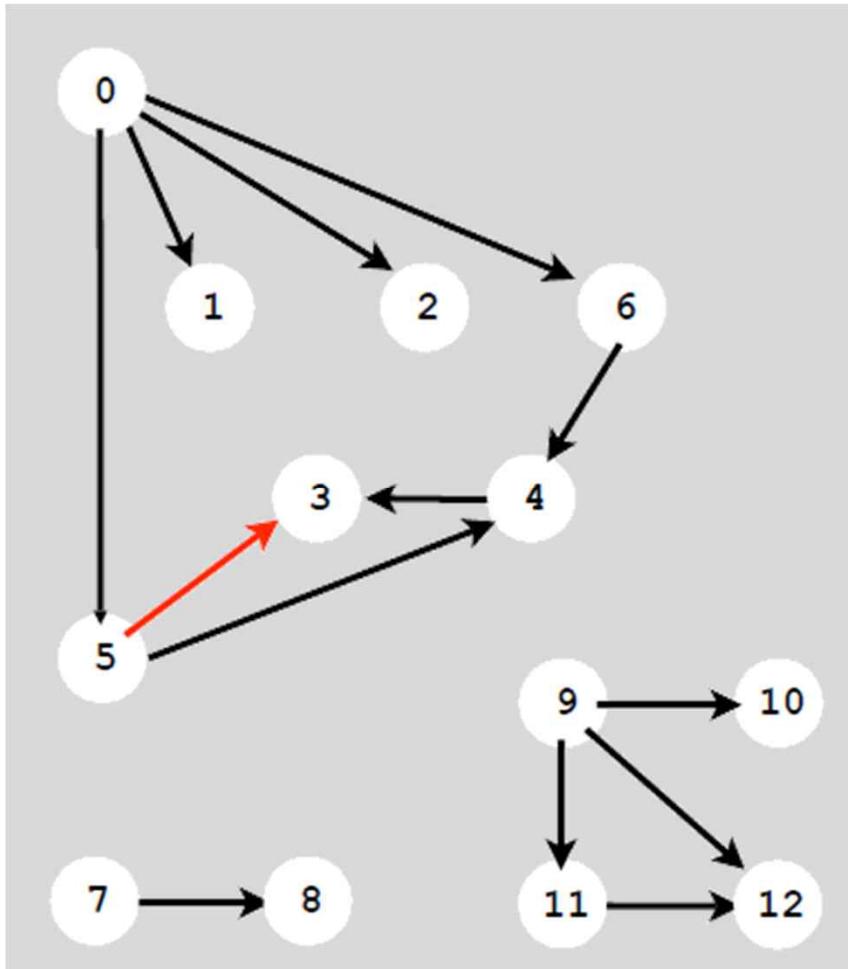


one entry for each edge

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	0	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0

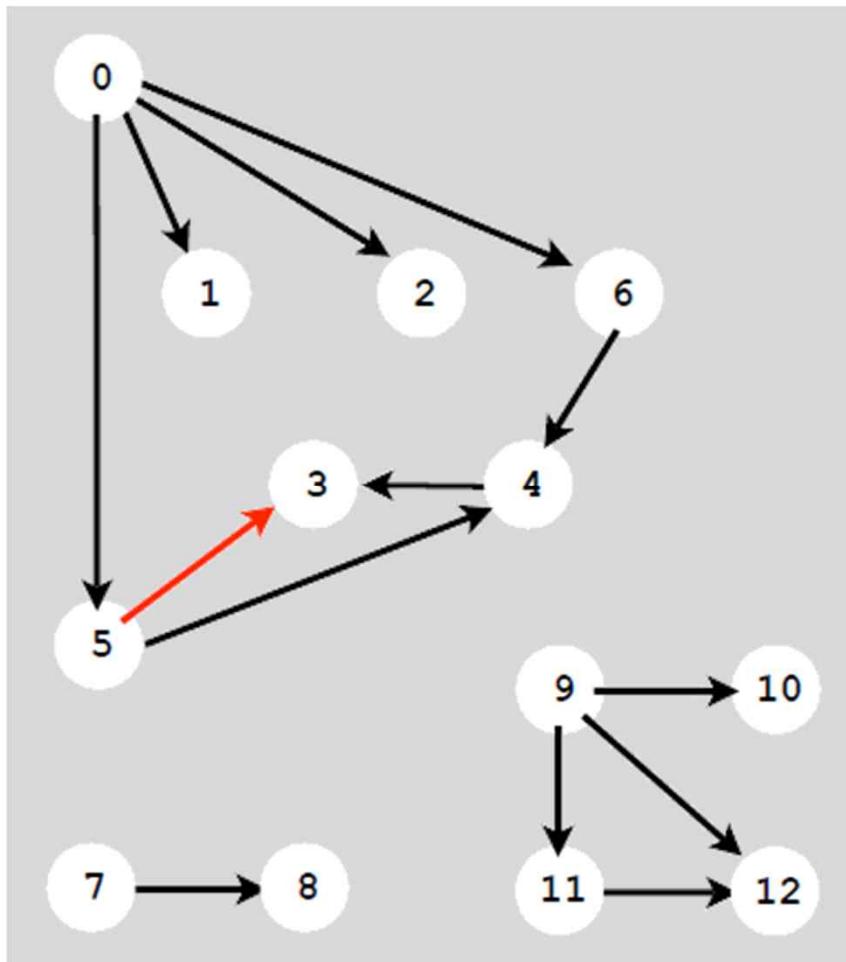
Adjacency-List Digraph Representation

- Maintain vertex-indexed array of lists.



Adjacency-SET Digraph Representation

- Maintain vertex-indexed array of SETs.



0:	{ 1 2 5 6 }
1:	{ }
2:	{ }
3:	{ }
4:	{ 3 }
5:	{ 3 4 }
6:	{ 4 }
7:	{ 8 }
8:	{ }
9:	{ 10 11 12 }
10:	{ }
11:	{ 12 }
12:	{ }

one entry
for each
edge

Adjacency-SET Digraph Representation: Java Implementation

Same as Graph, but only insert **one copy** of each edge.

```
public class Digraph
{
    private int V;
    private SET<Integer>[] adj;
```

adjacency
SETS

```
public Digraph(int V)
{
    this.V = V;
    adj = (SET<Integer>[]) new SET[V];
    for (int v = 0; v < V; v++)
        adj[v] = new SET<Integer>();
```

create empty
V-vertex graph

```
public void addEdge(int v, int w)
{
    adj[v].add(w);
```

add edge from v to w
(Graph also has adj[w].add[v])

```
public Iterable<Integer> adj(int v)
{
    return adj[v];
```

iterable SET for
v's neighbors



Digraph Representations

- Digraphs are abstract mathematical objects, BUT
 - ◆ ADT implementation requires specific representation.
 - ◆ Efficiency depends on matching algorithms to representations.

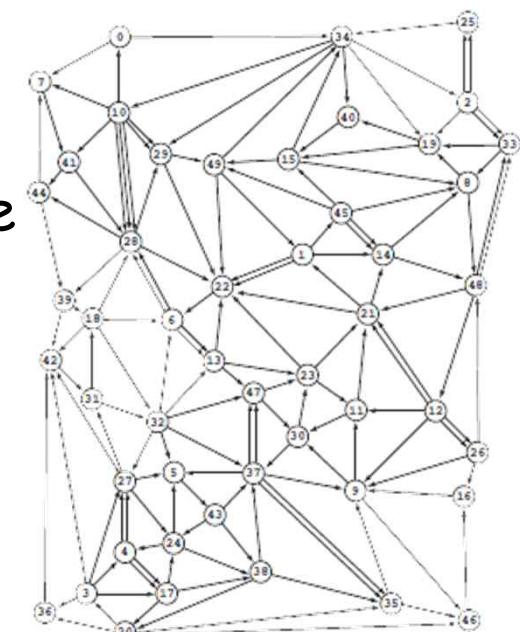
representation	space	edge between v and w?	iterate over edges incident to v?
list of edges	E	E	E
adjacency matrix	V^2	1	V
adjacency list	$E + V$	$\text{degree}(v)$	$\text{degree}(v)$
adjacency SET	$E + V$	$\log(\text{degree}(v))$	$\text{degree}(v)$

- In practice: Use adjacency SET representation

- ◆ Take advantage of proven technology
- ◆ Real-world digraphs tend to be "sparse"
[huge number of vertices, small average vertex degree]
- ◆ Algs all based on iterating over edges incident to v.

Typical Digraph Application: Google's PageRank Algorithm

- Goal. Determine which web pages on Internet are important.
- Solution. Ignore keywords and content, focus on hyperlink structure.
- Random surfer model.
 - ◆ Start at random page.
 - ◆ With probability 0.85, randomly select a hyperlink to visit next; with probability 0.15, randomly select any page.
 - ◆ PageRank = proportion of time random surfer spends on each page.
- Solution 1: Simulate random surfer for a long time.
- Solution 2: Compute ranks directly until they converge
- Solution 3: Compute eigenvalues of adjacency matrix!
- None feasible without sparse digraph representation
- Every square matrix is a weighted digraph



Algorithms

11. Directed Graphs

- **digraph search**
- **transitive closure**
- **topological sort**
- **strong components**

Digraph Application: Program Control-Flow Analysis

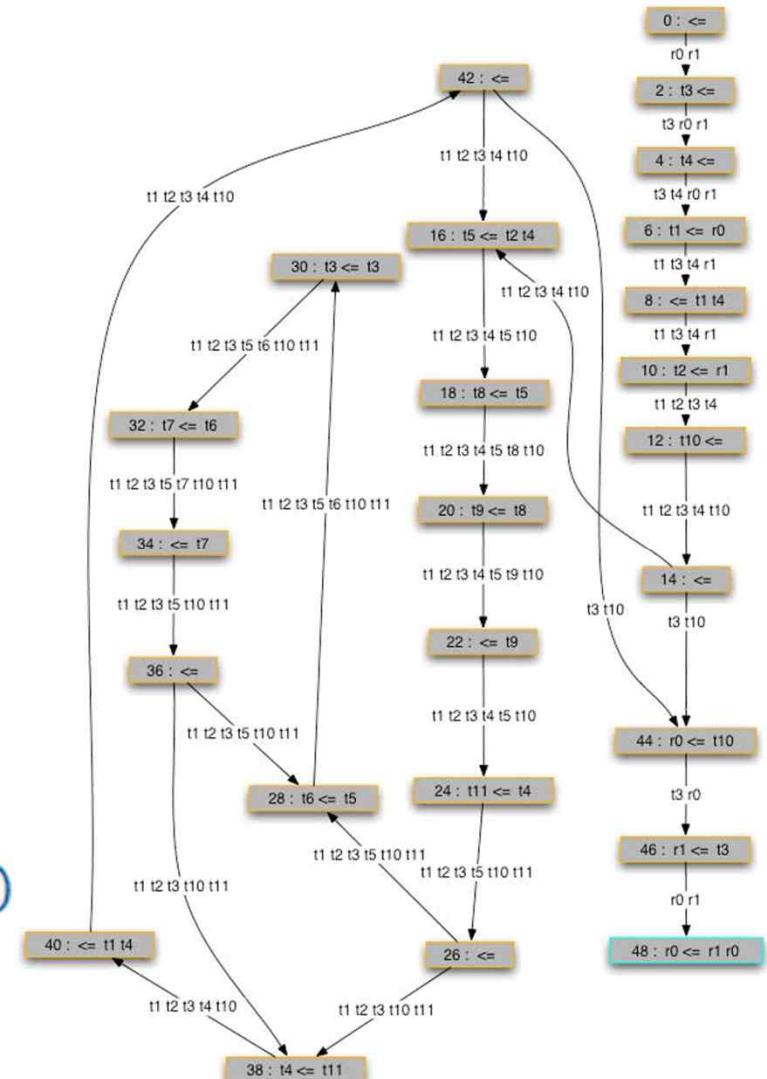
- Every **program** is a digraph (instructions connected to possible successors)

- Dead code elimination.
- Find (and remove) **unreachable** code

↑
can arise from compiler optimization (or bad code)

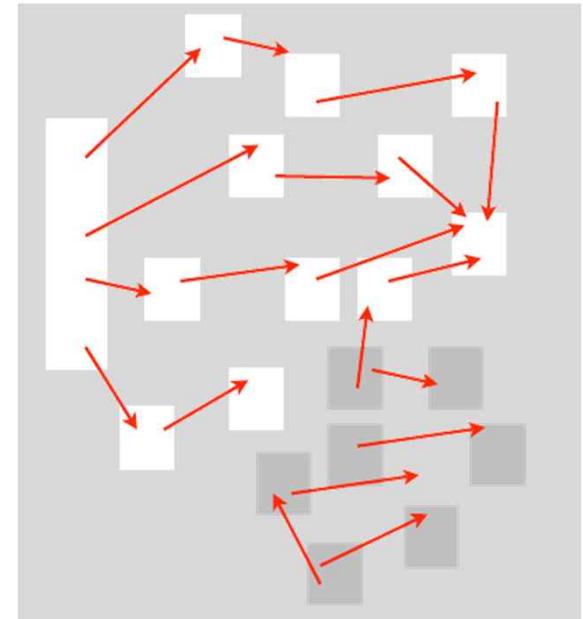
- Infinite loop detection.
- Determine whether exit is **unreachable**

↑
can't detect all possible infinite loops (halting problem)



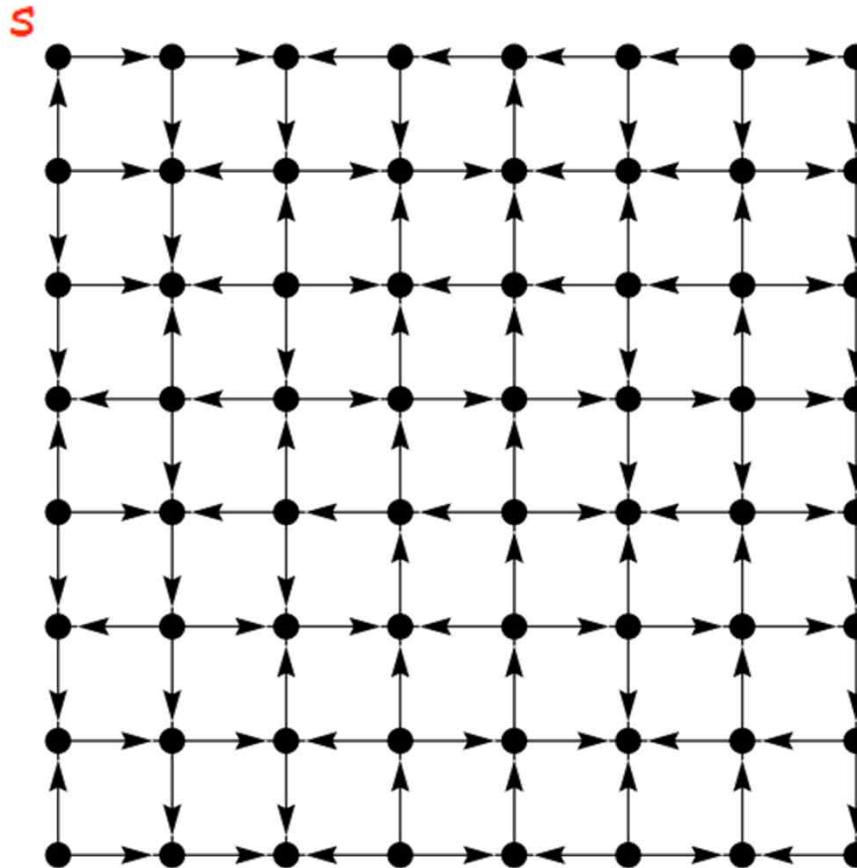
Digraph Application: Mark-Sweep Garbage Collector

- Every **data structure** is a digraph (objects connected by references)
- **Roots.** Objects known to be directly accessible by program (e.g., stack).
- **Reachable objects.**
 - ◆ Objects indirectly accessible by program
 - ◆ starting at a root and
 - ◆ following a chain of pointers.
- **Mark-sweep algorithm.** [McCarthy, 1960]
 - ◆ Mark: mark all reachable objects.
 - ◆ Sweep: if object is unmarked, it is garbage, so add to free list.
- **Memory cost:** Uses 1 extra mark bit per object, plus DFS stack.



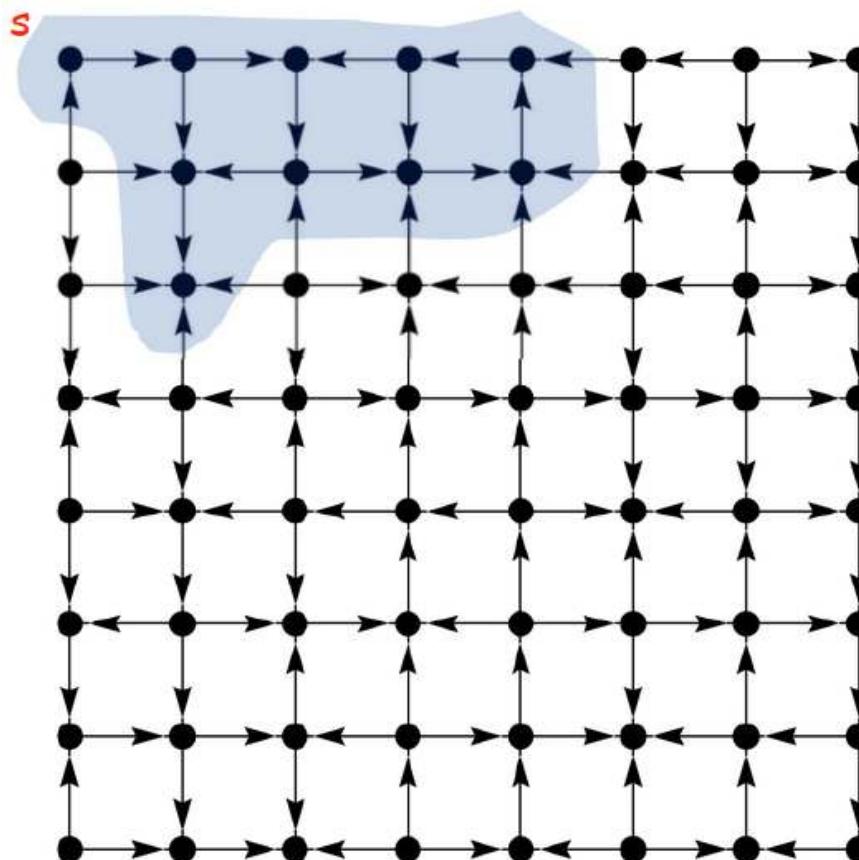
Reachability

- Goal. Find all vertices reachable from s along a directed path.



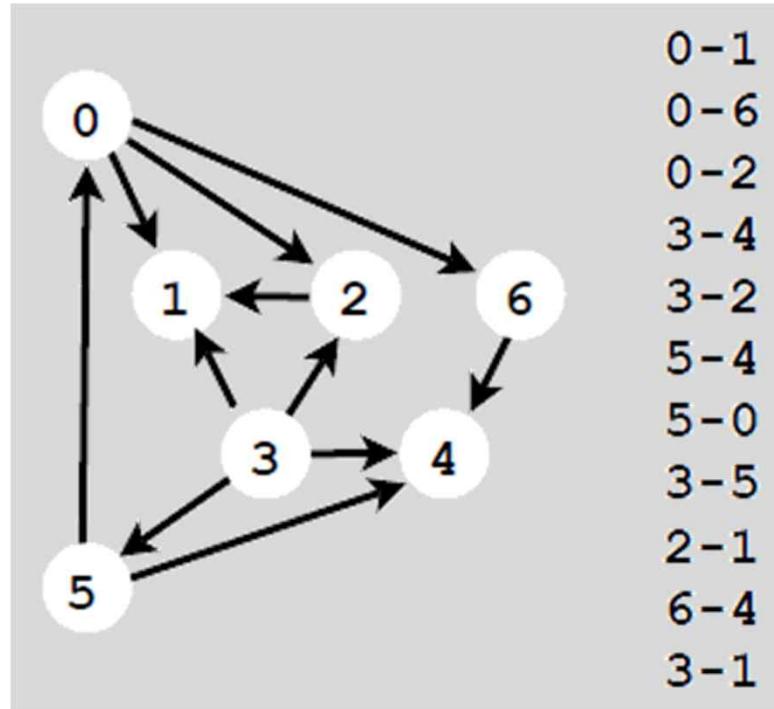
Reachability

- Goal. Find all vertices reachable from s along a directed path.



Digraph-Processing Challenge:

- Problem 1: Mark all vertices reachable from a given vertex.



0-1
0-6
0-2
3-4
3-2
5-4
5-0
3-5
2-1
6-4
3-1

Depth-First Search in Digraphs

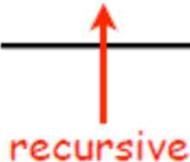
- Same method as for undirected graphs
- Every undirected graph **is** a digraph
 - ◆ happens to have edges in both directions
 - ◆ DFS is a digraph algorithm

DFS (to visit a vertex v)

Mark v as visited.

Visit all unmarked vertices w adjacent to v .

recursive



Depth-First Search (Single-Source Reachability)

- Identical to undirected version (substitute Digraph for Graph).

```
public class DFSearcher
{
    private boolean[] marked; ← true if connected to s

    public DFSearcher(Digraph G, int s)
    {
        marked = new boolean[G.V()]; ← constructor marks vertices connected to s
        dfs(G, s);
    }

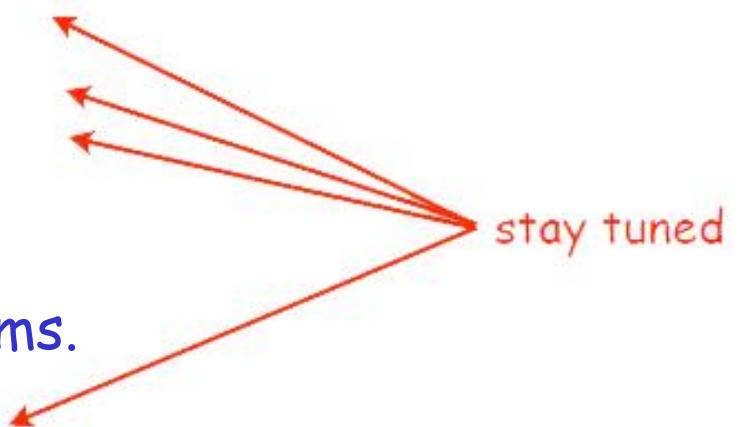
    private void dfs(Digraph G, int v)
    {
        marked[v] = true;
        for (int w : G.adj(v)) ← recursive DFS does the work
            if (!marked[w]) dfs(G, w);
    }

    public boolean isReachable(int v)
    {
        return marked[v]; ← client can ask whether any vertex is connected to s
    }
}
```

Depth-First Search (DFS)

- DFS enables direct solution of simple digraph problems.

- ✓ ♦ Reachability.
 - ♦ Cycle detection
 - ♦ Topological sort
 - ♦ Transitive closure.
 - ♦ Is there a path from s to t ?



- Basis for solving difficult digraph problems.

- ♦ Directed Euler path.
 - ♦ Strong connected components.

Breadth-First Search in Digraphs

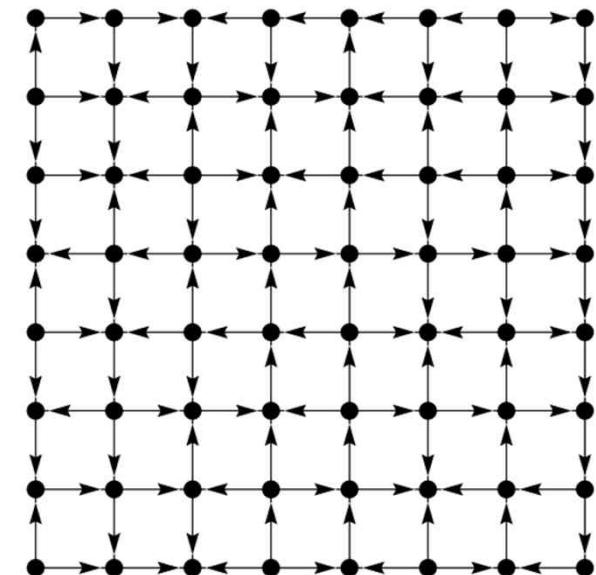
- Same method as for undirected graphs
- Every undirected graph **is** a digraph
 - ◆ happens to have edges in both directions
 - ◆ BFS is a digraph algorithm

BFS (from source vertex s)

Put s onto a FIFO queue.

Repeat until the queue is empty:

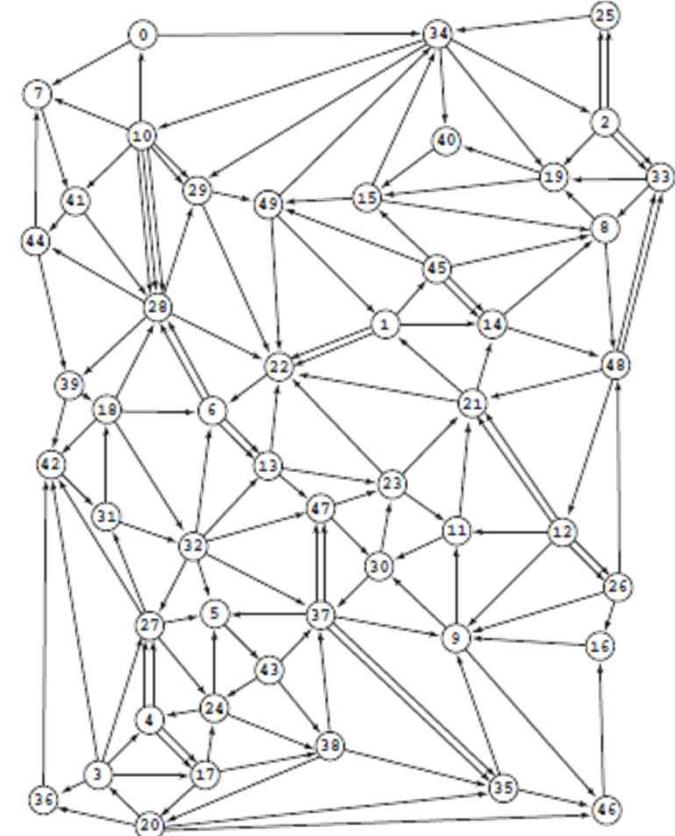
- remove the least recently added vertex v
- add each of v 's unvisited neighbors to the queue and mark them as visited.



Visits vertices in **increasing** distance from s

Digraph BFS Application: Web Crawler

- The **internet** is a digraph
- **Goal.** Crawl Internet, starting from some root website.
- **Solution.** BFS with implicit graph.
- **BFS.**
 - ◆ Start at some root website
(say `http://www.inu.ac.kr.`).
 - ◆ Maintain a **Queue** of websites to explore.
 - ◆ Maintain a **SET** of discovered websites.
 - ◆ Dequeue the next website
and enqueue websites to which it links
(provided you haven't done so before).
- **Q.** Why not use DFS?
- **A.** Internet is not fixed (some pages generate new ones when visited)
 - subtle point: think about it!



Web Crawler: BFS-Based Java Implementation

```
Queue<String> q = new Queue<String>();           ← queue of sites to crawl
SET<String> visited = new SET<String>();          ← set of visited sites

String s = "http://www.inu.ac.kr";
q.enqueue(s);
visited.add(s);

while (!q.isEmpty())
{
    String v = q.dequeue();                         ← read in raw html for next site in queue
    System.out.println(v);
    In in = new In(v);
    String input = in.readAll();

    String regexp = "http://((\\w+\\.)*)*(\\w+)";   ← http://xxx.yyy.zzz
    Pattern pattern = Pattern.compile(regexp);
    Matcher matcher = pattern.matcher(input);
    while (matcher.find())
    {
        String w = matcher.group();
        if (!visited.contains(w))
        {
            visited.add(w);                      ← if unvisited, mark as visited
            q.enqueue(w);                      ← and put on queue
        }
    }
}
```

start crawling from s

use regular expression to find all URLs in site

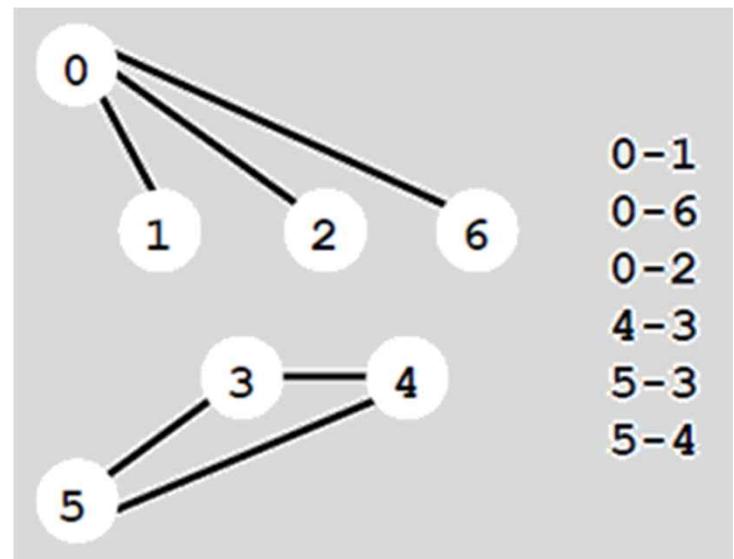
if unvisited, mark as visited and put on queue

11. Directed Graphs

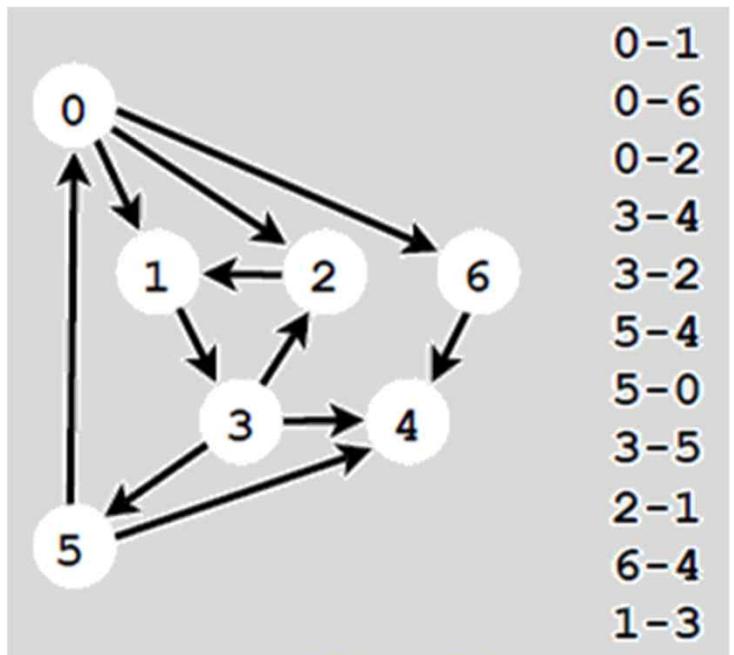
- digraph search
- transitive closure
- topological sort
- strong components

Graph-Processing Challenge

- Problem U1: Is there a path from s to t ?
- Goals: linear $\sim(V + E)$ preprocessing time
constant query time



- Problem D1: Is there a directed path from s to t ?
- Goals: linear $\sim(V + E)$ preprocessing time
constant query time

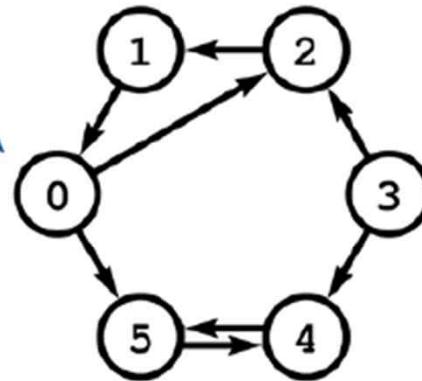


Transitive Closure

- The transitive closure of G has an directed edge from v to w if there is a directed path from v to w in G

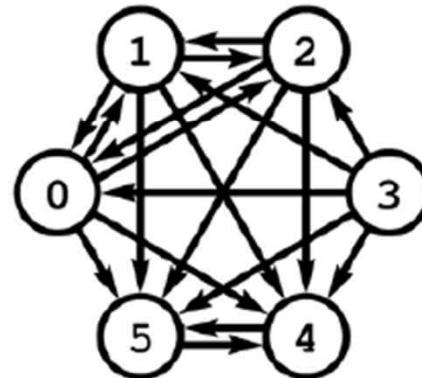
graph is usually sparse

G



	0	1	2	3	4	5
0	1	0	1	0	0	1
1	1	1	0	0	0	0
2	0	1	1	0	0	0
3	0	0	1	1	1	0
4	0	0	0	0	1	1
5	0	0	0	0	1	1

Transitive closure
of G



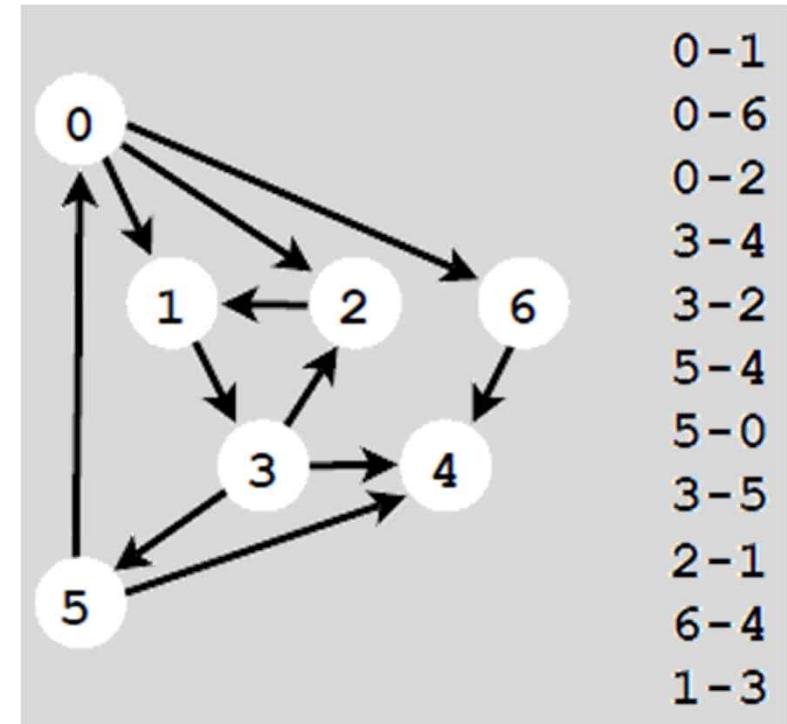
	0	1	2	3	4	5
0	1	1	1	0	1	1
1	1	1	1	0	1	1
2	1	1	1	0	1	1
3	1	1	1	1	1	1
4	0	0	0	0	1	1
5	0	0	0	0	1	1

TC is usually dense
so adjacency matrix
representation is OK

Digraph-Processing Challenge

- Problem 2: Is there a **directed** path from s to t ?
- Goals: $\sim V^2$ preprocessing time
constant query time

- Problem 2: Is there a **directed** path from s to t ?
- Goals: $\sim VE$ preprocessing time
($\sim V^3$ for dense digraphs)
 $\sim V^2$ space
constant query time



Transitive Closure: Java Implementation

- Use an array of DFSearcher objects, one for each row of transitive closure

```
public class TransitiveClosure
{
    private DFSearcher[] tc;

    public TransitiveClosure(Digraph G)
    {
        tc = new DFSearcher[G.V()];
        for (int v = 0; v < G.V(); v++)
            tc[v] = new DFSearcher(G, v);
    }

    public boolean reachable(int v, int w)
    {
        return tc[v].isReachable(w); ←
    }
}
```

```
public class DFSearcher
{
    private boolean[] marked;
    public DFSearcher(Digraph G, int s)
    {
        marked = new boolean[G.V()];
        dfs(G, s);
    }
    private void dfs(Digraph G, int v)
    {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w]) dfs(G, w);
    }
    public boolean isReachable(int v)
    {
        return marked[v];
    }
}
```

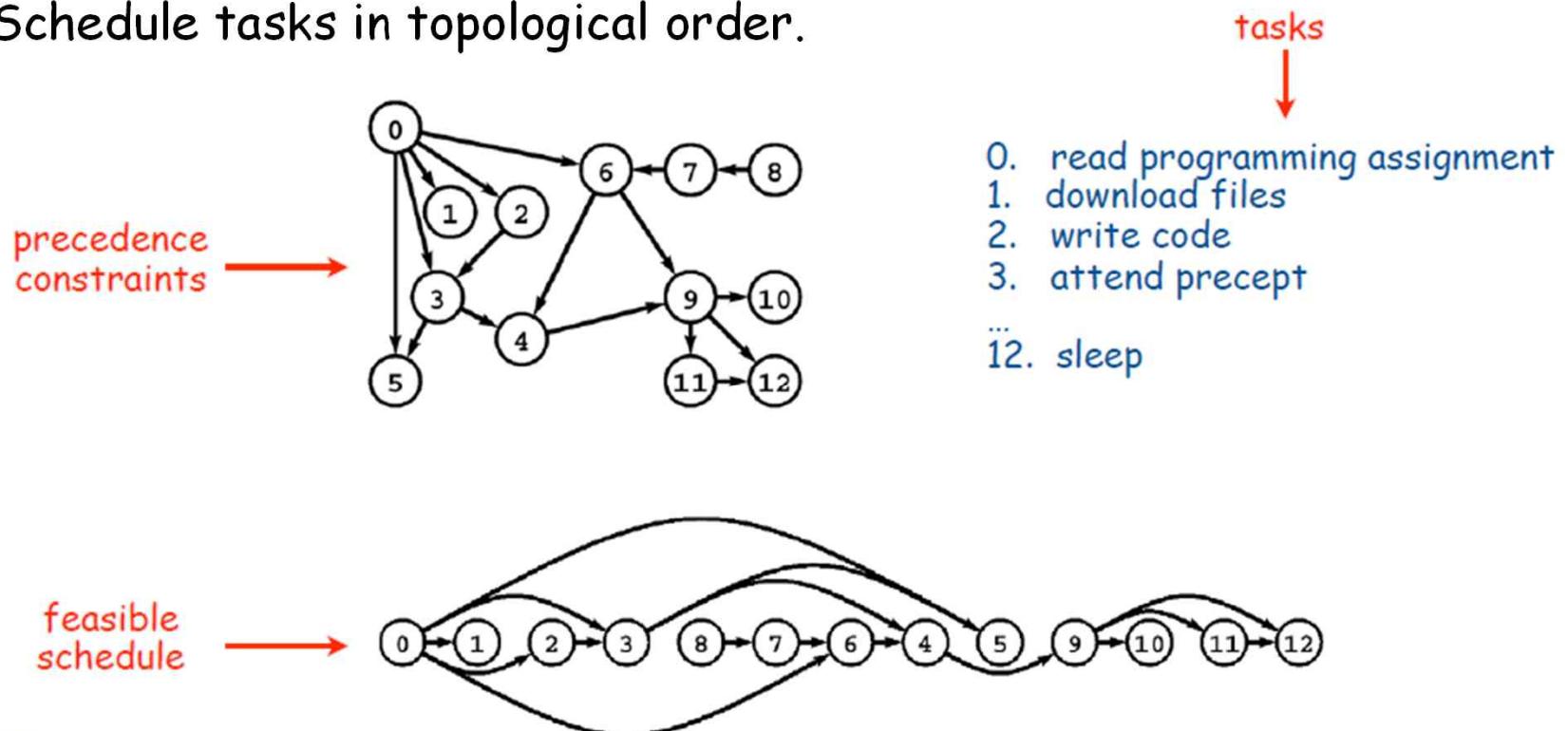
is there a directed path from v to w ?

11. Directed Graphs

- digraph search
- transitive closure
- topological sort
- strong components

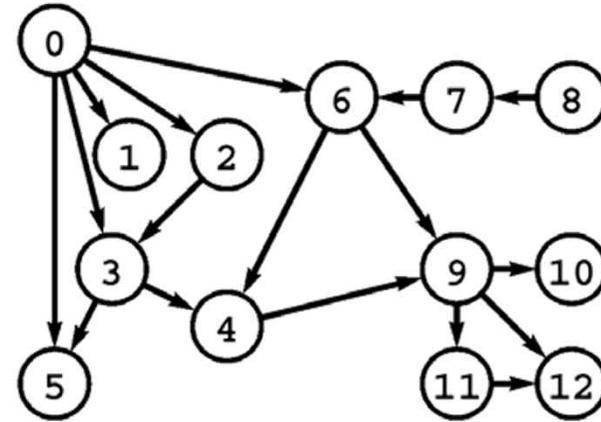
Digraph Application: Scheduling

- **Scheduling.** Given a set of tasks to be completed with precedence constraints, in what order should we schedule the tasks?
- **Graph model.**
 - ◆ Create a vertex v for each task.
 - ◆ Create an edge $v \rightarrow w$ if task v must precede task w .
 - ◆ Schedule tasks in topological order.

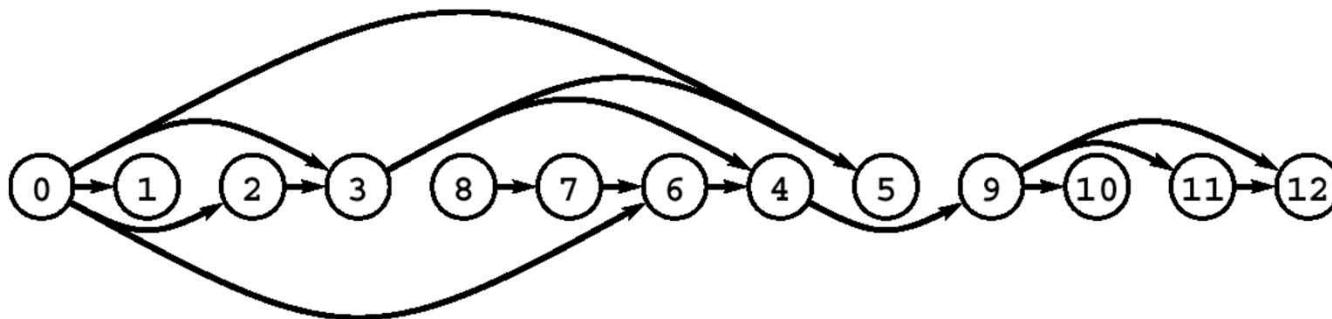


Topological Sort

- DAG. Directed **acyclic** graph.



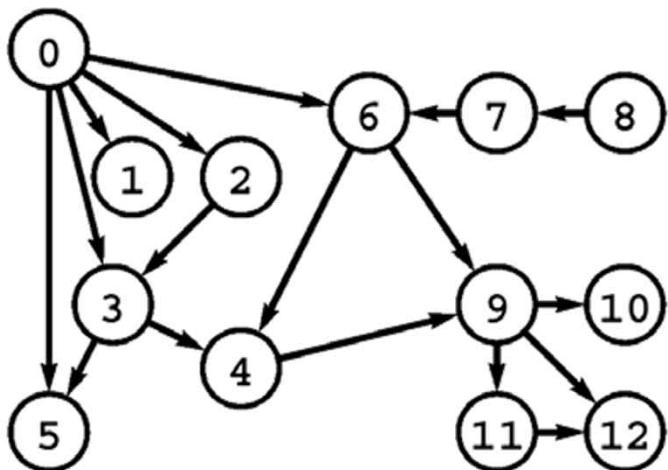
- Topological sort. Redraw DAG so all edges point left to right.



- Observation. Not possible if graph has a directed cycle.

Digraph-Processing Challenge

- Problem 3: Check that the digraph is a DAG.
If it is a DAG, do a topological sort.
- Goals: linear $\sim(V + E)$ preprocessing time
provide client with vertex iterator for topological order



0-1
0-6
0-2
0-5
2-3
4-9
6-4
6-9
7-6
8-7
9-10
9-11
9-12
11-12

Topological Sort in a DAG: Java Implementation

```
public class TopologicalSorter
{
    private int count;
    private boolean[] marked;
    private int[] ts;

    public TopologicalSorter(Digraph G)
    {
        marked = new boolean[G.V()];
        ts = new int[G.V()];
        count = G.V();
        for (int v = 0; v < G.V(); v++)
            if (!marked[v]) tsort(G, v);
    }

    private void tsort(Digraph G, int v)
    {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w]) tsort(G, w);
        ts[--count] = v;
    }
}
```

standard DFS
with 5
extra lines of code

add iterator that returns
ts[0], ts[1], ts[2]...

Seems easy? Missed by experts for a few decades

Topological Sort of a DAG: Trace

- “visit” means “call `tsort()`” and “leave” means “return from `tsort()`”

	marked[]	ts[]	adj SETs
visit 0:	1 0 0 0 0 0 0	0 0 0 0 0 0 0	0: 1 2 5
visit 1:	1 1 0 0 0 0 0	0 0 0 0 0 0 0	1: 4
visit 4:	1 1 0 0 1 0 0	0 0 0 0 0 0 0	2:
leave 4:	1 1 0 0 1 0 0	0 0 0 0 0 0 4	3: 2 4 5 6
leave 1:	1 1 0 0 1 0 0	0 0 0 0 0 1 4	4:
visit 2:	1 1 1 0 1 0 0	0 0 0 0 0 1 4	5: 2
leave 2:	1 1 1 0 1 0 0	0 0 0 0 2 1 4	6: 0 4
visit 5:	1 1 1 0 1 1 0	0 0 0 0 2 1 4	
check 2:	1 1 1 0 1 1 0	0 0 0 0 2 1 4	
leave 5:	1 1 1 0 1 1 0	0 0 0 5 2 1 4	
leave 0:	1 1 1 0 1 1 0	0 0 0 5 2 1 4	
check 1:	1 1 1 0 1 1 0	0 0 0 5 2 1 4	
check 2:	1 1 1 0 1 1 0	0 0 0 5 2 1 4	
visit 3:	1 1 1 1 1 1 0	0 0 0 5 2 1 4	
check 2:	1 1 1 1 1 1 0	0 0 0 5 2 1 4	
check 4:	1 1 1 1 1 1 0	0 0 0 5 2 1 4	
check 5:	1 1 1 1 1 1 0	0 0 0 5 2 1 4	
visit 6:	1 1 1 1 1 1 1	0 0 0 5 2 1 4	
leave 6:	1 1 1 1 1 1 1	0 6 0 5 2 1 4	
leave 3:	1 1 1 1 1 1 1	3 6 0 5 2 1 4	
check 4:	1 1 1 1 1 1 0	3 6 0 5 2 1 4	
check 5:	1 1 1 1 1 1 0	3 6 0 5 2 1 4	
check 6:	1 1 1 1 1 1 0	3 6 0 5 2 1 4	

Topological Sort in a DAG: Correctness Proof

Invariant:

`tsort(G, v)` visits all vertices
reachable from `v` with a directed path

□ Proof by induction:

- ◆ `w` marked: vertices reachable from `w`
are already visited
- ◆ `w` not marked: call `tsort(G, w)` to
visit the vertices reachable from `w`

Therefore, algorithm is correct
in placing `v` before all vertices visited
during call to `tsort(G, v)` just before returning.

```
public class TopologicalSorter
{
    private int count;
    private boolean[] marked;
    private int[] ts;

    public TopologicalSorter(Digraph G)
    {
        marked = new boolean[G.V()];
        ts = new int[G.V()];
        count = G.V();
        for (int v = 0; v < G.V(); v++)
            if (!marked[v]) tsort(G, v);
    }

    private void tsort(Digraph G, int v)
    {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w]) tsort(G, w);
        ts[--count] = v;
    }
}
```

- Q. How to tell whether the digraph has a cycle (is not a DAG)?
A. Use `TopologicalSorter` (exercise)

Hint: `marked` but
`v` not in `ts`, yet



Topological Sort Applications.

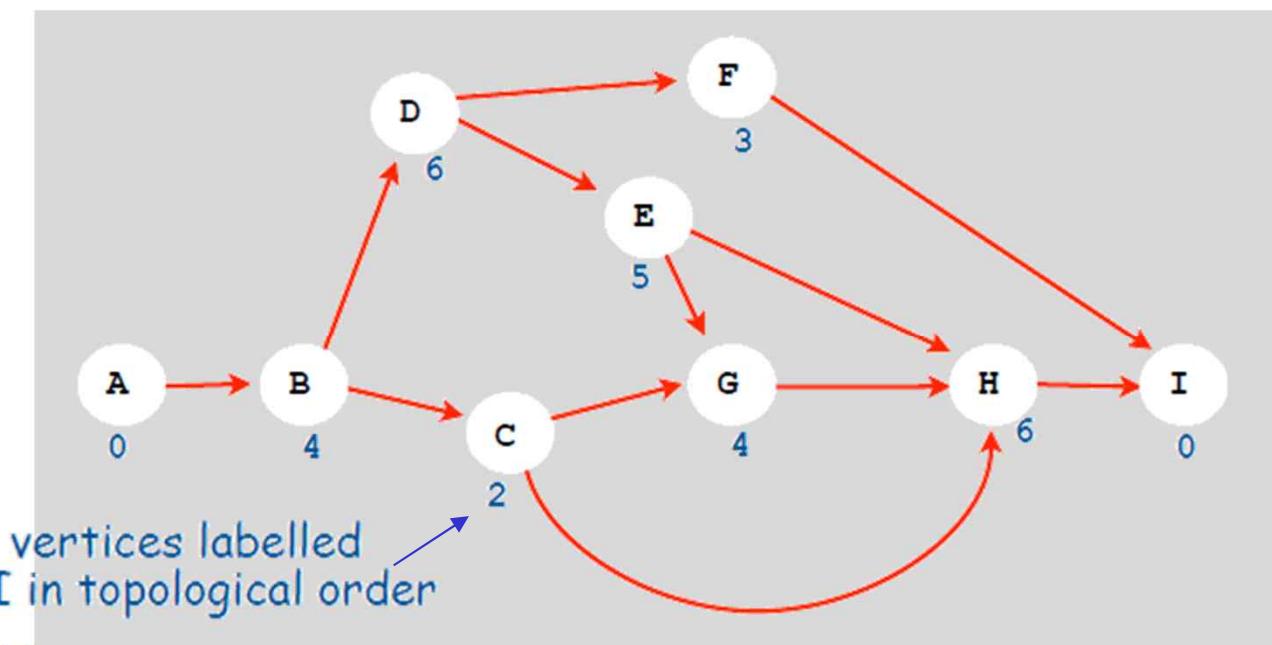
- ◆ Causalities.
- ◆ Compilation units.
- ◆ Class inheritance.
- ◆ Course prerequisites.
- ◆ Deadlocking detection.
- ◆ Temporal dependencies.
- ◆ Pipeline of computing jobs.
- ◆ Check for symbolic link loop.
- ◆ Evaluate formula in spreadsheet.
- ◆ Program Evaluation and Review Technique / Critical Path Method

Topological Sort Application (Weighted DAG)

□ Precedence scheduling

- ◆ Task v takes time[v] units of time.
 - ✓ Can work on jobs in parallel.
- ◆ Precedence constraints:
 - ✓ must finish task v before beginning task w .
- ◆ Goal: finish each task as soon as possible

□ Example:

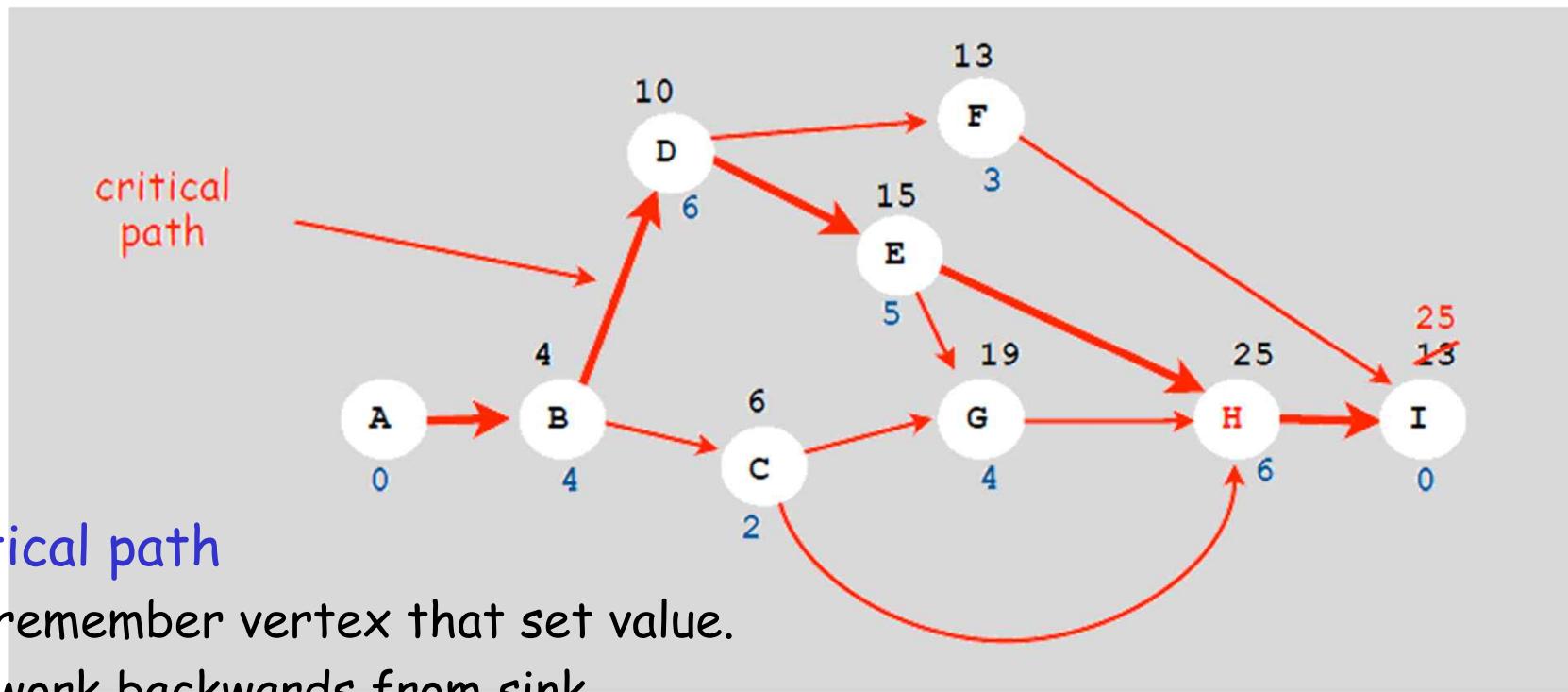


index	task	time	prereq
A	begin	0	-
B	framing	4	A
C	roofing	2	B
D	siding	6	B
E	windows	5	D
F	plumbing	3	D
G	electricity	4	C, E
H	paint	6	C, E
I	finish	0	F, H

Program Evaluation and Review Technique / Critical Path Method

□ PERT/CPM algorithm.

- ◆ compute topological order of vertices.
- ◆ initialize $\text{fin}[v] = 0$ for all vertices v .
- ◆ consider vertices v in topologically sorted order.
for each edge $v \rightarrow w$, set $\text{fin}[w] = \max(\text{fin}[w], \text{fin}[v] + \text{time}[w])$



□ Critical path

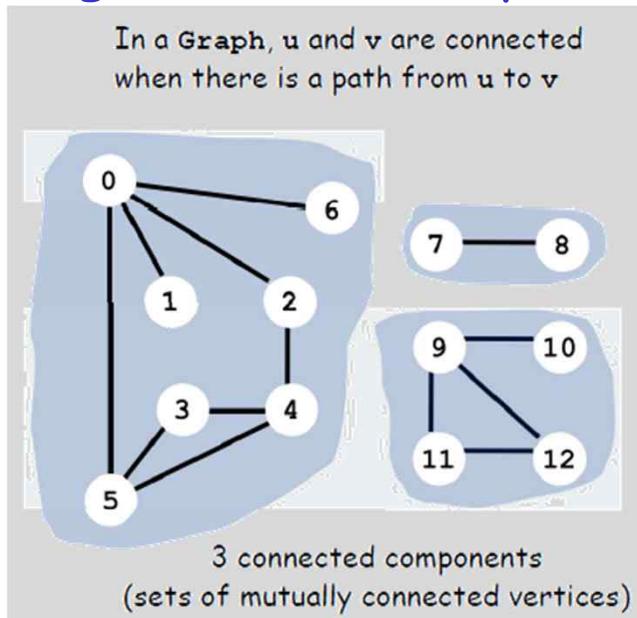
- ◆ remember vertex that set value.
- ◆ work backwards from sink

11. Directed Graphs

- digraph search
- transitive closure
- topological sort
- strong components

Strong Connectivity in Digraphs

- Analog to connectivity in undirected graphs

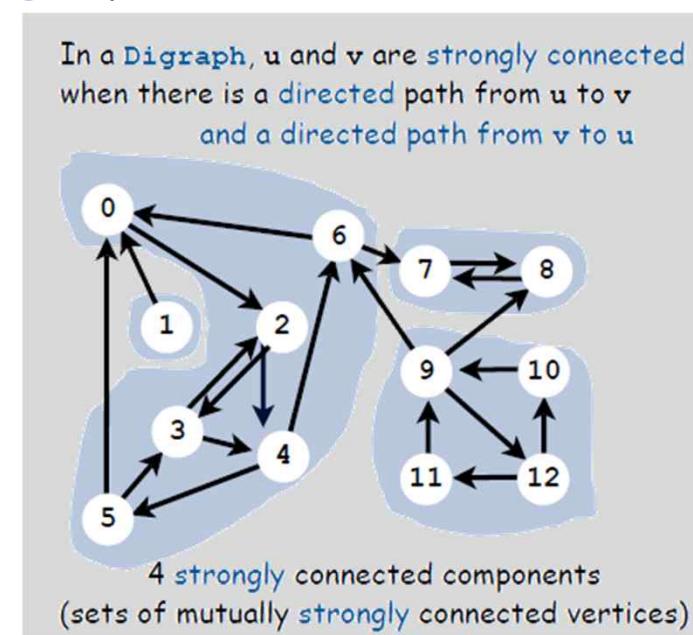


Connectivity table (easy to compute with DFS)

	0	1	2	3	4	5	6	7	8	9	10	11	12
cc	0	0	0	0	0	0	0	1	1	2	2	2	2

```
public int connected(int v, int w)
{ return cc[v] == cc[w]; }
```

constant-time client connectivity query



Strong connectivity table (how to compute?)

	0	1	2	3	4	5	6	7	8	9	10	11	12
sc	2	1	2	2	2	2	2	3	3	0	0	0	0

```
public int connected(int v, int w)
{ return cc[v] == cc[w]; }
```

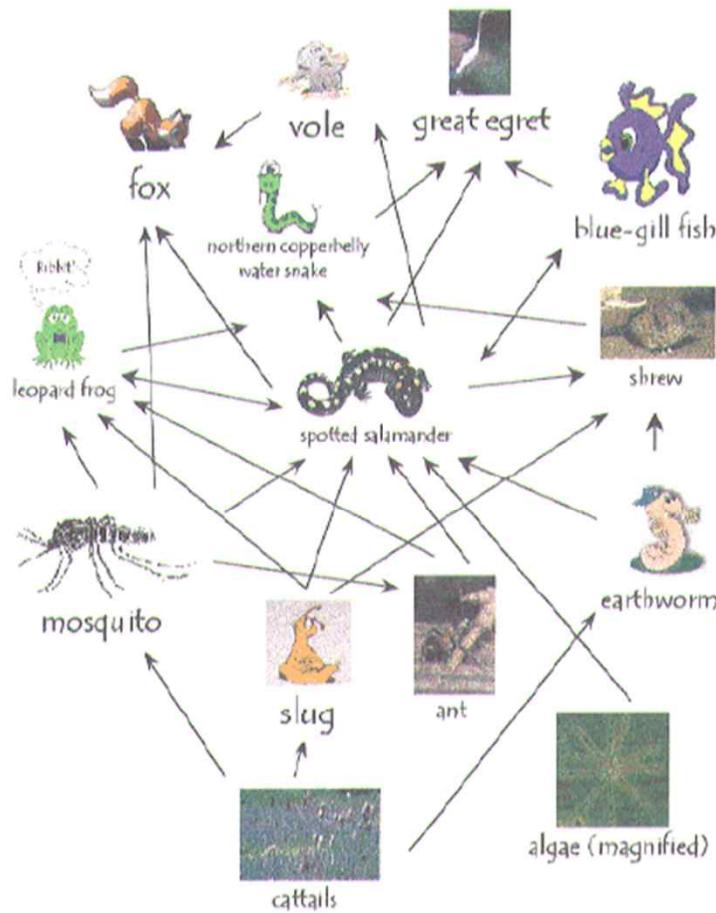
constant-time client strong connectivity query

Digraph-Processing Challenge

- Problem 4: Is there a directed cycle containing s and t ?
 - Equivalent: Are there directed paths from s to t and from t to s ?
 - Equivalent: Are s and t strongly connected?
-
- Goals: linear $(V + E)$ preprocessing time (like for undirected graphs)
constant query time

Typical Strong Components Applications

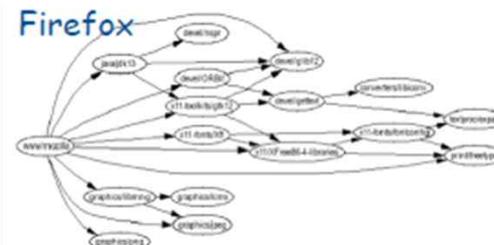
Ecological food web



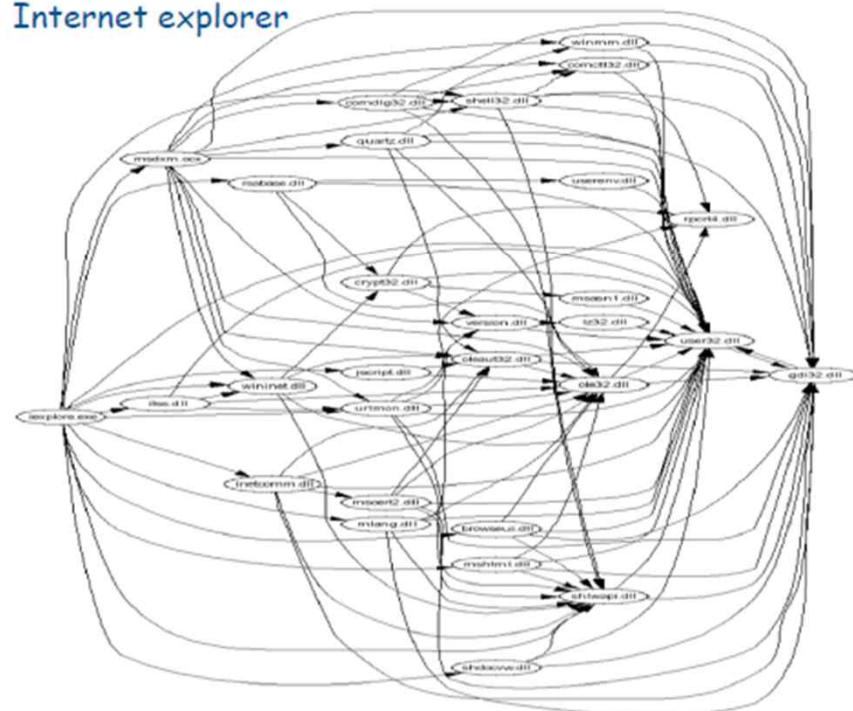
Strong component: subset with common energy flow

- source in kernel DAG: needs outside energy?
 - sink in kernel DAG: heading for growth?

Software module dependency digraphs



Internet explorer



Strong component: subset of mutually interacting modules

- approach 1: package strong components together
 - approach 2: use to improve design!

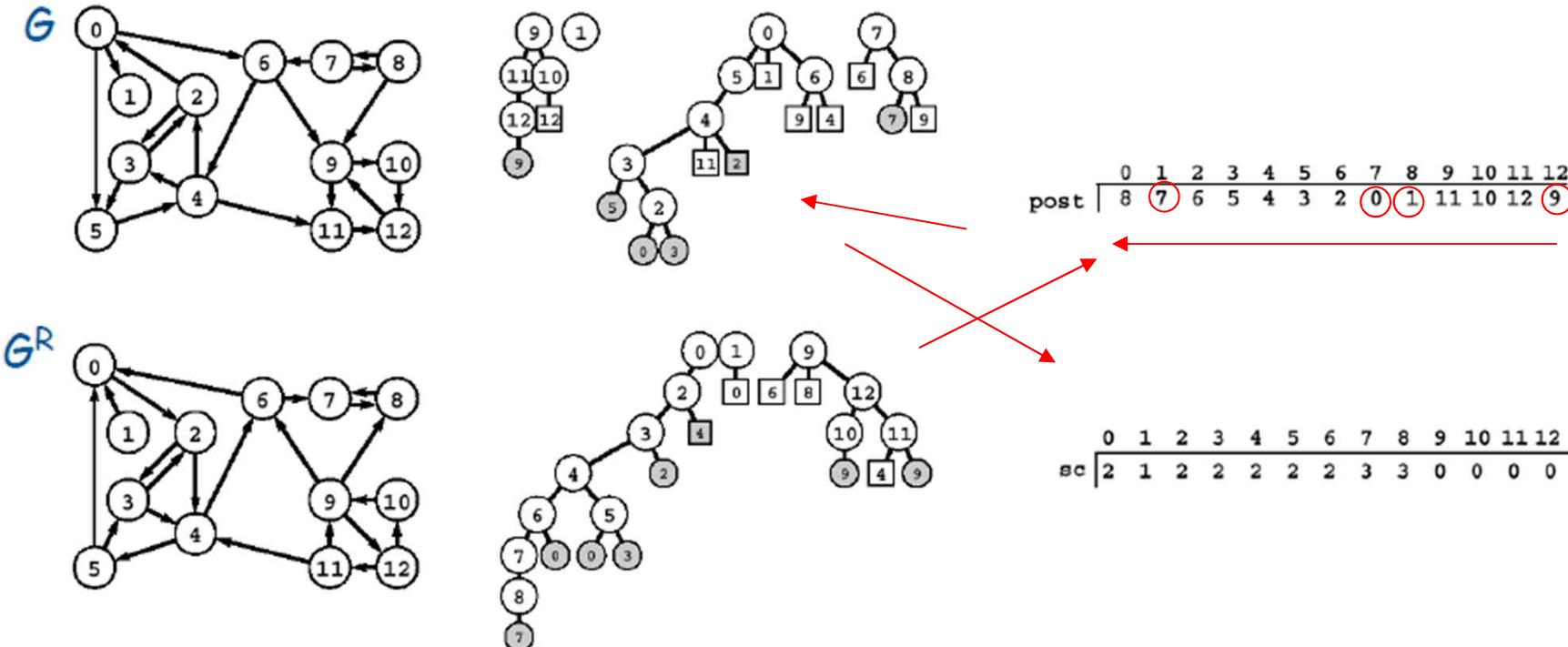
Strong Components Algorithms: Brief History

- 1960s: Core OR (Open Research) problem
 - ◆ widely studied
 - ◆ some practical algorithms
 - ◆ complexity not understood
- 1972: Linear-time DFS algorithm (Tarjan)
 - ◆ classic algorithm
 - ◆ demonstrated broad applicability and importance of DFS
- 1980s: Easy two-pass linear-time algorithm (Kosaraju)
 - ◆ forgot notes for teaching algorithms class
 - ◆ developed algorithm in order to teach it!
 - ◆ later found in Russian scientific literature (1972)
- 1990s: More easy linear-time algorithms (Gabow, Mehlhorn)
 - ◆ Gabow: fixed old OR algorithm
 - ◆ Mehlhorn: needed one-pass algorithm for LEDA (Library of Efficient Data types and Algorithms) 45



Kosaraju's Algorithm

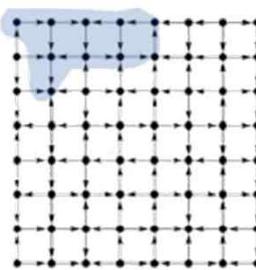
- Simple (but mysterious) algorithm for computing strong components
 - ◆ Run DFS on G^R and compute postorder.
 - ◆ Run DFS on G , considering vertices in reverse postorder
 - ◆ [has to be seen to be believed: follow the example below]



- Theorem. Trees in second DFS are strong components. (!)
- Proof. [stay tuned]

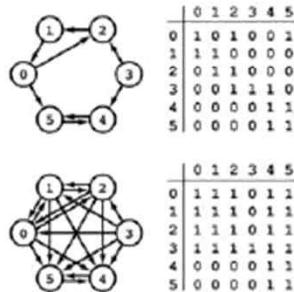
Digraph-Processing Summary: Algorithms of the Day

Single-source
reachability



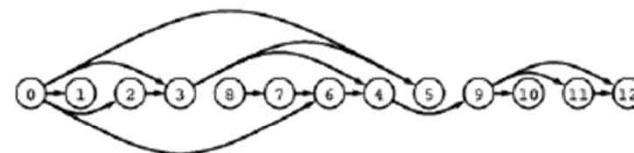
DFS

transitive closure



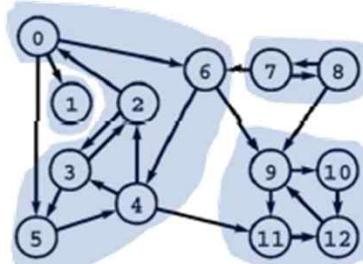
DFS from each vertex

topological sort
(DAG)



DFS

strong components



Kosaraju
DFS (twice)

Algorithms

Data Structures & Algorithms

12. Minimum Spanning Trees

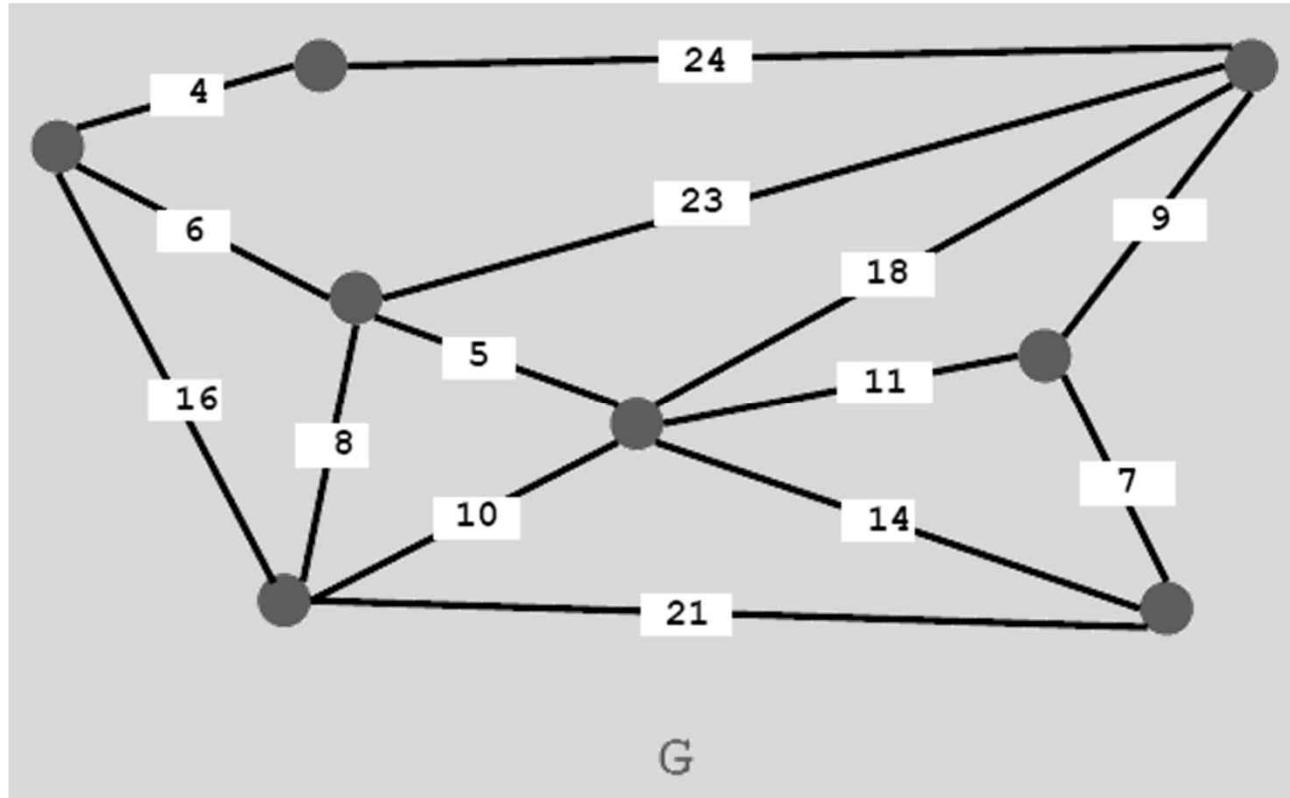
- weighted graph API
- cycles and cuts
- Kruskal's algorithm
- Prim's algorithm
- advanced topics



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

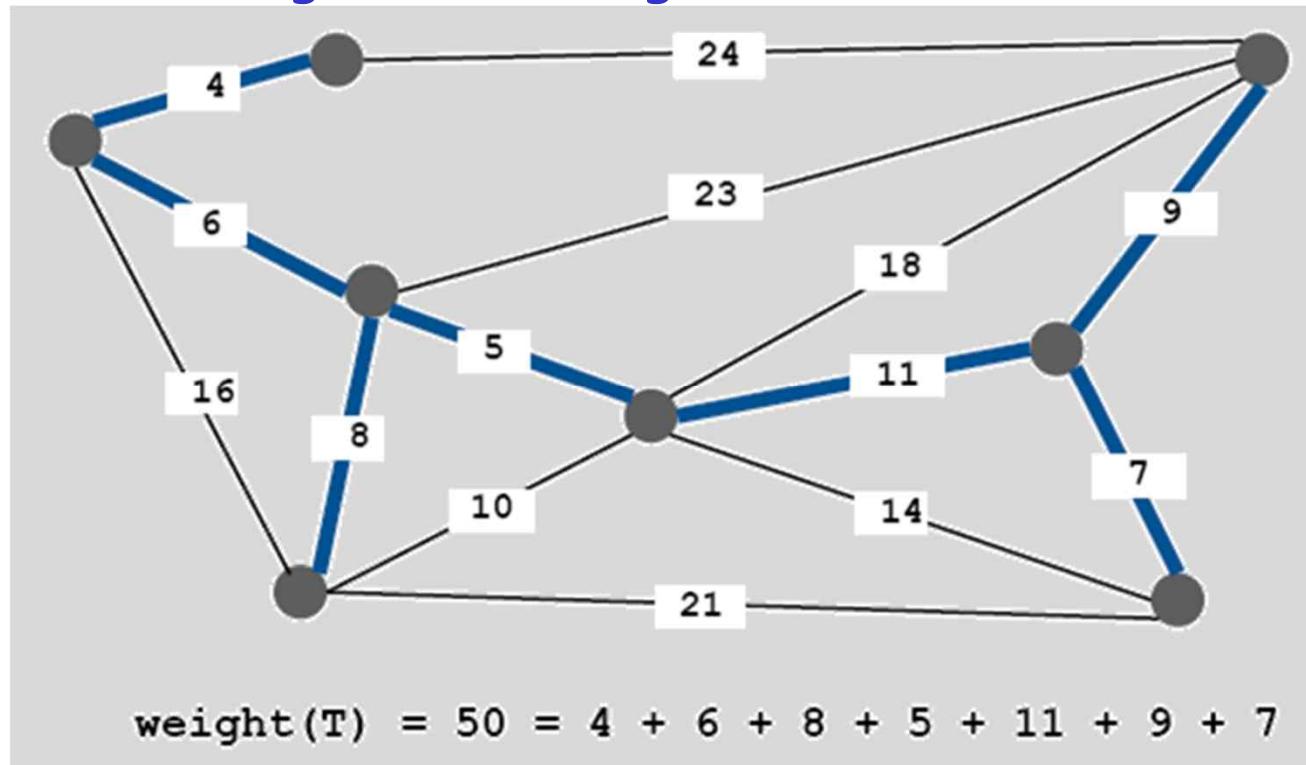
Minimum Spanning Tree

- Given. Undirected graph G with positive edge weights (connected).
- Goal. Find a min weight set of edges that connects all of the vertices.



Minimum Spanning Tree

- Given. Undirected graph G with positive edge weights (connected).
- Goal. Find a min weight set of edges that connects all of the vertices.



- Brute force: Try all possible spanning trees

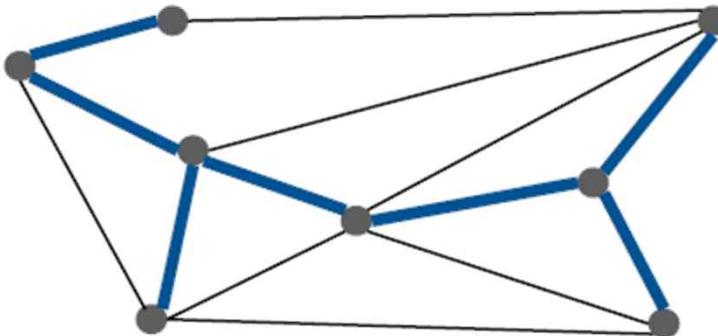
- problem 1: not so easy to implement
- problem 2: far too many of them

Ex: [Cayley, 1889]: V^{V-2} spanning trees
on the complete graph on V vertices.

MST Origin

□ Otakar Boruvka (1926).

- ◆ Electrical Power Company of Western Moravia in Brno (Czech).
- ◆ Most economical construction of electrical power network.
- ◆ Concrete engineering problem is now a cornerstone
- ◆ Problem-solving model in combinatorial optimization.



Otakar Boruvka

□ Brute force: Try all possible spanning trees

- ◆ problem 1: not so easy to implement
- ◆ problem 2: far too many of them



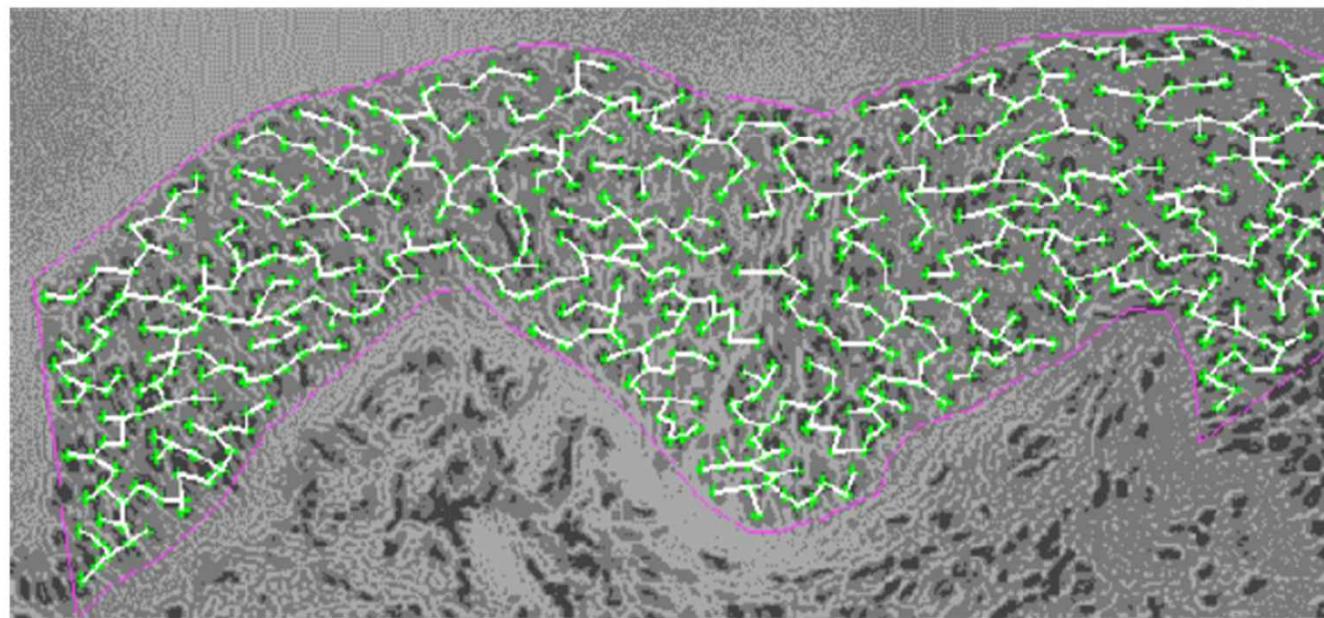
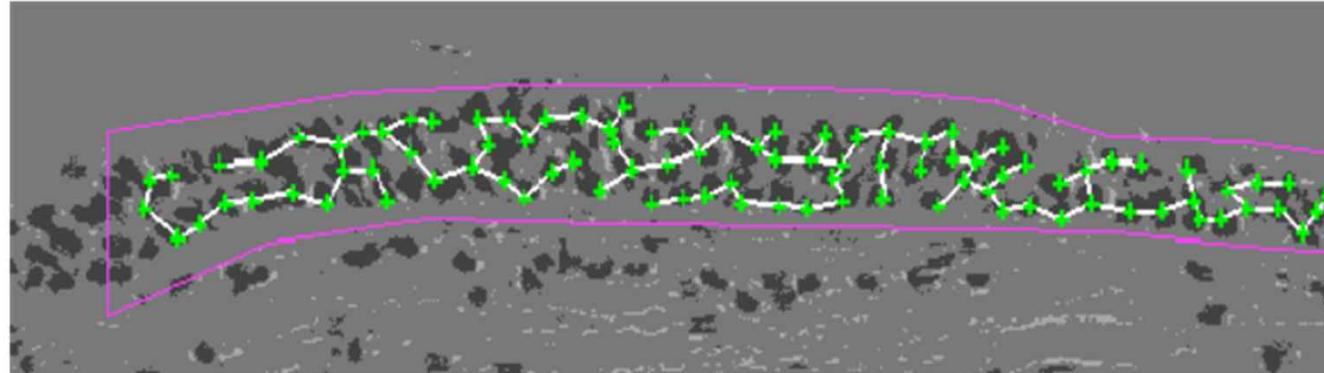
Ex: [Cayley, 1889]: V^{V-2} spanning trees on the complete graph on V vertices.

Applications

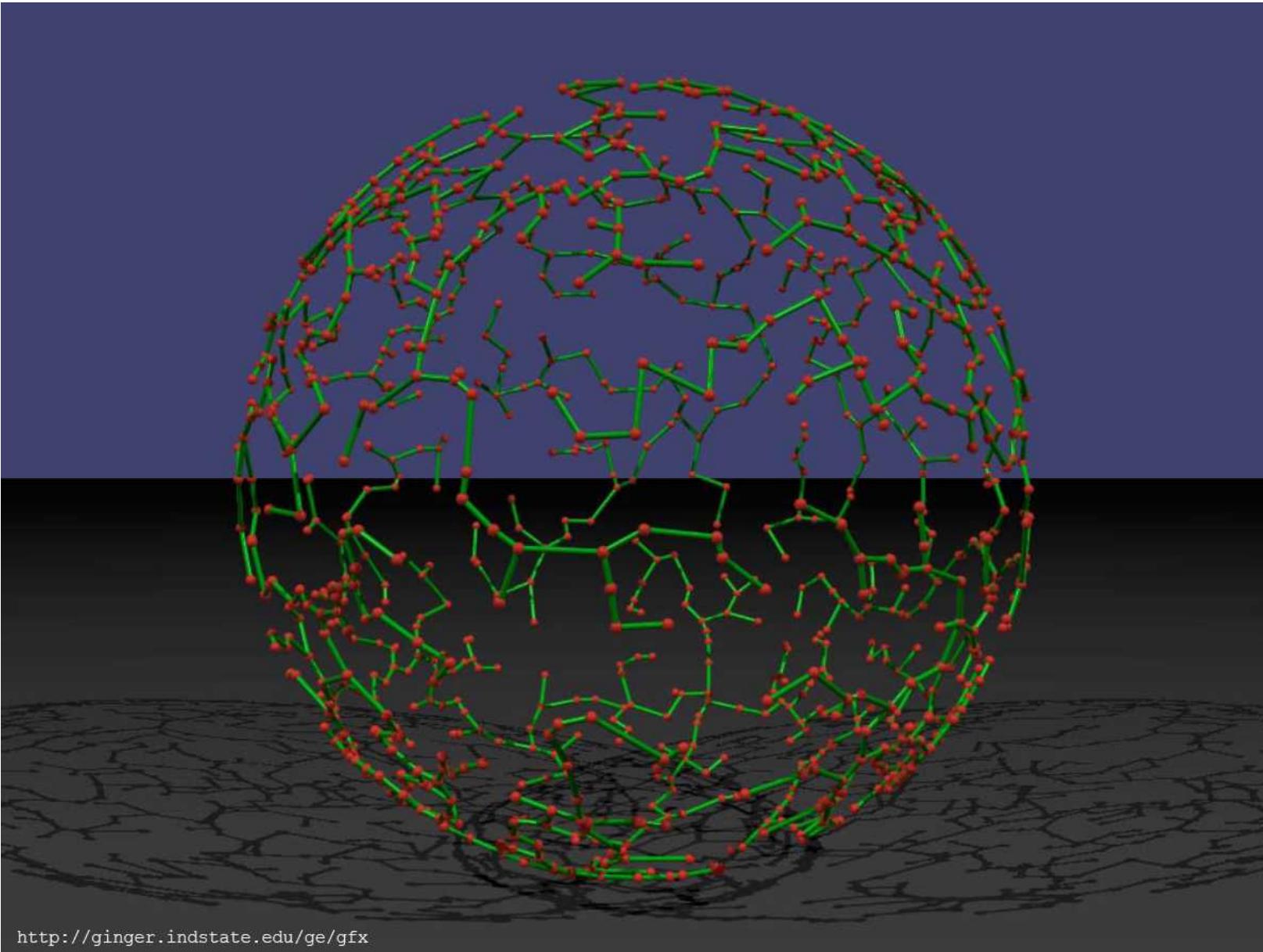
- MST is fundamental problem with diverse applications.
 - ◆ Network design.
 - ✓ telephone, electrical, hydraulic, TV cable, computer, road
 - ◆ Approximation algorithms for NP-hard problems.
 - ✓ traveling salesperson problem, Steiner tree
 - ◆ Indirect applications.
 - ✓ max bottleneck paths
 - ✓ LDPC (Low Density Parity Check) **codes** for error correction
 - ✓ **image** registration with Renyi entropy
 - ✓ learning salient features for real-time **face** verification
 - ✓ reducing data storage in sequencing **amino acids** in a protein
 - ✓ model locality of particle interactions in turbulent **fluid flows**
 - ✓ autoconfig protocol for **Ethernet** bridging to avoid cycles in a network
 - ◆ Cluster analysis.

Medical Image Processing

- ❑ MST describes arrangement of nuclei in the epithelium for cancer research



Medical Image Processing

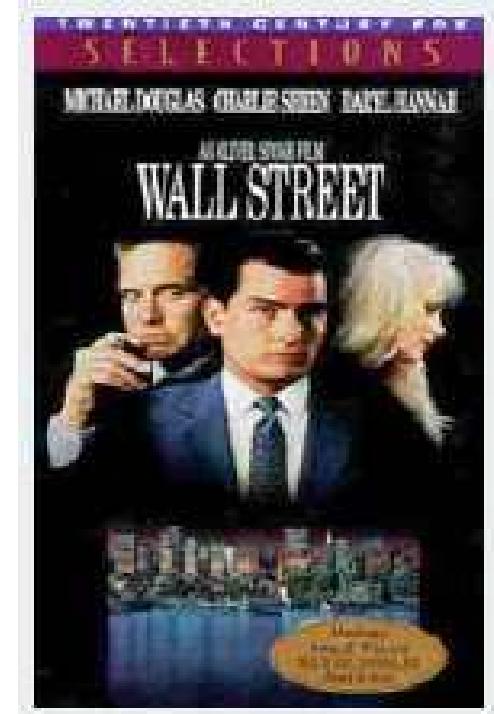


<http://ginger.indstate.edu/ge/gfx>

Two Greedy Algorithms

- **Kruskal's algorithm.** Consider edges in ascending order of cost. Add the next edge to T unless doing so would create a cycle.
- **Prim's algorithm.** Start with any vertex s and greedily grow a tree T from s . At each step, add the cheapest edge to T that has exactly one endpoint in T .
- **Proposition.** Both greedy algorithms compute an MST.

Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit." - Gordon Gecko



12. Minimum Spanning Trees

- weighted graph API
- cycles and cuts
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

Weighted Graph API

```
public class WeightedGraph  
{  
    WeightedGraph(int V)           create an empty graph with V vertices  
    void insert(Edge e)            insert edge e  
    Iterable<Edge> adj(int v)     return an iterator over edges incident to v  
    int V()                      return the number of vertices  
    String toString()             return a string representation
```

iterate through all edges (once in each direction)

Weighted graph data type

- Identical to *Graph.java* but use *Edge* adjacency sets instead of *int*.

```
public class WeightedGraph
{
    private int V;
    private SET<Edge>[] adj;

    public Graph(int V)
    {
        this.V = V;
        adj = (SET<Edge>[]) new SET[V];
        for (int v = 0; v < V; v++)
            adj[v] = new SET<Edge>();
    }

    public void addEdge(Edge e)
    {
        int v = e.v, w = e.w;
        adj[v].add(e);
        adj[w].add(e);
    }

    public Iterable<Edge> adj(int v)
    { return adj[v]; }
}
```

Weighted edge data type

□ Edge abstraction needed for weights

```
public class Edge implements Comparable<Edge>
{
    private final int v, int w;
    private final double weight;

    public Edge(int v, int w, double weight)
    {
        this.v = v;
        this.w = w;
        this.weight = weight;
    }

    public int either()
    {   return v;   }

    public int other(int vertex)
    {
        if (vertex == v) return w;
        else return v;
    }

    public int weight()
    {   return weight;   }

    // See next slide for edge compare methods.
}
```

slightly tricky accessor methods
(enables client code like this)

```
for (int v = 0; v < G.V(); v++)
{
    for (Edge e : G.adj(v))
    {
        int w = e.other(v);
        // edge v-w
    }
}
```



Weighted edge data type: compare methods

- Two different compare methods for edges
 - ◆ `compareTo()` so that edges are Comparable (for use in SET)
 - ◆ `compare()` so that clients can compare edges by weight.

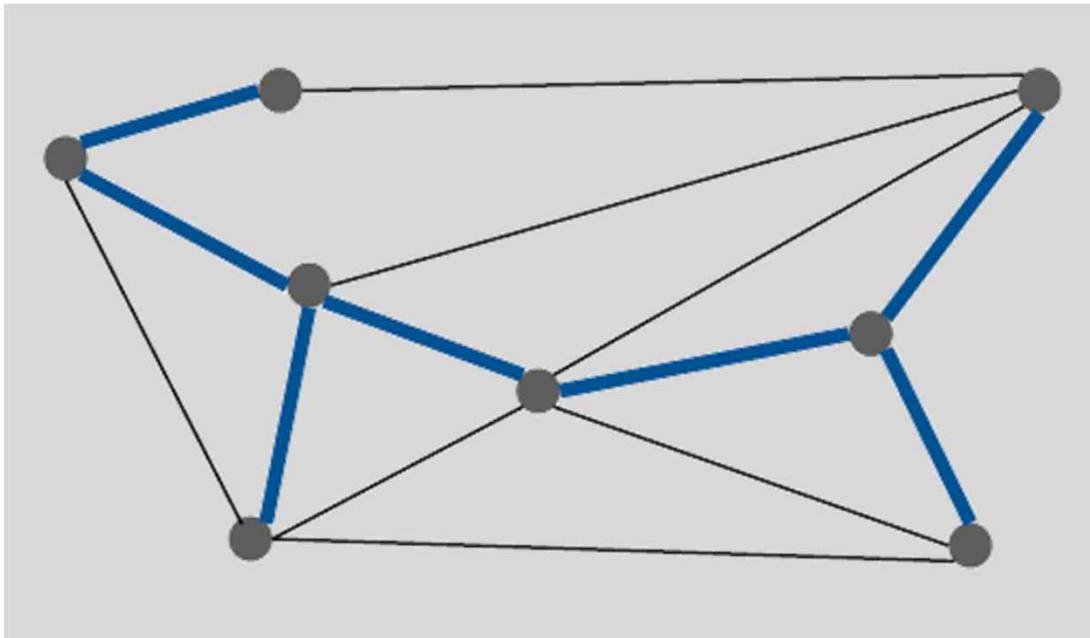
```
public final static Comparator<Edge> BY_WEIGHT = new ByWeightComparator();  
  
private static class ByWeightComparator implements Comparator<Edge>  
{  
    public int compare(Edge e, Edge f)  
    {  
        if (e.weight < f.weight) return -1;  
        if (e.weight > f.weight) return +1;  
        return 0;  
    }  
}  
  
public int compareTo(Edge that)  
{  
    if (this.weight < that.weight) return -1;  
    else if (this.weight > that.weight) return +1;  
    else  
        return 0;  
}
```

12. Minimum Spanning Trees

- weighted graph API
- *cycles and cuts*
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

Spanning Tree

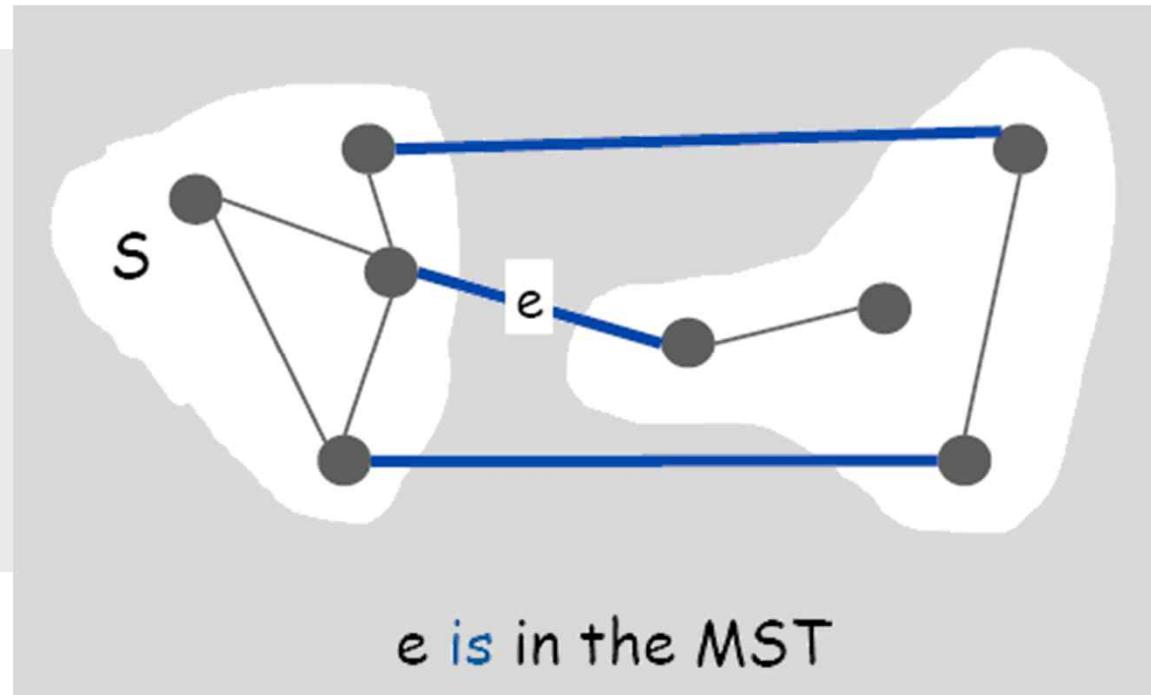
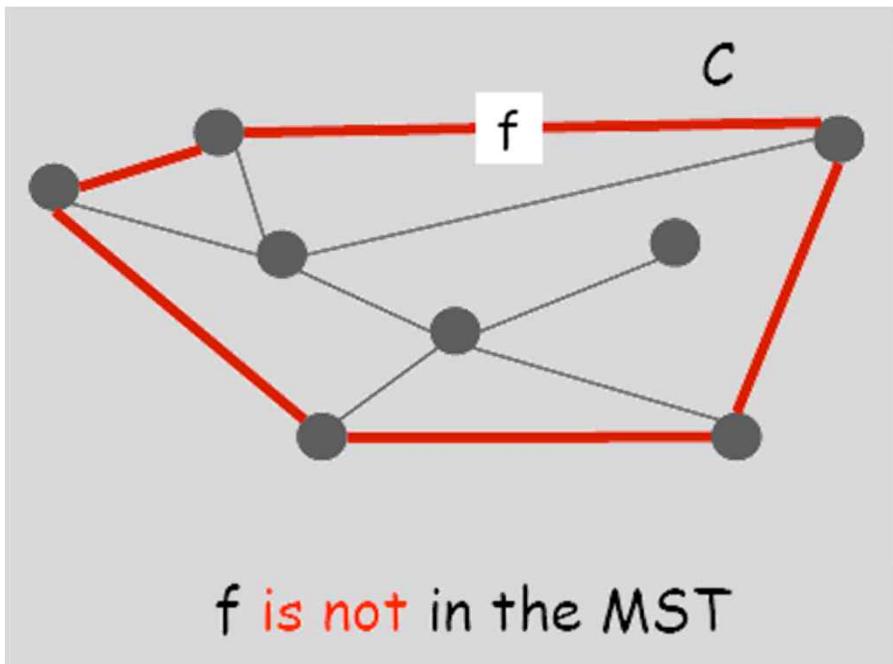
- **MST.** Given connected graph G with positive edge weights, find a min weight set of edges that connects all of the vertices.
- **Def.** A **spanning tree** of a graph G is a subgraph T that is connected and acyclic.



- **Property.** MST of G is always a spanning tree.

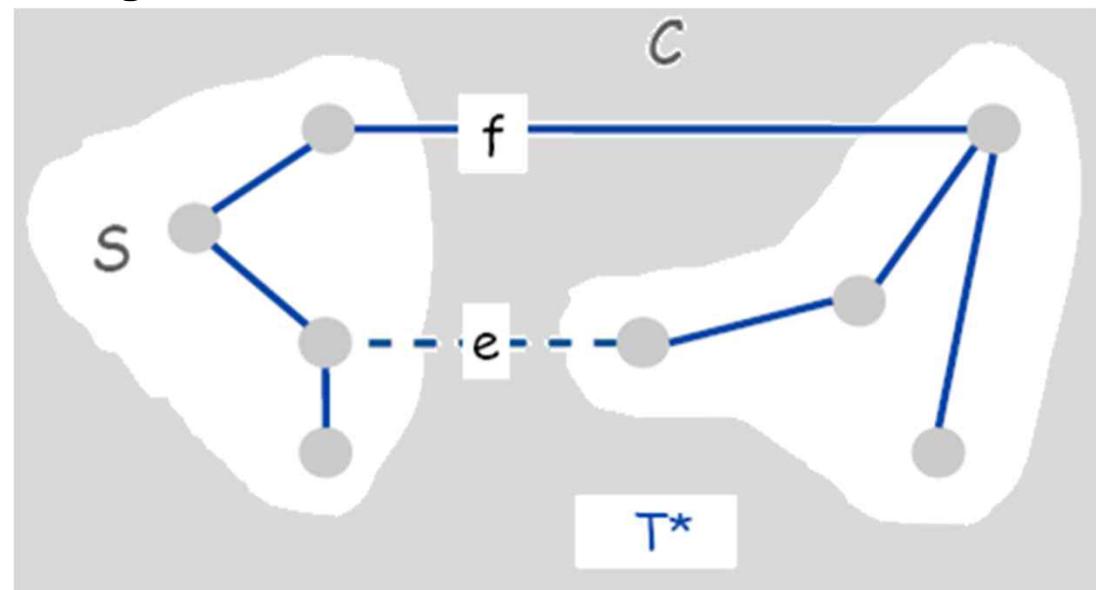
Greedy Algorithms

- Simplifying assumption. All edge weights w_e are distinct.
- Cycle property. Let C be any cycle, and let f be the **max cost** edge belonging to C . Then the MST does not contain f .
- Cut property. Let S be any subset of vertices, and let e be the **min cost** edge with exactly one endpoint in S . Then the MST contains e .



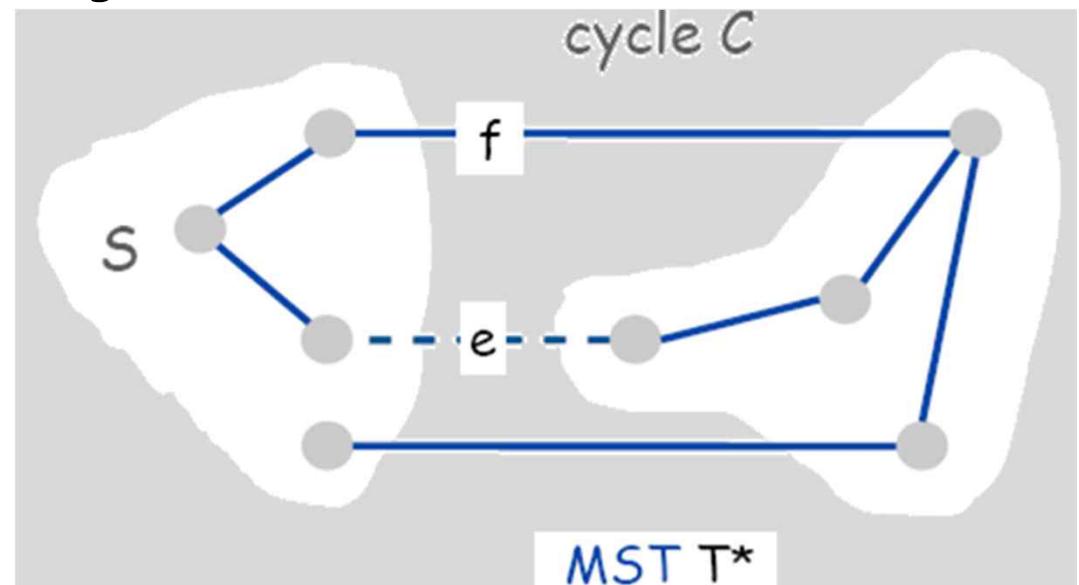
Cycle Property

- Simplifying assumption. All edge weights w_e are distinct.
- Cycle property. Let C be any cycle, and let f be the **max cost** edge belonging to C . Then the MST T^* does not contain f .
- Pf. [by contradiction]
 - ◆ Suppose f belongs to T^* . Let's see what happens.
 - ◆ Deleting f from T^* disconnects T^* . Let S be one side of the cut.
 - ◆ Some other edge in C , say e , has exactly one endpoint in S .
 - ◆ $T = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
 - ◆ Since $c_e < c_f$, $\text{cost}(T) < \text{cost}(T^*)$.
 - ◆ Contradicts minimality of T^* .



Cut Property

- Simplifying assumption. All edge costs c_e are distinct.
- Cut property. Let S be any subset of vertices, and let e be the min cost edge with exactly one endpoint in S . Then the MST T^* contains e .
- Pf. [by contradiction]
 - ◆ Suppose e does not belong to T^* . Let's see what happens.
 - ◆ Adding e to T^* creates a (unique) cycle C in T^* .
 - ◆ Some other edge in C , say f , has exactly one endpoint in S .
 - ◆ $T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
 - ◆ Since $c_e < c_f$, $\text{cost}(T) < \text{cost}(T^*)$.
 - ◆ Contradicts minimality of T^* .

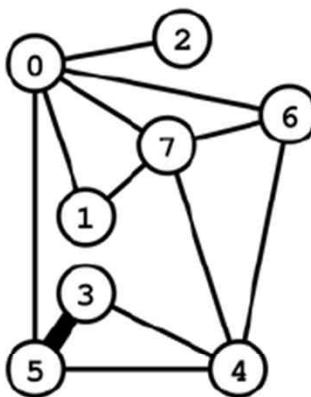
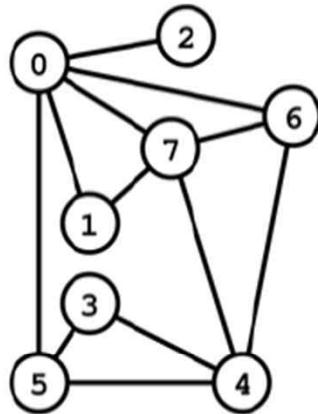


12. Minimum Spanning Trees

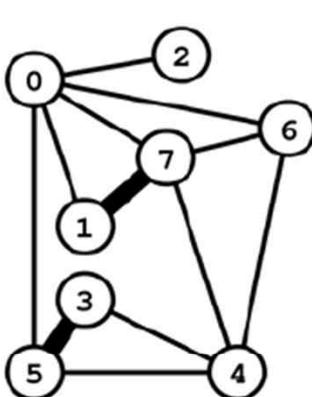
- weighted graph API
- cycles and cuts
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

Kruskal's Algorithm: Example

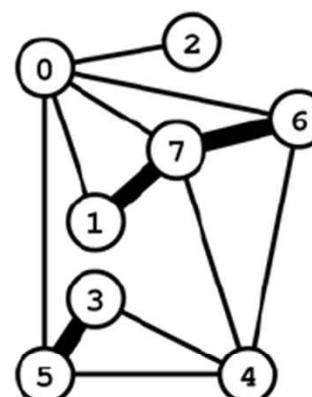
- Kruskal's algorithm. [Kruskal, 1956] Consider edges in ascending order of cost. Add the next edge to T unless doing so would create a cycle.



3-5

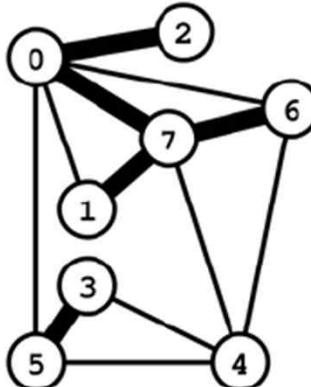
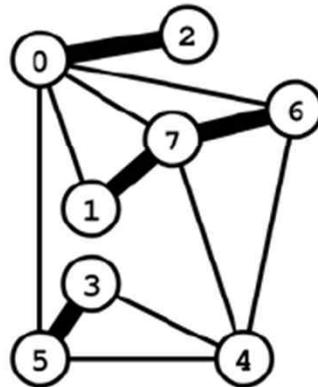


1-7

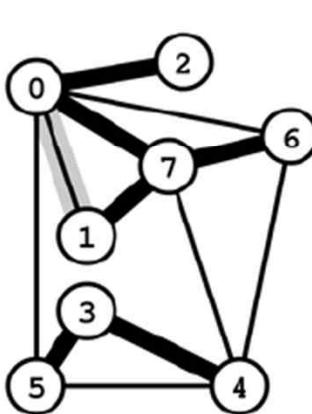


6-7

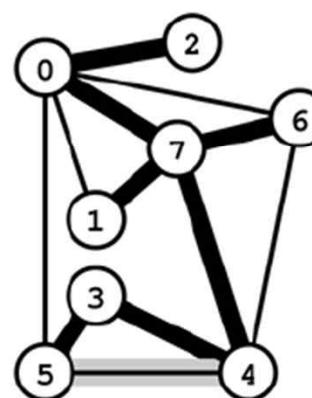
3-5	0.18
1-7	0.21
6-7	0.25
0-2	0.29
0-7	0.31
0-1	0.32
3-4	0.34
4-5	0.40
4-7	0.46
0-6	0.51
4-6	0.51
0-5	0.60



0-7

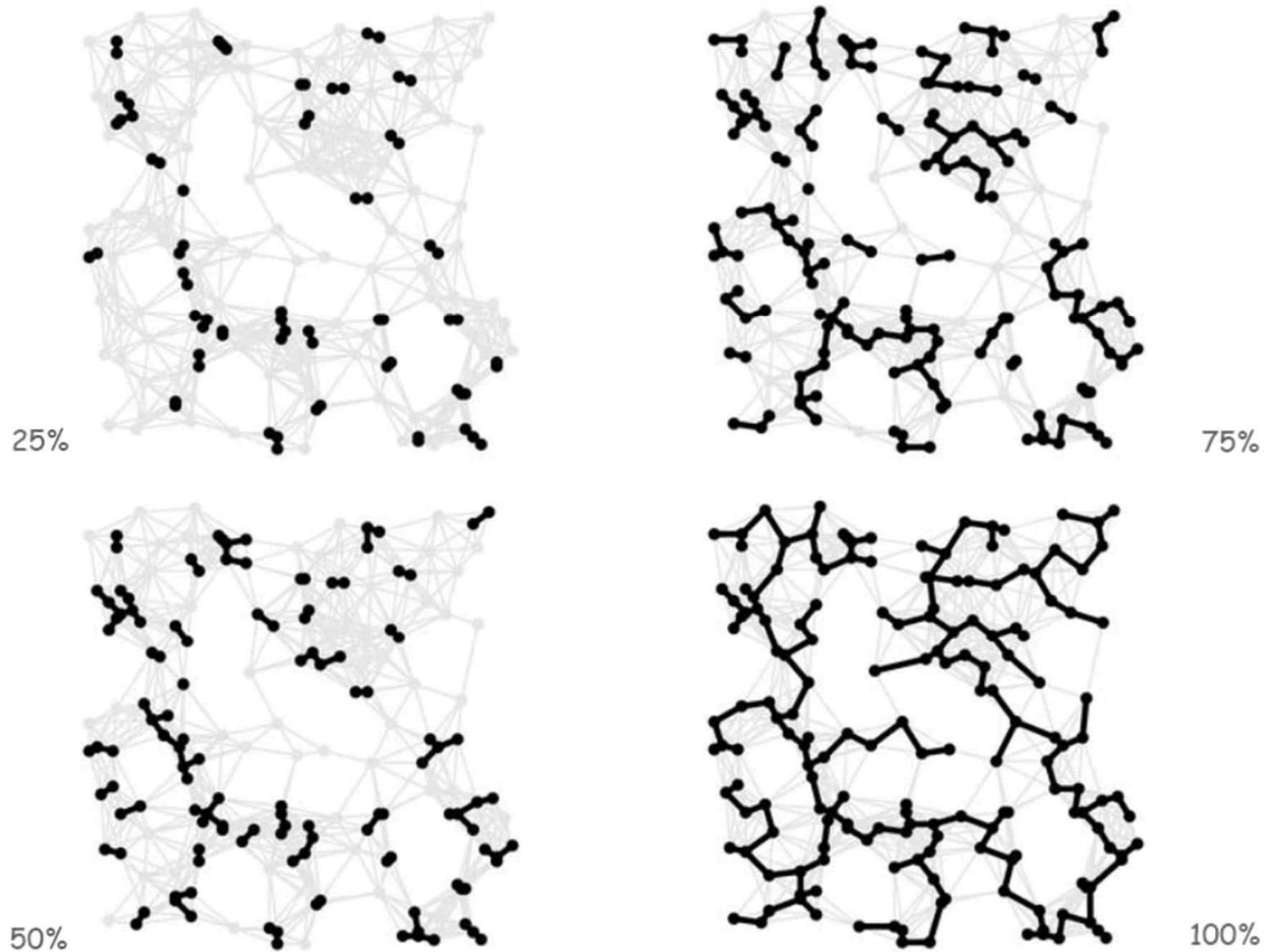


0-1 3-4



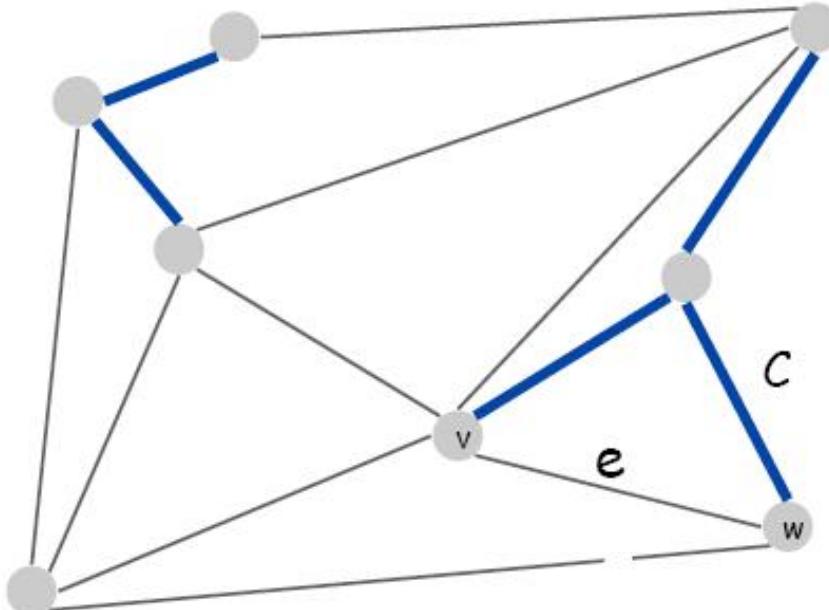
4-5 4-7

Kruskal's algorithm example



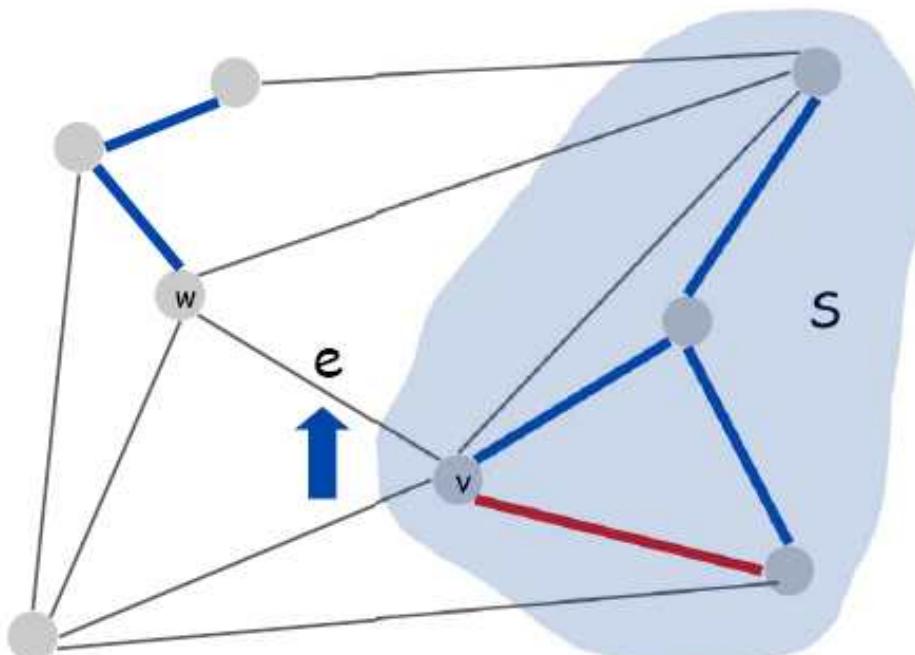
Kruskal's algorithm correctness proof

- Proposition. Kruskal's algorithm computes the MST.
- Pf. [case 1] Suppose that adding e to T creates a cycle C
 - ◆ e is the max weight edge in C (weights come in increasing order)
 - ◆ e is not in the MST (cycle property)



Kruskal's algorithm correctness proof

- Proposition. Kruskal's algorithm computes the MST.
- Pf. [case 2] Suppose that adding $e = (v, w)$ to T does not create a cycle
 - ◆ let S be the vertices in v 's connected component
 - ◆ w is not in S
 - ◆ e is the min weight edge with exactly one endpoint in S
 - ◆ e is in the MST (cut property)



Kruskal's algorithm implementation

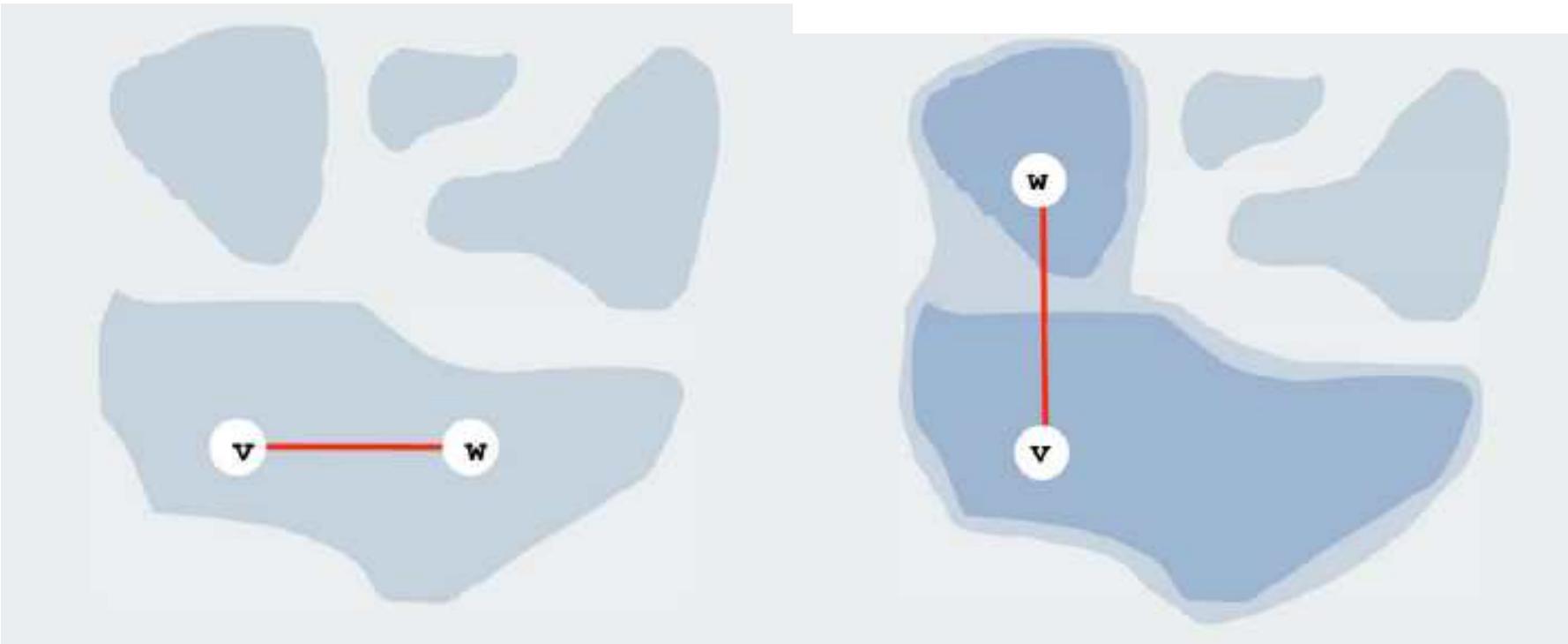
Q. How to check if adding an edge to T would create a cycle?

A1. Naive solution: use DFS.

- ◆ $O(V)$ time per cycle check.
- ◆ $O(E V)$ time overall.

Kruskal's algorithm implementation

- Q. How to check if adding an edge to T would create a cycle?
- A2. Use the union-find data structure from lecture 1 (!).
 - ◆ Maintain a set for each connected component.
 - ◆ If v and w are in same component, then adding $v-w$ creates a cycle.
 - ◆ To add $v-w$ to T , merge sets containing v and w .



Case 1: adding $v-w$ creates a cycle

Case 2: add $v-w$ to T and merge sets

Kruskal's algorithm: Java implementation

```
public class Kruskal
{
    private SET<Edge> mst = new SET<Edge>();

    public Kruskal(WeightedGraph G)
    {
        Edge[] edges = G.edges();
        Arrays.sort(edges, Edge.BY_WEIGHT); ← sort edges by weight

        UnionFind uf = new UnionFind(G.V());
        for (Edge e: edges)
            if (!uf.find(e.either(), e.other()))
            {
                uf.unite(e.either(), e.other());
                mst.add(edge); ← greedily add edges to MST
            }
    }

    public Iterable<Edge> mst() ← return to client iterable sequence of edges
    {   return mst;   }
}
```

- Easy speedup: Stop as soon as there are $V-1$ edges in MST.

Kruskal's algorithm running time

- Kruskal running time: Dominated by the cost of the sort.

Operation	Frequency	Time per op
sort	1	$E \log E$
union	V	$\log^* V$ †
find	E	$\log^* V$ †

† amortized bound using weighted quick union with path compression

recall: $\log^* V \leq 5$ in this universe

- Remark 1.

- ◆ If edges are already sorted, time is proportional to $E \log^* V$

- Remark 2.

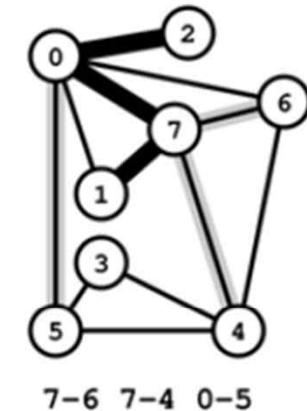
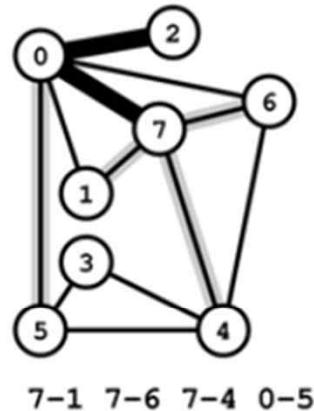
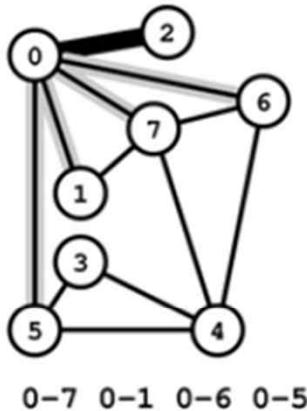
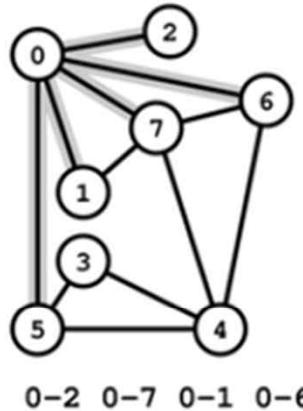
- ◆ Linear in practice with PQ or quicksort partitioning (see book: don't need full sort)

12. Minimum Spanning Trees

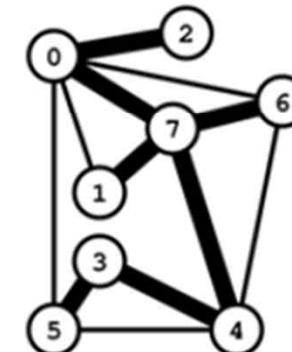
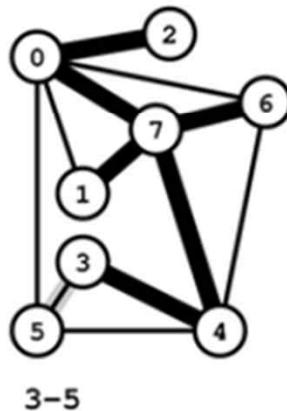
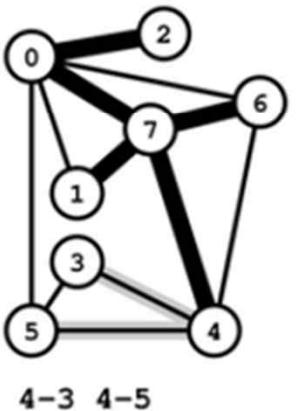
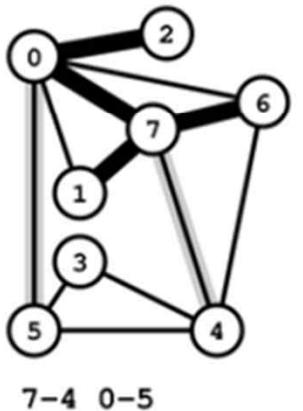
- weighted graph API
- cycles and cuts
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

Prim's algorithm example

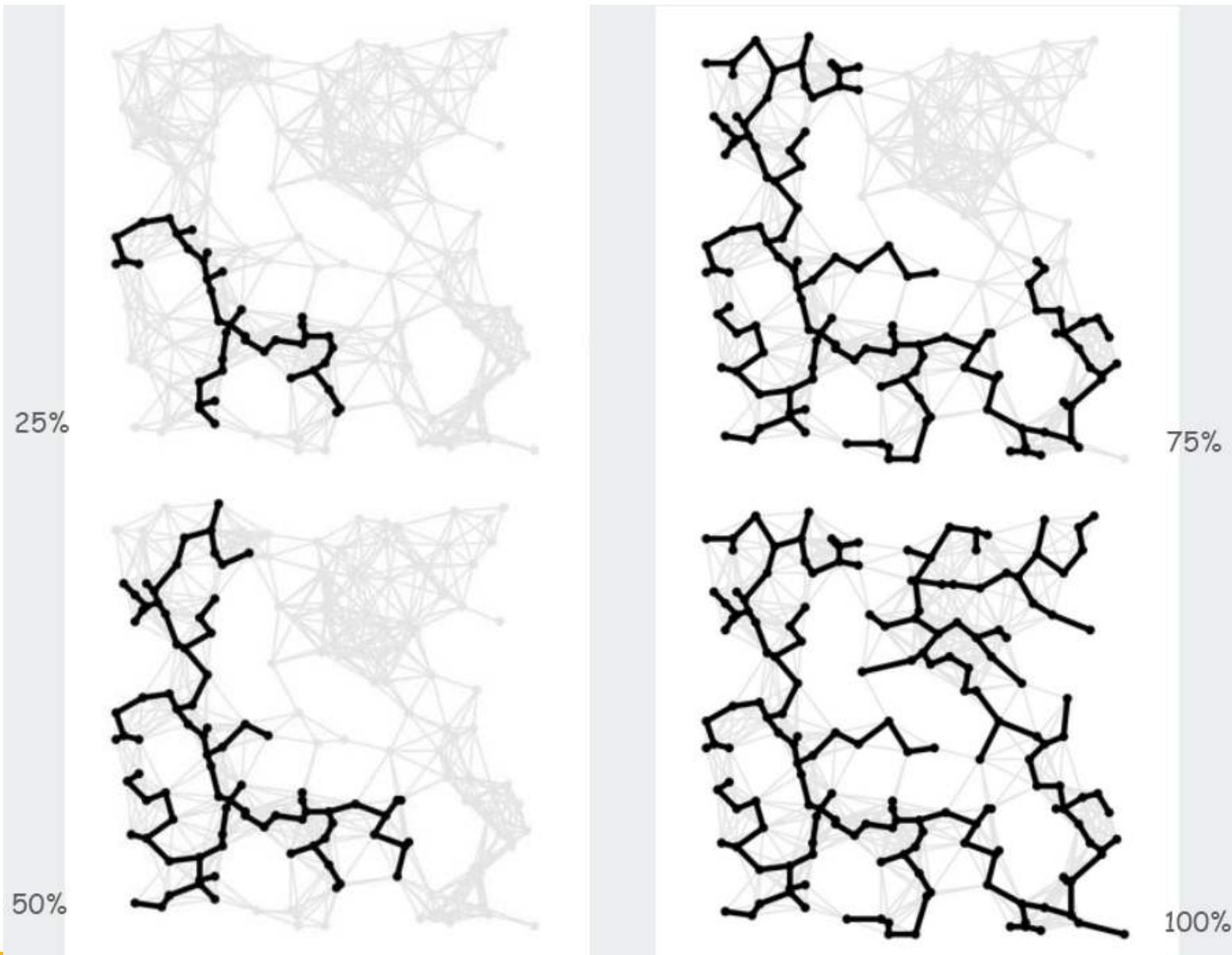
- Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959] Start with vertex 0 and greedily grow tree T. At each step, add cheapest edge that has exactly one endpoint in T.



0-1	0.32
0-2	0.29
0-5	0.60
0-6	0.51
0-7	0.31
1-7	0.21
3-4	0.34
3-5	0.18
4-5	0.40
4-6	0.51
4-7	0.46
6-7	0.25

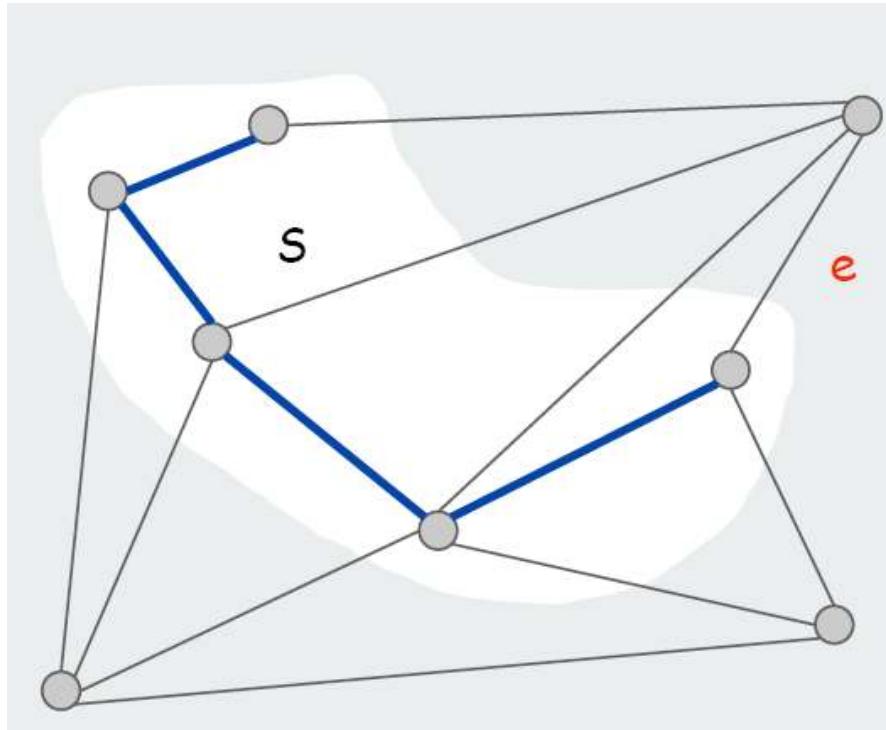


Prim's Algorithm example



Prim's algorithm correctness proof

- Proposition. Prim's algorithm computes the MST.
- Pf.
 - ◆ Let S be the subset of vertices in current tree T .
 - ◆ Prim adds the cheapest edge e with exactly one endpoint in S .
 - ◆ e is in the MST (cut property)



Prim's algorithm implementation

Q. How to find cheapest edge with exactly one endpoint in S ?

A1. Brute force: try all edges.

- ◆ $O(E)$ time per spanning tree edge.
- ◆ $O(E V)$ time overall.

Prim's algorithm implementation

- Q. How to find cheapest edge with exactly one endpoint in S ?
- A2. Maintain a priority queue of vertices connected by an edge to S
 - ◆ Delete min to determine next vertex v to add to S .
 - ◆ Disregard v if already in S .
 - ◆ Add to PQ any vertex brought closer to S by v .
- Running time.
 - ◆ $\log V$ steps per edge (using a binary heap).
 - ◆ $E \log V$ steps overall.

Note: This is a **lazy** version of implementation in Algs in Java

- ◆ lazy: put all adjacent vertices (that are not already in MST) on PQ
- ◆ eager: first check whether vertex is already on PQ and decrease its key

Key-value priority queue

- Associate a value with each key in a priority queue.
- API:

public class MinPQplus<Key extends Comparable<Key>, Value>	
MinPQplus ()	create a key-value priority queue
void put(Key key, Value val)	put key-value pair into the priority queue
Value delMin()	return value paired with minimal key
Key min()	return minimal key

- Implementation:

- ◆ start with same code as standard heap-based priority queue
- ◆ use a parallel array vals[] (value associated with keys[i] is vals[i])
- ◆ modify exch() to maintain parallel arrays (do exch in vals[])
- ◆ modify delMin() to return Value
- ◆ add min() (just returns keys[1])



Lazy implementation of Prim's algorithm

```
public class LazyPrim
{
    Edge[] pred = new Edge[G.V()];
    public LazyPrim(WeightedGraph G)
    {
        boolean[] marked = new boolean[G.V()];
        double[] dist = new double[G.V()];
        MinPQplus<Double, Integer> pq;
        pq = new MinPQplus<Double, Integer>();
        for(i = 0; i < G.V(); i++) dist[i] = INFINITE;
        dist[s] = 0.0;
        pq.put(dist[s], s);
        while (!pq.isEmpty())
        {
            int v = pq.delMin();
            if (marked[v]) continue;
            marked(v) = true;
            for (Edge e : G.adj(v))
            {
                int w = e.other(v);
                if (!marked[w] && (dist[w] > e.weight()))
                {
                    dist[w] = e.weight(); pred[w] = e;
                    pq.insert(dist[w], w);
                }
            }
        }
    }
}
```

pred[v] is edge attaching v to MST

marks vertices in MST

distance to MST

key-value PQ

get next vertex

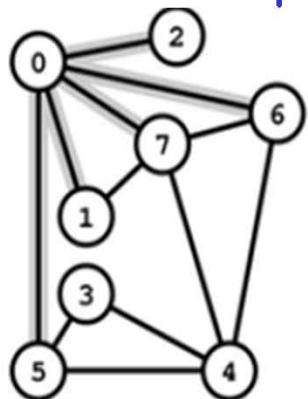
ignore if already in MST

add to PQ any vertices brought closer to S by v

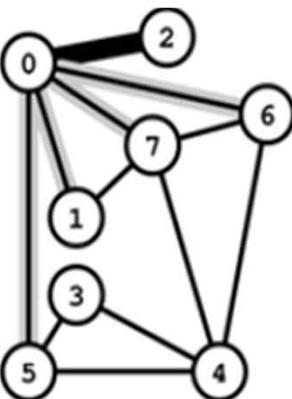
Lazy: heavier edges already put in PQ are not removed.

Prim's algorithm (lazy) example

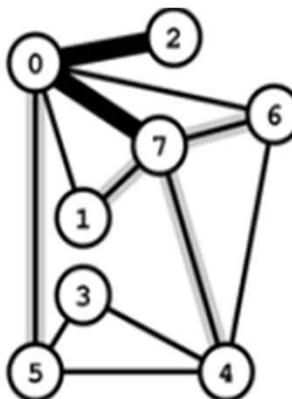
- Priority queue **key** is distance (edge weight); **value** is vertex
- Lazy version leaves obsolete entries in the PQ therefore may have multiple entries with same value



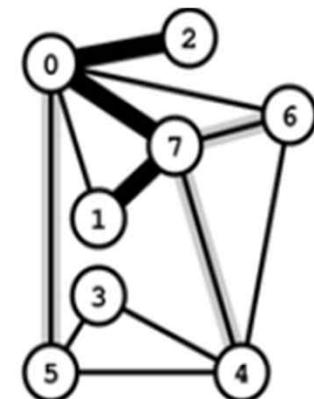
0-2 0-7 0-1 0-6 0-5



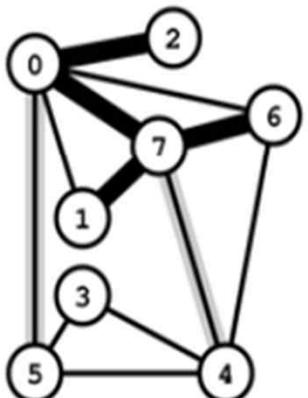
0-7 0-1 0-6 0-5



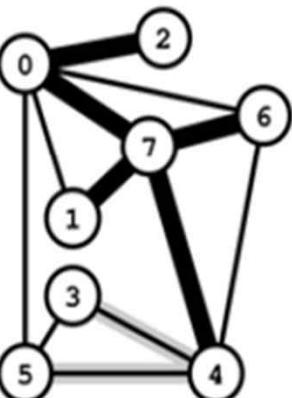
7-1 7-6 0-1 7-4 0-6 0-5 7-6 0-1 7-4 0-6 0-5



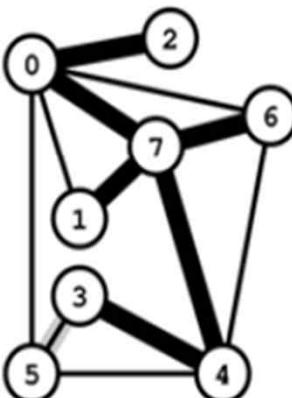
0-1	0.32
0-2	0.29
0-5	0.60
0-6	0.51
0-7	0.31
1-7	0.21
3-4	0.34
3-5	0.18
4-5	0.40
4-6	0.51
4-7	0.46
6-7	0.25



0-1 7-4 0-6 0-5



4-3 4-5 0-6 0-5



3-5 4-5 0-6 0-5



red: pq value (vertex)
blue: obsolete value

thms

Eager implementation of Prim's algorithm

- Use indexed priority queue that supports
 - ◆ contains: is there a key associated with value v in the priority queue?
 - ◆ decrease key: decrease the key associated with value v
- [more complicated data structure, see text]
- Putative “benefit”: reduces PQ size guarantee from E to V
 - ◆ not important for the huge sparse graphs found in practice
 - ◆ PQ size is far smaller in practice
 - ◆ widely used, but practical utility is debatable

Removing the distinct edge costs assumption

- Simplifying assumption. All edge weights w_e are distinct.
- Fact. Prim and Kruskal don't actually rely on the assumption (our proof of correctness does)
- Suffices to introduce tie-breaking rule for `compare()`.

- Approach 1:

```
public int compare(Edge e, Edge f)
{
    if (e.weight < f.weight) return -1;
    if (e.weight > f.weight) return +1;
    if (e.v < f.v) return -1;
    if (e.v > f.v) return +1;
    if (e.w < f.w) return -1;
    if (e.w > f.w) return +1;
    return 0;
}
```

- Approach 2: add tiny random perturbation.

12. Minimum Spanning Trees

- weighted graph API
- cycles and cuts
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

Advanced MST theorems

- does an algorithm with a linear-time guarantee exist?

Year	Worst Case	Discovered By
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E \alpha(V) \log \alpha(V)$	Chazelle
2000	$E \alpha(V)$	Chazelle
2002	optimal	Pettie-Ramachandran
20xx	E	???

deterministic comparison based MST algorithms

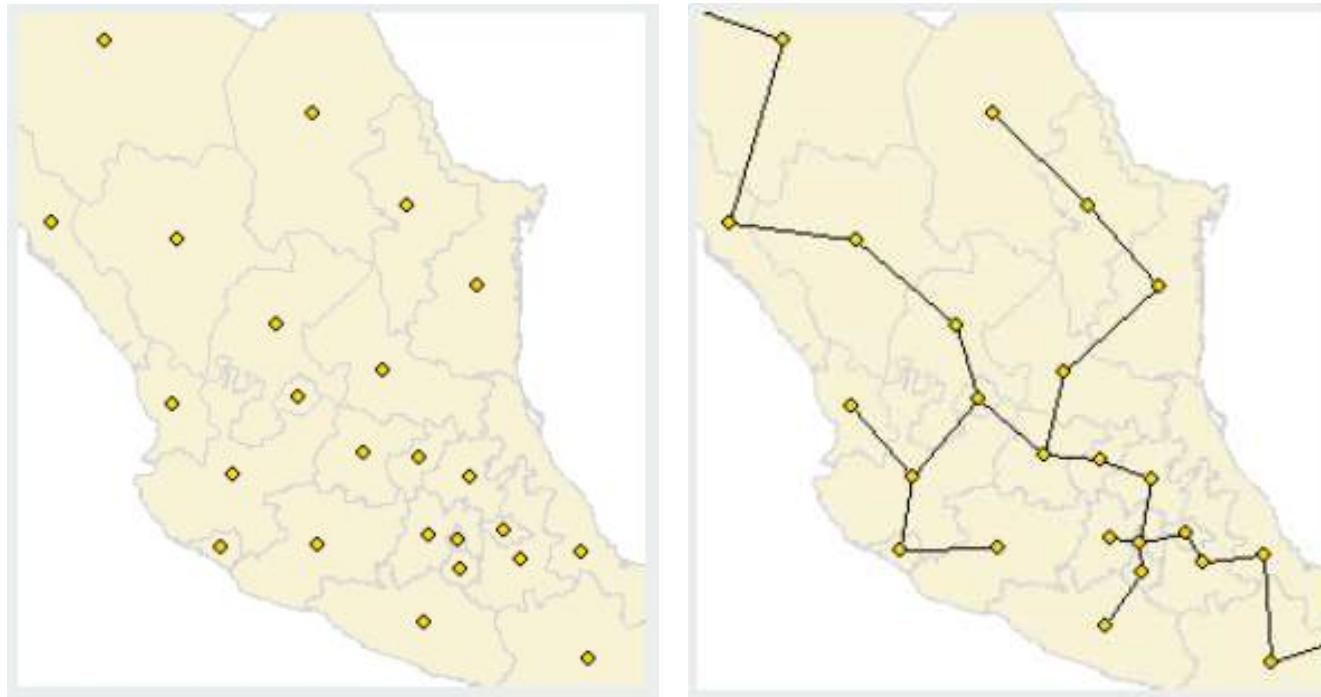
Year	Problem	Time	Discovered By
1976	Planar MST	E	Cheriton-Tarjan
1992	MST Verification	E	Dixon-Rauch-Tarjan
1995	Randomized MST	E	Karger-Klein-Tarjan



related problems

Euclidean MST

- Euclidean MST. Given N points in the plane, find MST connecting them.
 - ◆ Distances between point pairs are Euclidean distances.



- Brute force. Compute $N^2 / 2$ distances and run Prim's algorithm.
- Ingenuity. Exploit geometry and do it in $O(N \log N)$
 - ◆ [stay tuned for geometric algorithms]

Scientific application: clustering

- k-clustering. Divide a set of objects classify into k coherent groups.
- distance function. numeric value specifying "closeness" of two objects.

- Fundamental problem.
 - ◆ Divide into clusters so that points in different clusters are far apart.

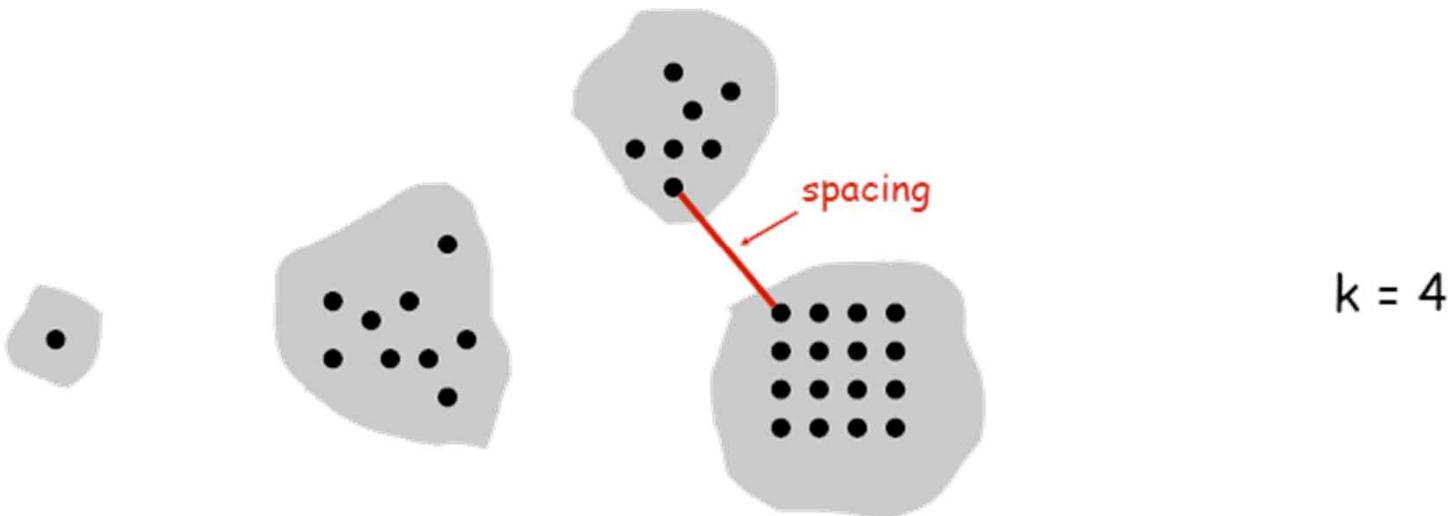
Outbreak of cholera deaths in London in 1850s.
Reference: Nina Mishra, HP Labs

- Applications.
 - ◆ Routing in mobile ad hoc networks.
 - ◆ Identify patterns in gene expression.
 - ◆ Document categorization for web search.
 - ◆ Similarity searching in medical image databases
 - ◆ Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.



k-clustering of maximum spacing

- k-clustering. Divide a set of objects classify into **k** coherent groups.
- distance function. numeric value specifying "closeness" of two objects.
- Spacing. numeric value specifying "closeness" of two objects.
- k-clustering of maximum spacing.
 - ◆ Given an integer k , find a k -clustering such that spacing is maximized.

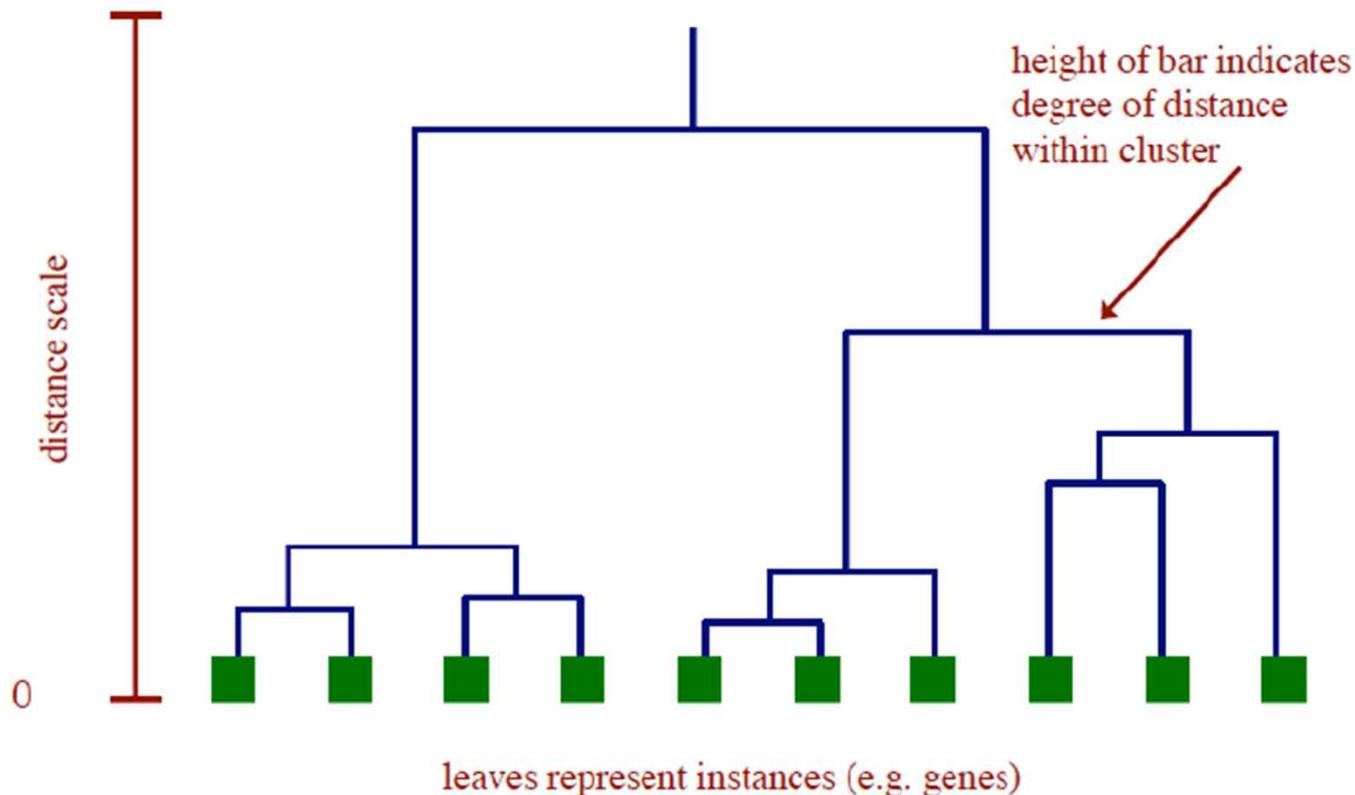


Single-link clustering algorithm

- "Well-known" algorithm for single-link clustering:
 - ◆ Form V clusters of one object each.
 - ◆ Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
 - ◆ Repeat until there are exactly k clusters.
- Observation. This procedure is precisely Kruskal's algorithm
 - ◆ stop when there are k connected components.
- Property. Kruskal's algorithm finds a k -clustering of maximum spacing.

Clustering application: dendograms

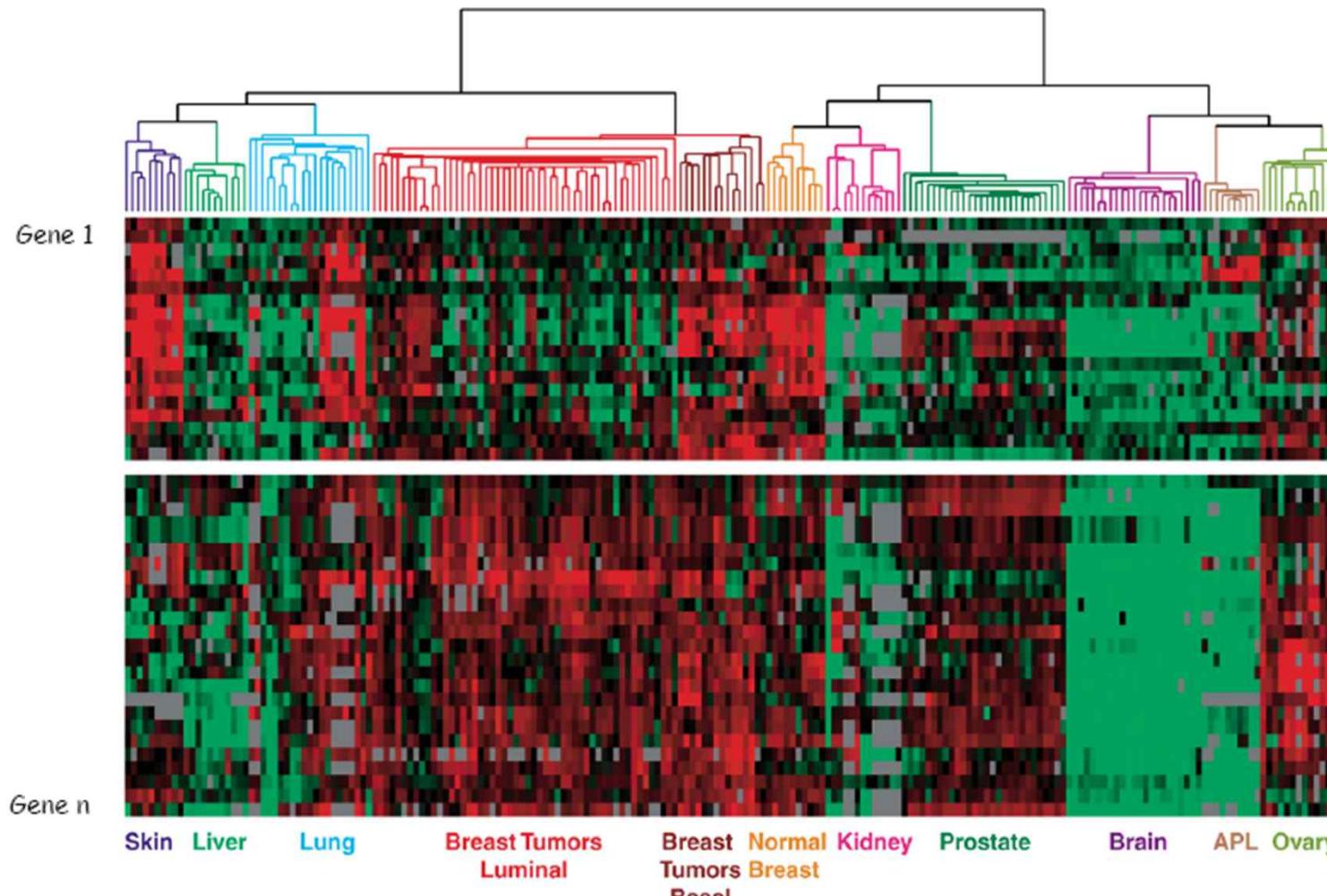
- Dendrogram. Scientific visualization of hypothetical sequence of evolutionary events.
 - ◆ Leaves = genes.
 - ◆ Internal nodes = hypothetical ancestors.



Reference: <http://www.biostat.wisc.edu/bmi576/fall-2003/lecture13.pdf>

Dendrogram of cancers in human

- Tumors in similar tissues cluster together.



Reference: Botstein & Brown group

gene expressed
■ gene not expressed

Algorithms

Data Structures & Algorithms

13. Shortest Paths

- Dijkstra's algorithm
- implementation
- negative weights



Slides are Reformatted From Lecture Note of Algorithms Course
by Robert Sedgewick, Princeton University, Fall, 2008.

Edsger W. Dijkstra: a few select quotes

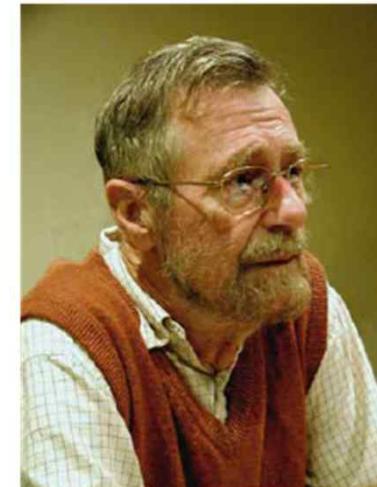
The question of whether computers can think is like
the question of whether submarines can swim.

Do only what only you can do.

In their capacity as a tool, computers will be but a
ripple on the surface of our culture. In their
capacity as intellectual challenge, they are without
precedent in the cultural history of mankind.

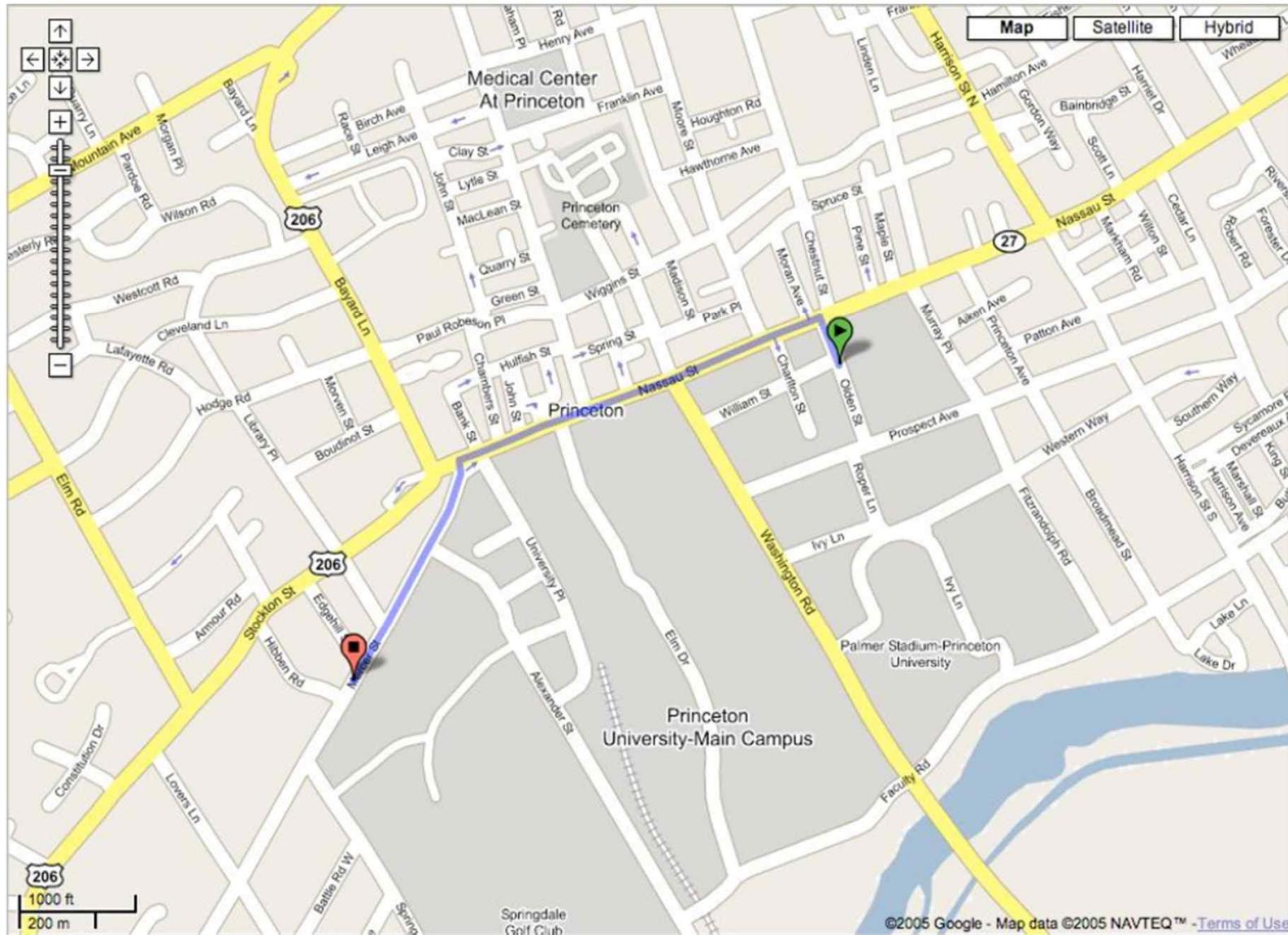
The use of COBOL cripples the mind; its teaching
should, therefore, be regarded as a criminal offence.

APL is a mistake, carried through to perfection. It is
the language of the future for the programming
techniques of the past: it creates a new generation
of coding bums.



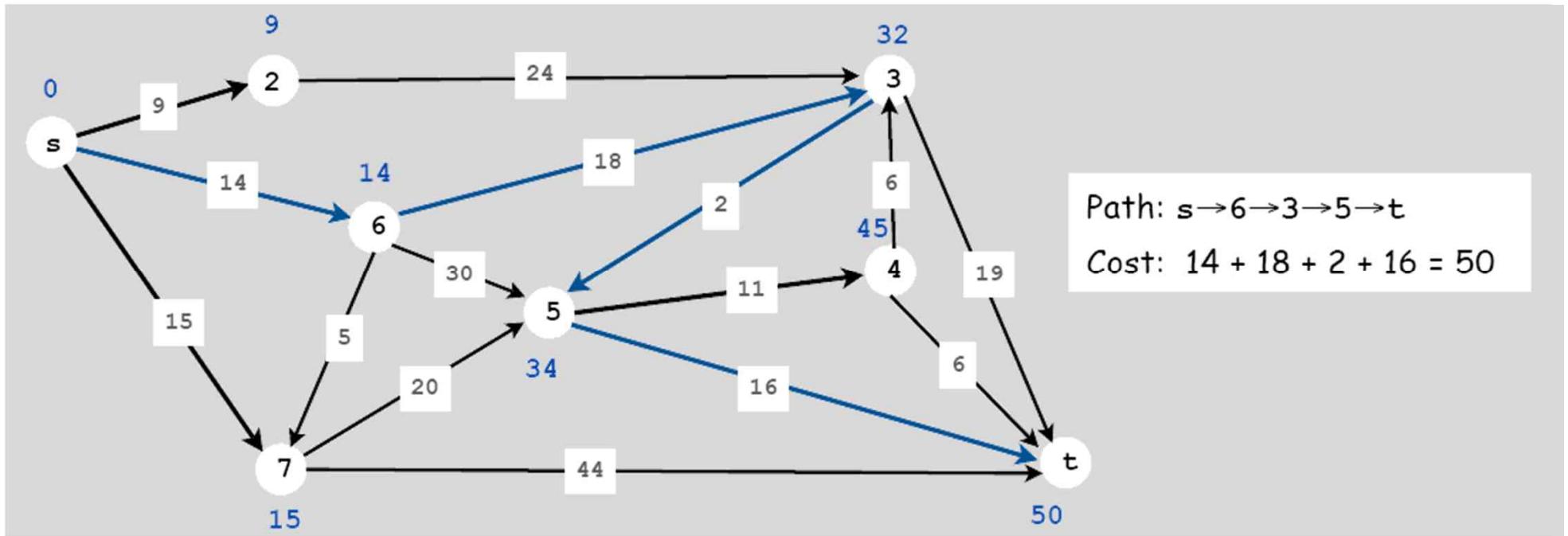
Edsger Dijkstra
Turing award 1972

Shortest paths in a weighted digraph



Shortest paths in a weighted digraph

- Given a weighted digraph, find the shortest directed path from s to t .
cost of path = sum of edge costs in path



- Note: weights are arbitrary numbers
 - not necessarily distances
 - need not satisfy the triangle inequality
 - Ex: airline fares [stay tuned for others]

Versions

- source-target (s-t)
- single source
- all pairs.
- nonnegative edge weights
- arbitrary weights
- Euclidean weights.

Early history of shortest paths algorithms

- Shimbel (1955). Information networks.
- Ford (1956). RAND, economics of transportation.
- Leyzorek, Gray, Johnson, Ladew, Meaker, Petry, Seitz (1957).
 - ◆ Combat Development Dept. of the Army Electronic Proving Ground.
- Dantzig (1958). Simplex method for linear programming.
- Bellman (1958). Dynamic programming.
- Moore (1959). Routing long-distance telephone calls for Bell Labs.
- Dijkstra (1959). Simpler and faster version of Ford's algorithm.

Applications

- Shortest-paths is a broadly useful problem-solving model
 - ◆ Maps
 - ◆ Robot navigation.
 - ◆ Texture mapping.
 - ◆ Typesetting in TeX.
 - ◆ Urban traffic planning.
 - ◆ Optimal pipelining of VLSI chip.
 - ◆ Subroutine in advanced algorithms.
 - ◆ Telemarketer operator scheduling.
 - ◆ Routing of telecommunications messages.
 - ◆ Approximating piecewise linear functions.
 - ◆ Network routing protocols (OSPF, BGP, RIP).
 - ◆ Exploiting arbitrage opportunities in currency exchange.
 - ◆ Optimal truck routing through given traffic congestion pattern.

Reference: Network Flows: Theory, Algorithms, and Applications, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993.

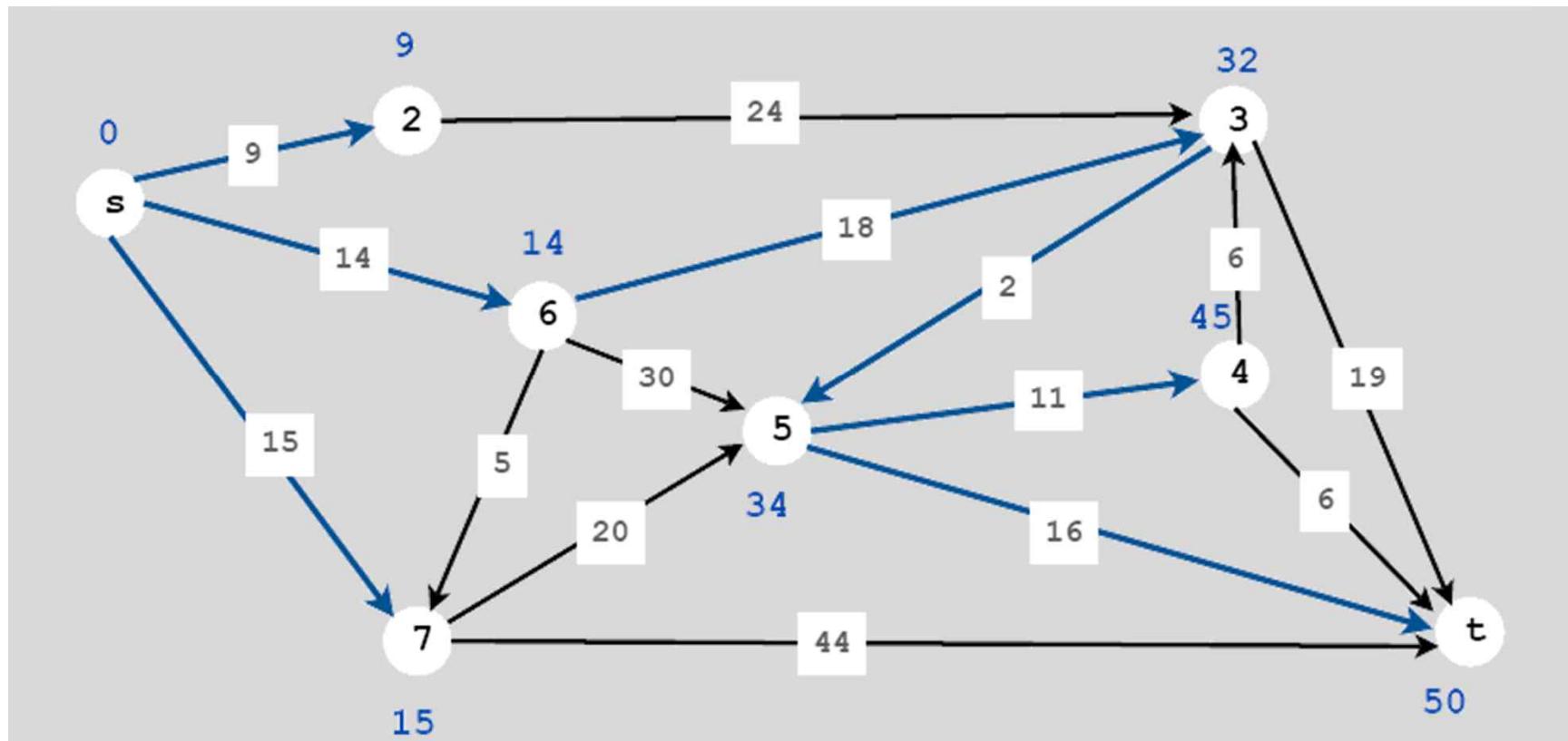


13. Shortest Paths

- Dijkstra's algorithm
- implementation
- negative weights

Single-source shortest-paths

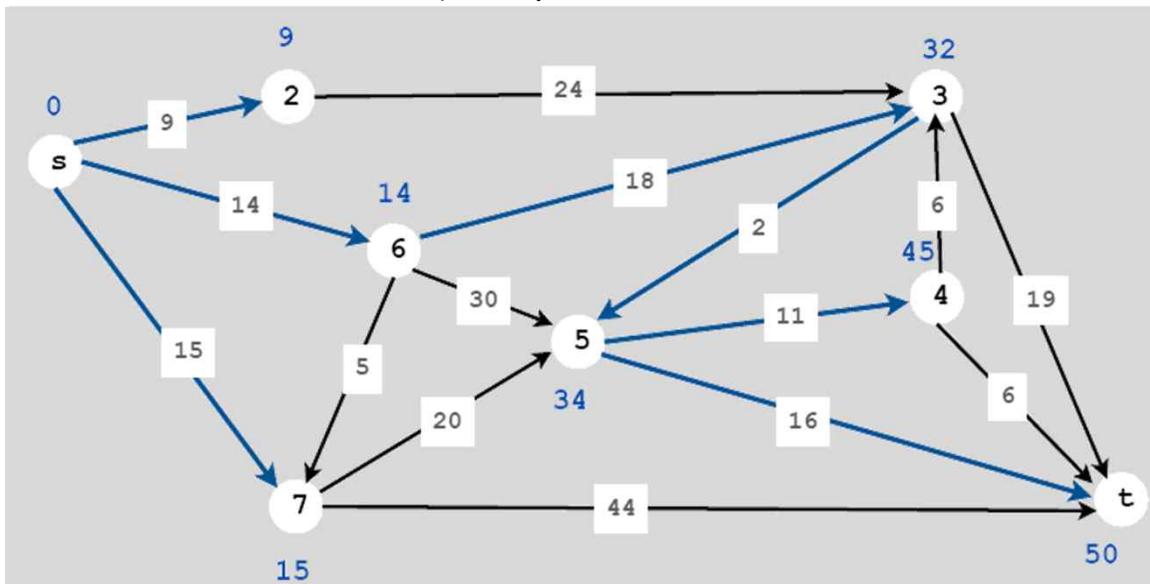
- Given. Weighted digraph, single source s .
- Distance from s to v : length of the shortest path from s to v .
- Goal. Find distance (and shortest path) from s to every other vertex.



- Shortest paths form a tree

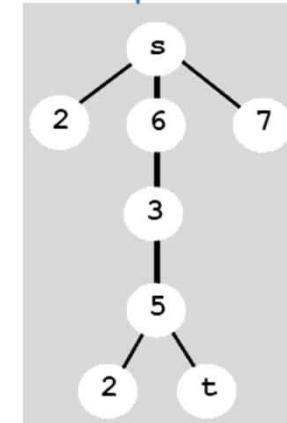
Single-source shortest-paths: basic plan

- Goal: Find distance (and shortest path) from s to every other vertex.
- Design pattern:
 - ◆ ShortestPaths class (WeightedDigraph client)
 - ◆ instance variables: vertex-indexed arrays $\text{dist}[]$ and $\text{pred}[]$
 - ◆ client query methods return distance and path iterator



v	s	2	3	4	5	6	7	t
dist[]	0	9	32	45	34	14	15	50
pred[]	0	0	6	5	3	0	0	5

shortest path tree
(parent-link representation)



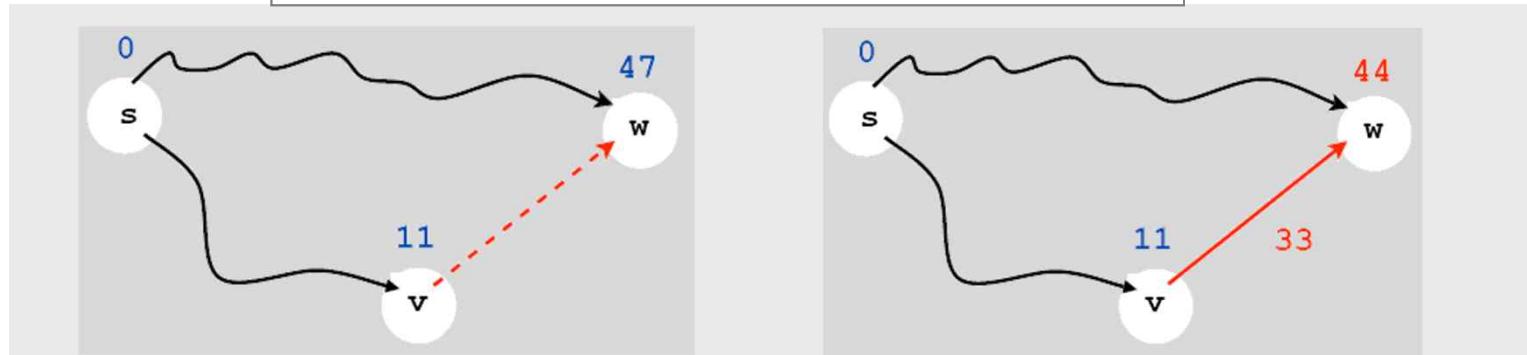
- Note:

- ◆ Same pattern as Prim, DFS, BFS; BFS works when weights are all 1.

Edge relaxation

- For all v , $\text{dist}[v]$ is the length of some path from s to v .
- Relaxation along edge e from v to w .
 - ◆ $\text{dist}[v]$ is length of some path from s to v
 - ◆ $\text{dist}[w]$ is length of some path from s to w
 - ◆ if $v-w$ gives a shorter path to w through v , update $\text{dist}[w]$ and $\text{pred}[w]$

```
if (dist[w] > dist[v] + e.weight())
{
    dist[w] = dist[v] + e.weight();
    pred[w] = e;
}
```

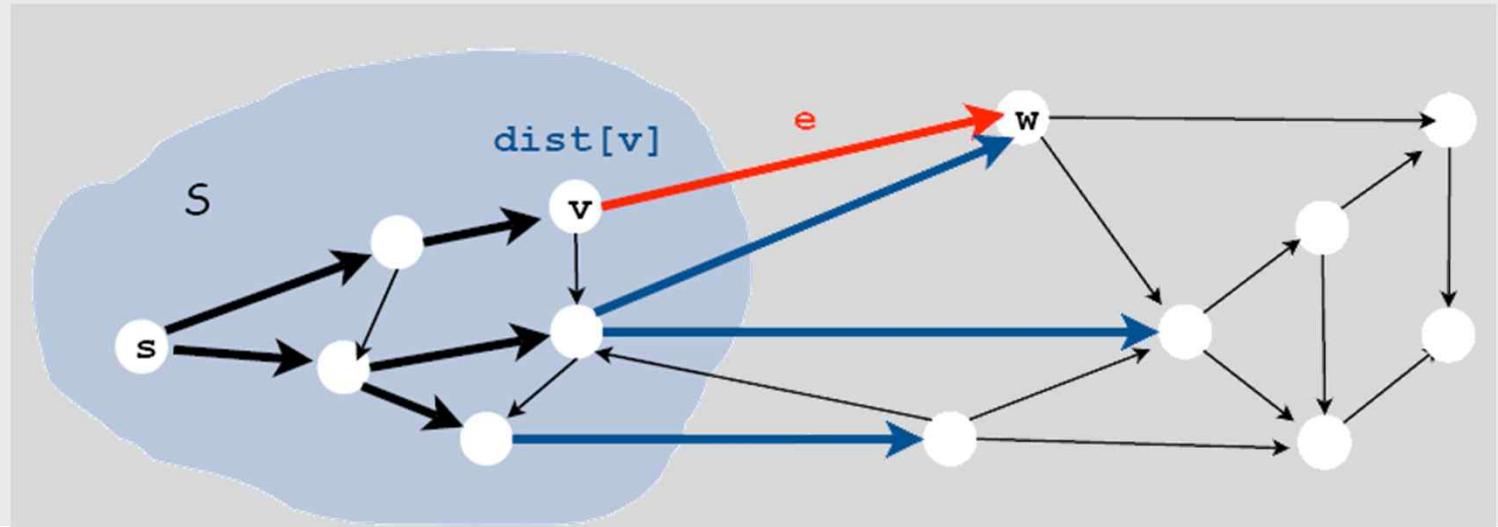


- Relaxation sets $\text{dist}[w]$ to the length of a shorter path from s to w
 - ◆ (if $v-w$ gives one)

Dijkstra's algorithm

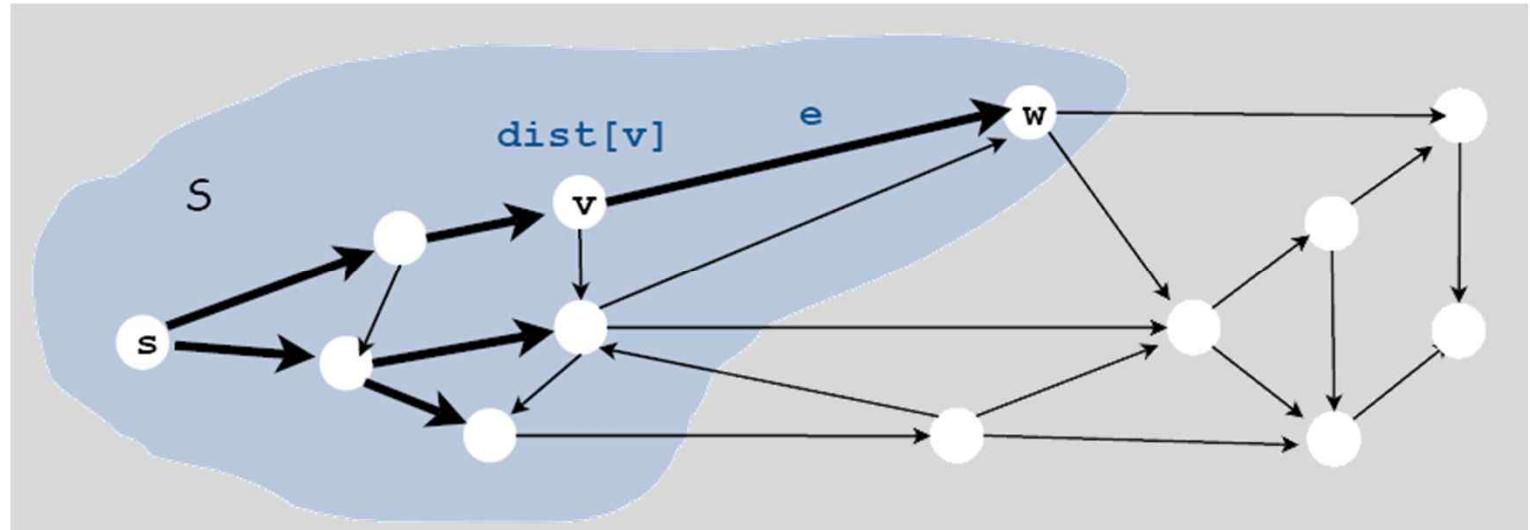
- S : set of vertices for which the shortest path length from s is known.
- **Invariant:** for v in S ,
 - ◆ $\text{dist}[v]$ is the length of the shortest path from s to v .

- Initialize S to s , $\text{dist}[s]$ to 0, $\text{dist}[v]$ to ∞ for all other v
- Repeat until S contains all vertices connected to s
 - ◆ find e with v in S and w in S' that minimizes $\text{dist}[v] + e.\text{weight}()$
 - ◆ relax along that edge
 - ◆ add w to S



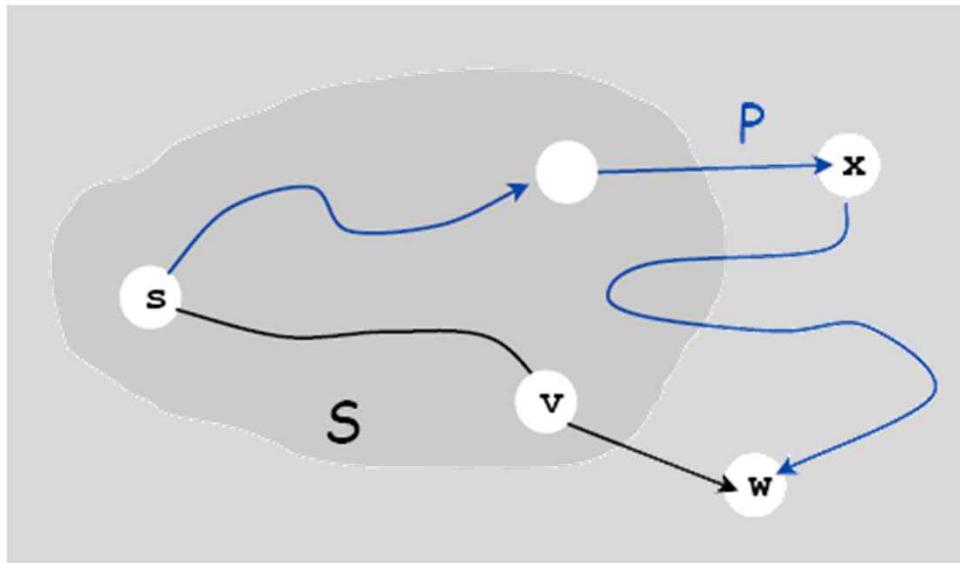
Dijkstra's algorithm

- S : set of vertices for which the shortest path length from s is known.
- Invariant: for v in S ,
 - ◆ $\text{dist}[v]$ is the length of the shortest path from s to v .
- Initialize S to s , $\text{dist}[s]$ to 0, $\text{dist}[v]$ to ∞ for all other v
- Repeat until S contains all vertices connected to s
 - ◆ find e with v in S and w in S' that minimizes $\text{dist}[v] + e.\text{weight}()$
 - ◆ relax along that edge
 - ◆ add w to S

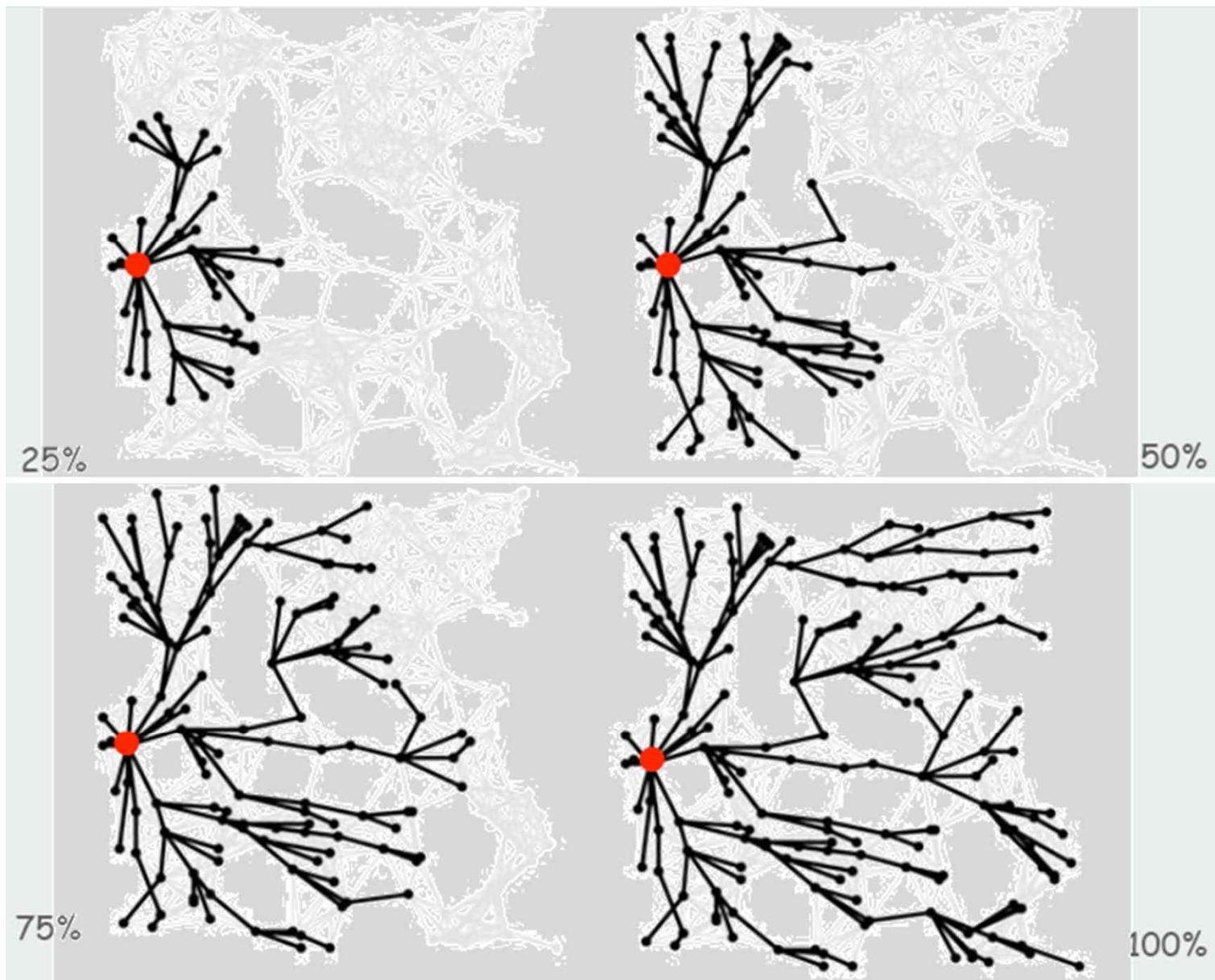


Dijkstra's algorithm proof of correctness

- S : set of vertices for which the shortest path length from s is known.
- **Invariant:** for v in S ,
 - ◆ $\text{dist}[v]$ is the length of the shortest path from s to v .
- **Pf.** (by induction on $|S|$)
 - ◆ Let w be next vertex added to S .
 - ◆ Let P^* be the $s-w$ path through v .
 - ◆ Consider any other $s-w$ path P , and let x be first node on path outside S .
 - ◆ P is already longer than P^* as soon as it reaches x by greedy choice.



Shortest Path Tree



13. Shortest Paths

- Dijkstra's algorithm
- implementation
- negative weights

Weighted directed edge data type

```
public class Edge implements Comparable<Edge>
{
    public final int v, int w;
    public final double weight;

    public Edge(int v, int w, double weight)
    {
        this.v = v;
        this.w = w;
        this.weight = weight;
    }

    public int from()
    {   return v; }

    public int to()
    {   return w; }

    public int weight()
    {   return weight; }

    public int compareTo(Edge that)
    {
        if      (this.weight < that.weight) return -1;
        else if (this.weight > that.weight) return +1;
        else                                return 0;
    }
}
```

code is the same as for
(undirected) WeightedGraph

except
from() and to() replace
either() and other()



Weighted digraph data type

- Identical to WeightedGraph but just one representation of each Edge.

```
public class WeightedDigraph
{
    private int V;
    private SET<Edge>[] adj;

    public Graph(int V)
    {
        this.V = V;
        adj = (SET<Edge>[]) new SET[V];
        for (int v = 0; v < V; v++)
            adj[v] = new SET<Edge>();
    }

    public void addEdge(Edge e)
    {
        int v = e.from();
        adj[v].add(e);
    }

    public Iterable<Edge> adj(int v)
    {   return adj[v];  }
}
```



}

Dijkstra's algorithm: implementation approach

- Initialize S to s , $\text{dist}[s]$ to 0, $\text{dist}[v]$ to ∞ for all other v
- Repeat until S contains all vertices connected to s
 - ◆ find $v-w$ with v in S and w in S' that minimizes $\text{dist}[v] + \text{weight}[v-w]$
 - ◆ relax along that edge
 - ◆ add w to S
- Idea 1 (easy): Try all edges
- Total running time proportional to VE

Dijkstra's algorithm: implementation approach

- Initialize S to s , $\text{dist}[s]$ to 0, $\text{dist}[v]$ to ∞ for all other v
- Repeat until S contains all vertices connected to s
 - ◆ find $v-w$ with v in S and w in S' that minimizes $\text{dist}[v] + \text{weight}[v-w]$
 - ◆ relax along that edge
 - ◆ add w to S
- Idea 2 (Dijkstra) : maintain these invariants
 - ◆ for v in S , $\text{dist}[v]$ is the length of **the shortest** path from s to v .
 - ◆ for w in S' , $\text{dist}[w]$ minimizes $\text{dist}[v] + \text{weight}[v-w]$.
- Two implications
 - ◆ find $v-w$ in V steps (smallest $\text{dist}[]$ value among vertices in S')
 - ◆ update $\text{dist}[]$ in at most V steps (check neighbors of w)
- Total running time proportional to V^2

Dijkstra's algorithm implementation

- Initialize S to s , $\text{dist}[s]$ to 0, $\text{dist}[v]$ to ∞ for all other v
- Repeat until S contains all vertices connected to s
 - ◆ find $v-w$ with v in S and w in S' that minimizes $\text{dist}[v] + \text{weight}[v-w]$
 - ◆ relax along that edge
 - ◆ add w to S
- Idea 3 (modern implementations):
 - ◆ for all v in S , $\text{dist}[v]$ is the length of the shortest path from s to v .
 - ◆ use a **priority queue** to find the edge to relax

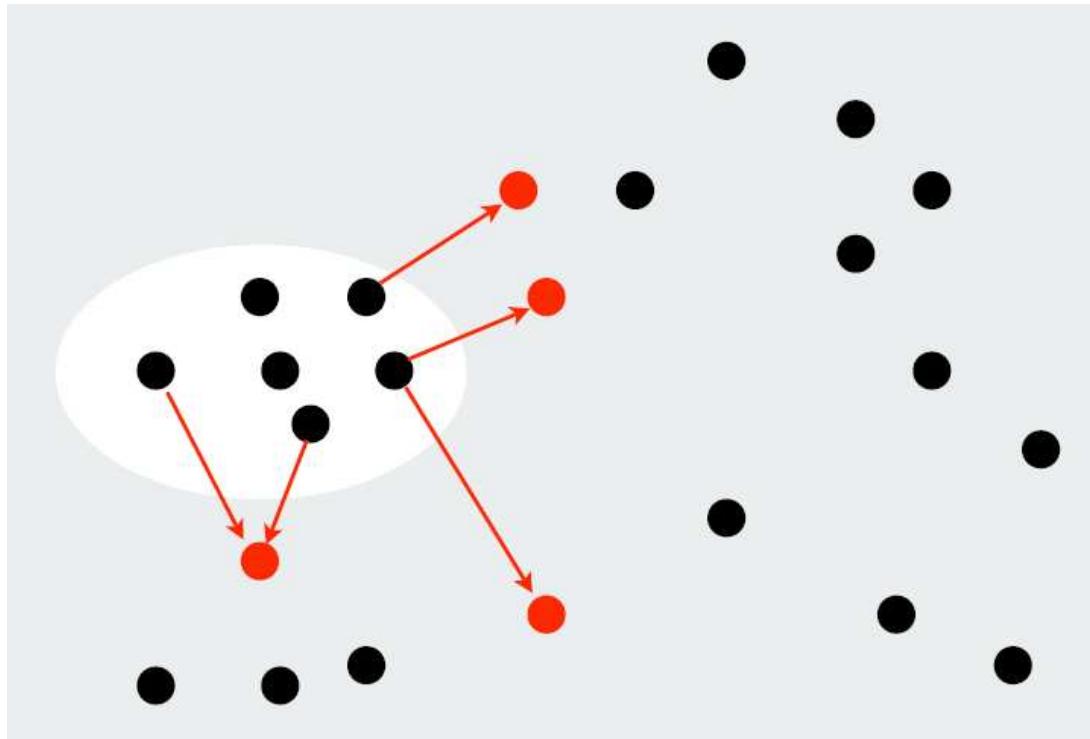
	sparse	dense
easy	V^2	EV
Dijkstra	V^2	V^2
modern	$E \lg E$	$E \lg E$

- Total running time proportional to $E \lg E$

Dijkstra's algorithm implementation

Q. What goes onto the priority queue?

A. Fringe vertices connected by a single edge to a vertex in S



Starting to look familiar?

Lazy implementation of Prim's MST algorithm

```
public class LazyPrim
{
    Edge[] pred = new Edge[G.V()];
    public LazyPrim(WeightedGraph G)
    {
        boolean[] marked = new boolean[G.V()];           ← marks vertices in MST
        double[] dist = new double[G.V()];               ← distance to MST
        for (int v = 0; v < G.V(); v++)
            dist[v] = Double.POSITIVE_INFINITY;          ← edges to MST
        MinPQplus<Double, Integer> pq;
        pq = new MinPQplus<Double, Integer>();          ← key-value PQ
        dist[s] = 0.0;
        pq.put(dist[s], s);
        while (!pq.isEmpty())
        {
            int v = pq.delMin();                         ← get next vertex
            if (marked[v]) continue;                     ← ignore if already in MST
            marked(v) = true;
            for (Edge e : G.adj(v))
            {
                int w = e.other(v);
                if (!marked[w] && (dist[w] > e.weight() ))
                {
                    dist[w] = e.weight();
                    pred[w] = e;
                    pq.insert(dist[w], w);                 ← add to PQ any vertices
                                                               brought closer to S by v
                }
            }
        }
    }
}
```

Lazy implementation of Dijkstra's SPT algorithm

```
public class LazyDijkstra
{
    double[] dist = new double[G.V()];
    Edge[] pred = new Edge[G.V()];
    public LazyDijkstra(WeightedDigraph G, int s)
    {
        boolean[] marked = new boolean[G.V()];
        for (int v = 0; v < G.V(); v++)
            dist[v] = Double.POSITIVE_INFINITY;
        MinPQplus<Double, Integer> pq;
        pq = new MinPQplus<Double, Integer>();
        dist[s] = 0.0;
        pq.put(dist[s], s);
        while (!pq.isEmpty())
        {
            int v = pq.delMin();
            if (marked[v]) continue;
            marked(v) = true;
            for (Edge e : G.adj(v))
            {
                int w = e.to();
                if (dist[w] > dist[v] + e.weight())
                {
                    dist[w] = dist[v] + e.weight();
                    pred[w] = e;
                    pq.insert(dist[w], w);
                }
            }
        }
    }
}
```

code is the same as Prim's (!!)

except

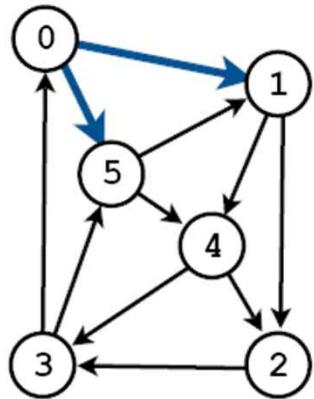
- WeightedDigraph, not WeightedGraph
- weight is distance to s, not to tree
- add client query for distances



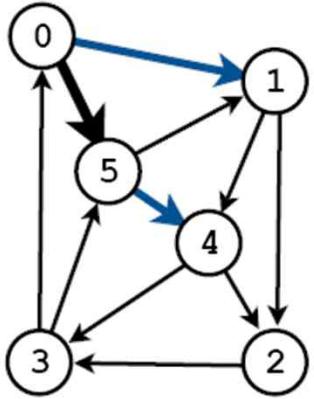
Dijkstra's algorithm example

□ Dijkstra's algorithm. [Dijkstra 1957]

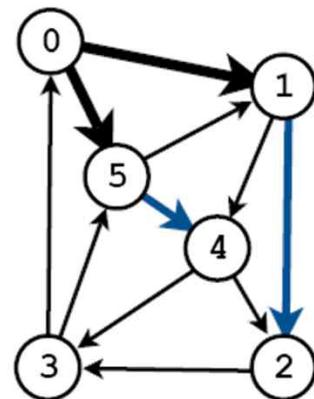
- ◆ Start with vertex 0 and greedily grow tree T. At each step,
 - ◆ add cheapest path ending in an edge that has exactly one endpoint in T.



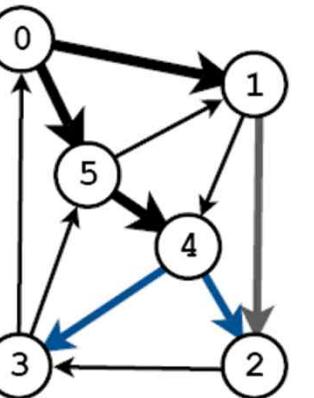
0-5 .29 0-1 .41



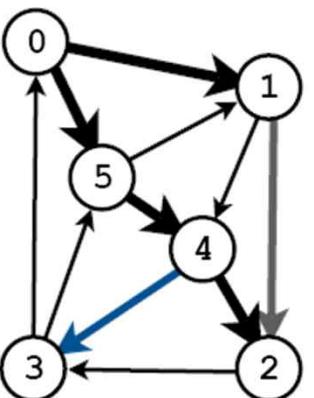
0-1 .41 5-4 .50



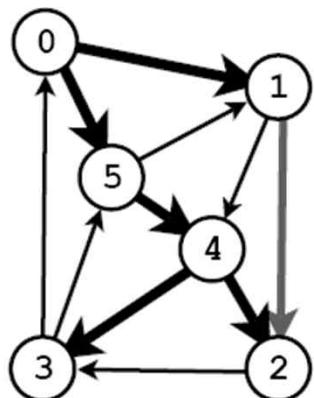
5-4 .50 1-2 .92



4-2 .82 4-3 .86 1-2 .92



4-3 .86 1-2 .92



1-2 .92

Eager implementation of Dijkstra's algorithm

- Use indexed priority queue that supports
 - ◆ contains: is there a key associated with value v in the priority queue?
 - ◆ decrease key: decrease the key associated with value v
- [more complicated data structure, see text]
- Putative “benefit”: reduces PQ size guarantee from E to V
 - ◆ no significant impact on time since $\lg E < 2\lg V$
 - ◆ extra space not important for huge sparse graphs found in practice [PQ size is far smaller than E or even V in practice]
 - ◆ widely used, but practical utility is debatable (as for Prim’s)

Improvements to Dijkstra's algorithm

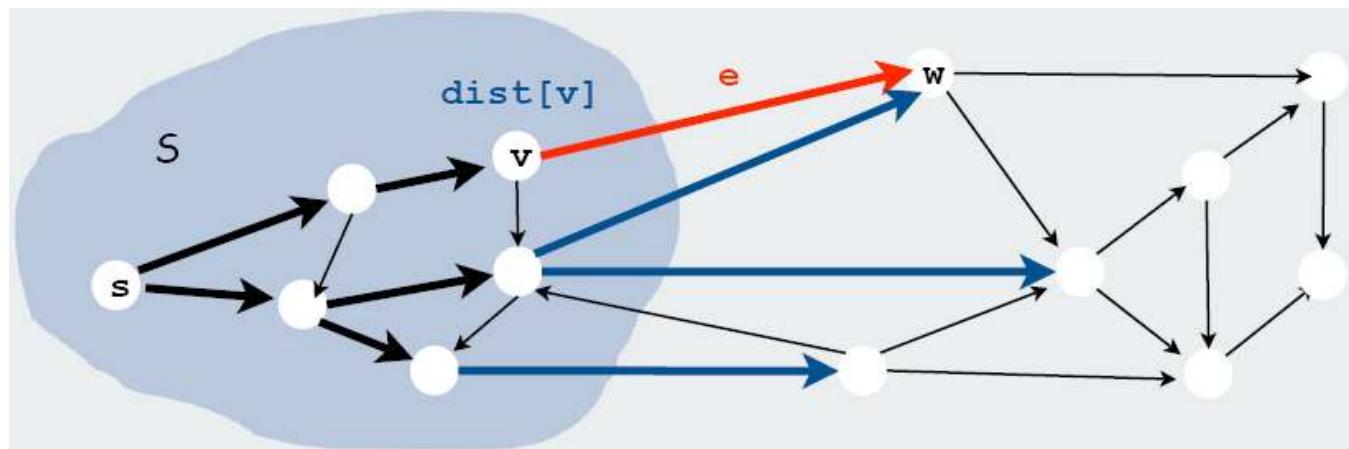
- Use a **d-way heap** (Johnson, 1970s)
 - ◆ easy to implement
 - ◆ reduces costs to $E d \log_d V$
 - ◆ indistinguishable from linear for huge sparse graphs found in practice
- Use a **Fibonacci heap** (Sleator-Tarjan, 1980s)
 - ◆ very difficult to implement
 - ◆ reduces worst-case costs (in theory) to $E + V \lg V$
 - ◆ not quite linear (in theory)
 - ◆ practical utility questionable
- Find an algorithm that provides a linear worst-case guarantee?
 - ◆ [open problem]

Dijkstra's Algorithm: performance summary

- Fringe implementation **directly impacts performance**
- Best choice depends on sparsity of graph.
 - ◆ 2,000 vertices, 1 million edges. heap 2-3x slower than array
 - ◆ 100,000 vertices, 1 million edges. heap gives 500x speedup.
 - ◆ 1 million vertices, 2 million edges. **heap gives 10,000x speedup.**
- Bottom line.
 - ◆ array implementation optimal for dense graphs
 - ◆ binary heap far better for sparse graphs
 - ◆ d-way heap worth the trouble in performance-critical situations
 - ◆ Fibonacci heap best in theory, but not worth implementing

Priority-first search

- Insight: All of our graph-search methods are the same algorithm!
- Maintain a set of explored vertices S
- Grow S by exploring edges with exactly one endpoint leaving S .
 - ◆ DFS. Take edge from vertex which was discovered most recently.
 - ◆ BFS. Take from vertex which was discovered least recently.
 - ◆ Prim. Take edge of minimum weight.
 - ◆ Dijkstra. Take edge to vertex that is closest to s .
 - ◆ ... Gives simple algorithm for many graph-processing problems

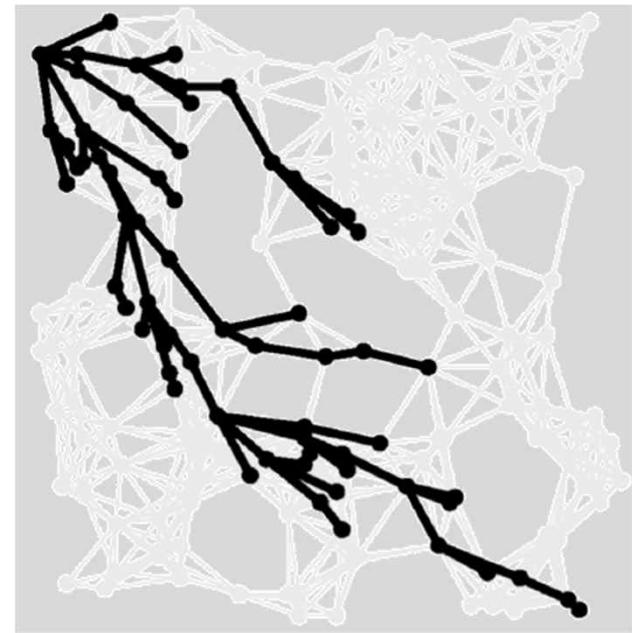


- Challenge:

◆ express this insight in (re)usable Java code

Priority-first search: application example

- Shortest s-t paths in Euclidean graphs (maps)
 - ◆ Vertices are points in the plane.
 - ◆ Edge weights are **Euclidean distances**.
- A sublinear algorithm.
 - ◆ Assume graph is already in memory.
 - ◆ Start Dijkstra at s .
 - ◆ Stop when you reach t .
- Even better: **exploit geometry**
 - ◆ For edge $v-w$, use weight $d(v, w) + d(w, t) - d(v, t)$.
 - ◆ Proof of correctness for Dijkstra still applies.
 - ◆ In practice only $O(V^{1/2})$ vertices examined.
 - ◆ Special case of **A^* algorithm**
- [Practical map-processing programs precompute many of the paths.]



Euclidean distance

13. Shortest Paths

- Dijkstra's algorithm
- implementation
- negative weights

Shortest paths application: Currency conversion

□ Currency conversion. Given currencies and exchange rates, what is best way to convert one ounce of gold to US dollars?

◆ 1 oz. gold → \$327.25.

◆ 1 oz. gold → £208.10 → → \$327.00.

[208.10×1.5714]

◆ 1 oz. gold → 455.2 Francs → 304.39 Euros → \$327.28.

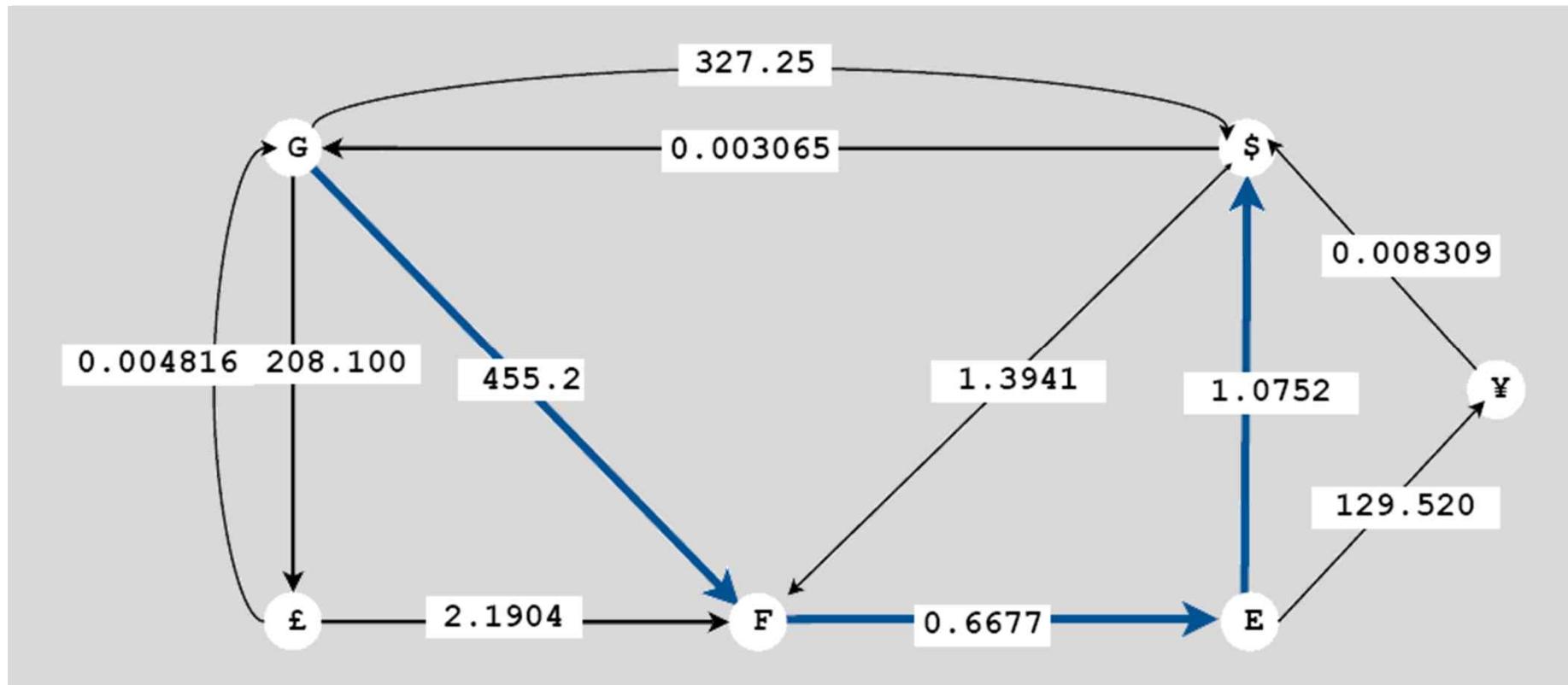
[$455.2 \times .6677 \times 1.0752$]

Currency	£	Euro	¥	Franc	\$	Gold
UK Pound	1.0000	0.6853	0.005290	0.4569	0.6368	208.100
Euro	1.4599	1.0000	0.007721	0.6677	0.9303	304.028
Japanese Yen	189.050	129.520	1.0000	85.4694	120.400	39346.7
Swiss Franc	2.1904	1.4978	0.011574	1.0000	1.3941	455.200
US Dollar	1.5714	1.0752	0.008309	0.7182	1.0000	327.250
Gold (oz.)	0.004816	0.003295	0.0000255	0.002201	0.003065	1.0000

Shortest paths application: Currency conversion

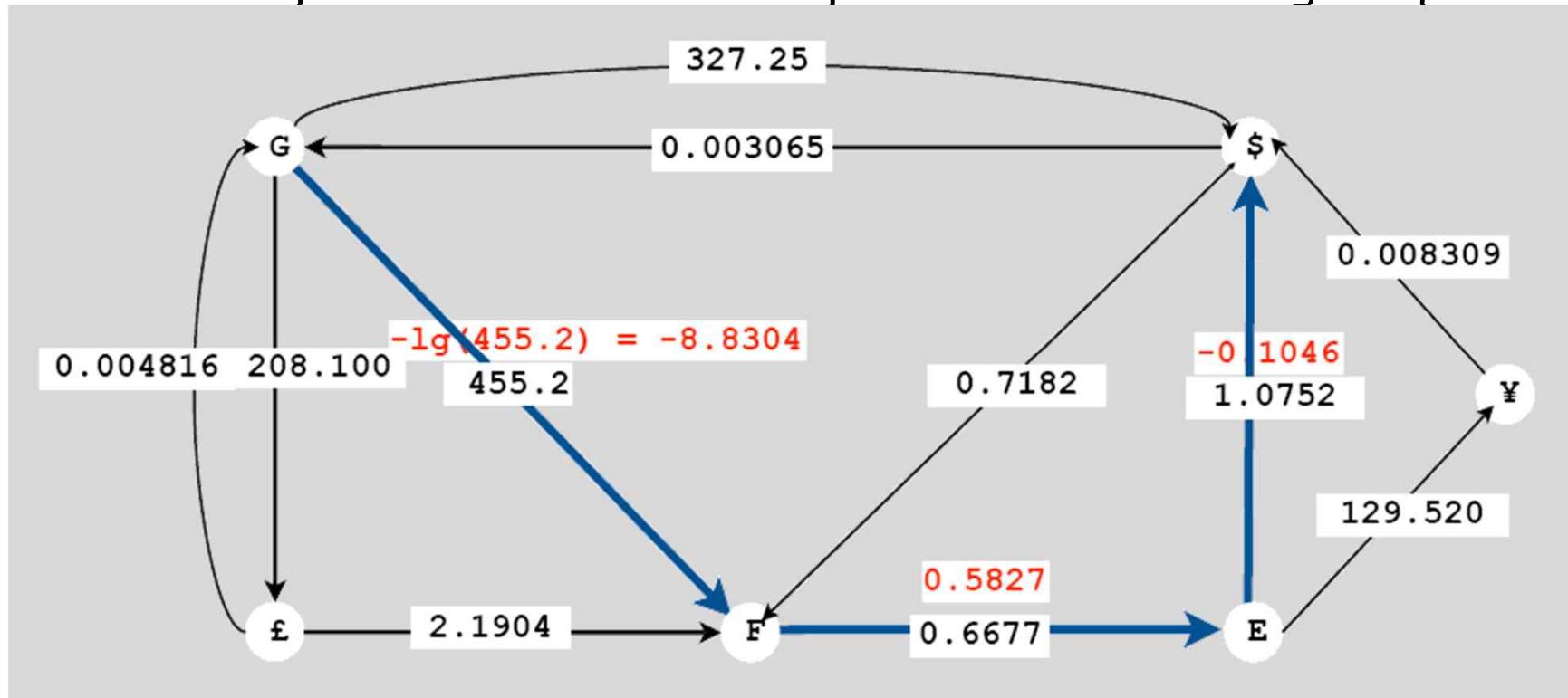
□ Graph formulation.

- ◆ Vertex = currency.
- ◆ Edge = transaction, with weight equal to exchange rate.
- ◆ Find path that maximizes product of weights.



Shortest paths application: Currency conversion

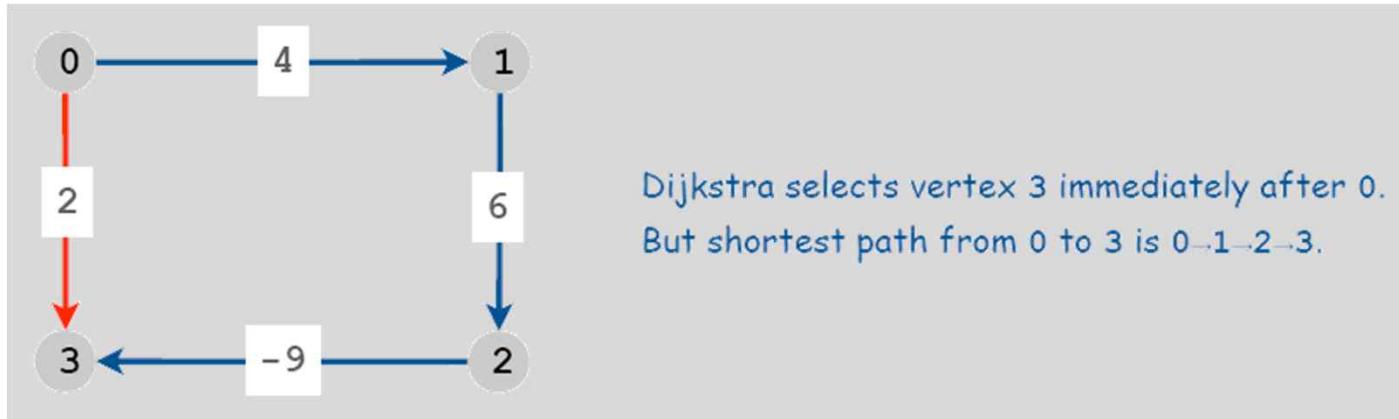
- ☐ Reduce to shortest path problem by taking logs
 - ◆ Let $\text{weight}(v-w) = -\lg$ (exchange rate from currency v to w)
 - ◆ multiplication turns to addition
 - ◆ Shortest path with costs c corresponds to best exchange sequence.



- Challenge. Solve shortest path problem with **negative** weights.

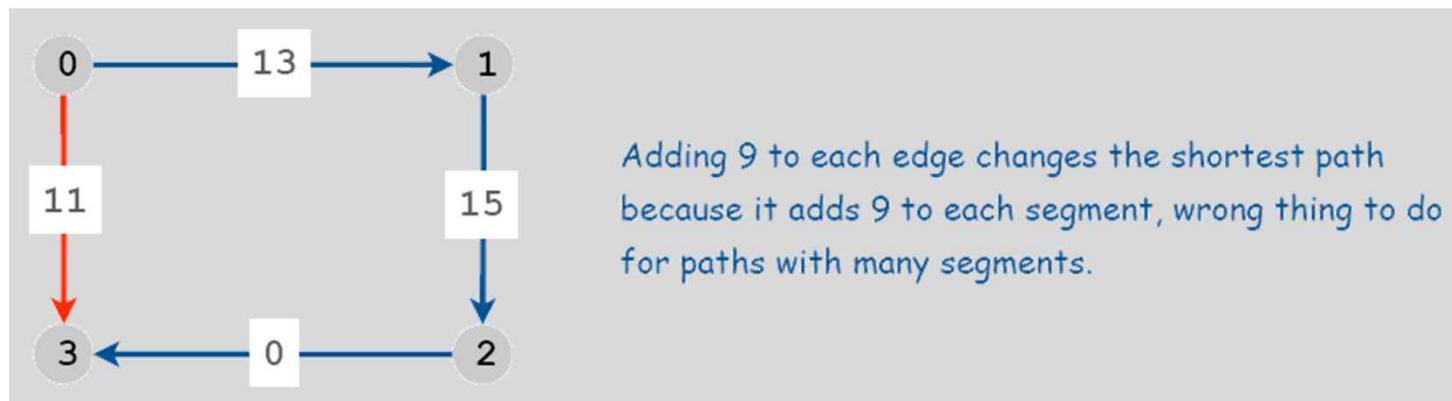
Shortest paths with negative weights: failed attempts

- Dijkstra. Doesn't work with negative edge weights.



- Re-weighting.

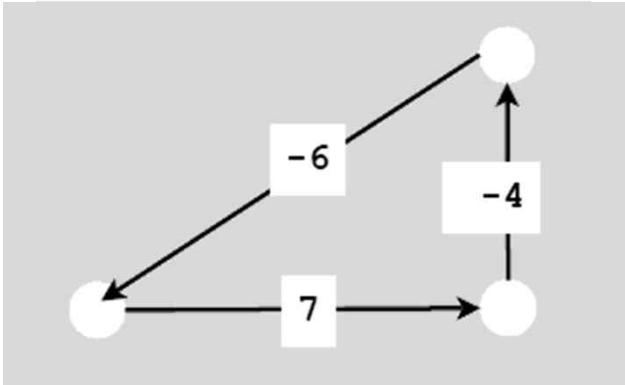
- ◆ Adding a constant to every edge weight also doesn't work.



- Bad news: need a different algorithm.

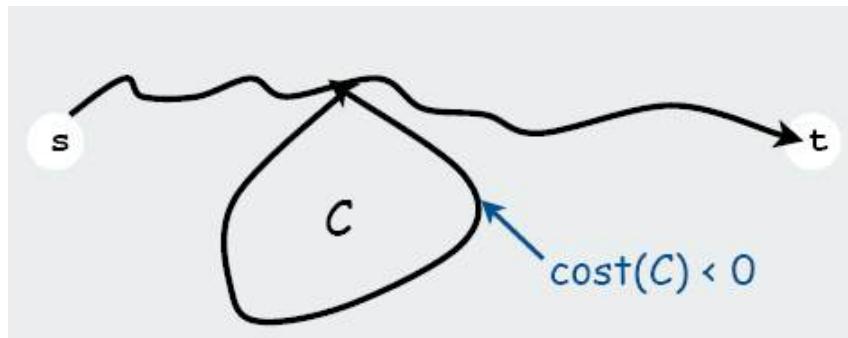
Shortest paths with negative weights: negative cycles

- ❑ Negative cycle. Directed cycle whose sum of edge weights is negative.



- ❑ Observations.

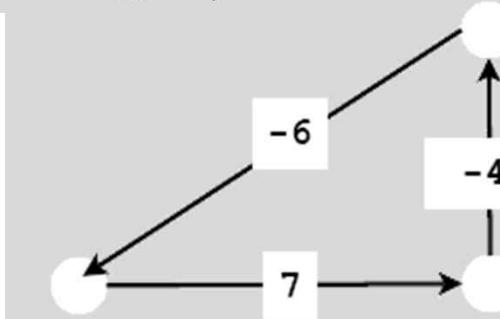
- ◆ If negative cycle C on path from s to t , then shortest path can be made arbitrarily negative by spinning around cycle
- ◆ There exists a shortest s - t path that is simple.



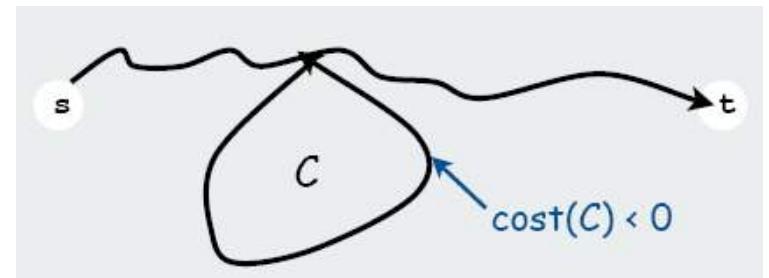
- ❑ Worse news: need a different problem

Shortest paths with negative weights

- Problem 1. Does a given digraph contain a negative cycle?



- Problem 2. Find the shortest simple path from s to t .

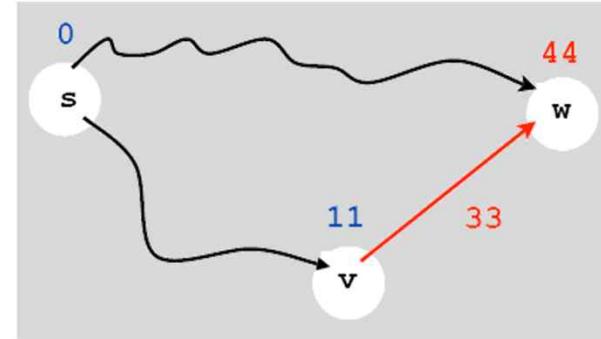
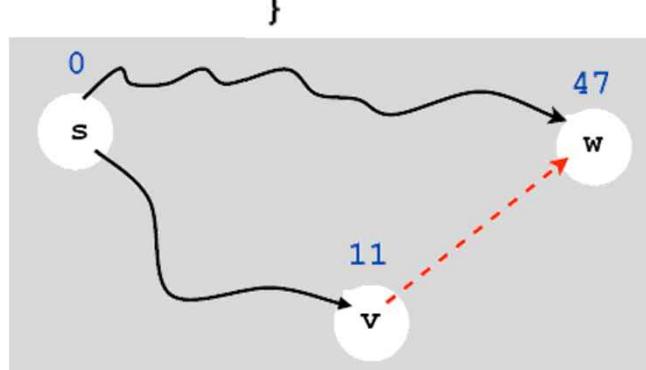


- Bad news: Problem 2 is intractable
- Good news: Can solve problem 1 in $O(VE)$ steps
- Good news: Same algorithm solves problem 2 if no negative cycle
- **Bellman-Ford algorithm**
 - ◆ detects a negative cycle if any exist
 - ◆ finds shortest simple path if no negative cycle exists

Edge relaxation

- For all v , $\text{dist}[v]$ is the length of **some** path from s to v .
- Relaxation along edge e from v to w .
 - ◆ $\text{dist}[v]$ is length of some path from s to v
 - ◆ $\text{dist}[w]$ is length of some path from s to w
 - ◆ if $v-w$ gives a shorter path to w through v , update $\text{dist}[w]$ and $\text{pred}[w]$

```
if (dist[w] > dist[v] + e.weight())
{
    dist[w] = dist[v] + e.weight();
    pred[w] = e;
}
```



- Relaxation sets $\text{dist}[w]$ to the length of a **shorter** path from s to w (if $v-w$ gives one)

Shortest paths with negative weights: dynamic programming algorithm

□ A simple solution that works!

- ◆ Initialize $\text{dist}[v] = \infty$, $\text{dist}[s] = 0$.
- ◆ Repeat V times: relax each edge e .

```
for (int i = 1; i <= G.V(); i++)           phase i
    for (int v = 0; v < G.V(); v++)
        for (Edge e : G.adj(v))
        {
            int w = e.to();
            if (dist[w] > dist[v] + e.weight()) ← relax v-w
            {
                dist[w] = dist[v] + e.weight()
                pred[w] = e;
            }
        }
```

Shortest paths with negative weights: dynamic programming algorithm

- Running time proportional to $E V$
- Invariant. At end of phase i , $\text{dist}[v] \leq$ length of any path from s to v using at most i edges.
- Theorem. If there are no negative cycles, upon termination $\text{dist}[v]$ is the length of the shortest path from s to v .



and $\text{pred}[]$ gives the shortest paths

Shortest paths with negative weights: Bellman-Ford-Moore algorithm

□ Observation. If $\text{dist}[v]$ doesn't change during phase i , no need to relax any edge leaving v in phase $i+1$.

□ FIFO implementation. Maintain queue of vertices whose distance changed.



Be careful to keep at most one copy of each vertex on queue

□ Running time.

- ◆ still could be proportional to EV in worst case
- ◆ much faster than that in practice

Shortest paths with negative weights: Bellman-Ford-Moore algorithm

- Initialize $\text{dist}[v] = \infty$ and $\text{marked}[v] = \text{false}$ for all vertices v .

```
Queue<Integer> q = new Queue<Integer>();
marked[s] = true;
dist[s] = 0;
q.enqueue(s);
```

```
while (!q.isEmpty())
{
    int v = q.dequeue();
    marked[v] = false;
    for (Edge e : G.adj(v))
    {
        int w = e.target();
        if (dist[w] > dist[v] + e.weight())
        {
            dist[w] = dist[v] + e.weight();
            pred[w] = e;
            if (!marked[w])
            {
                marked[w] = true;
                q.enqueue(w);
            }
        }
    }
}
```

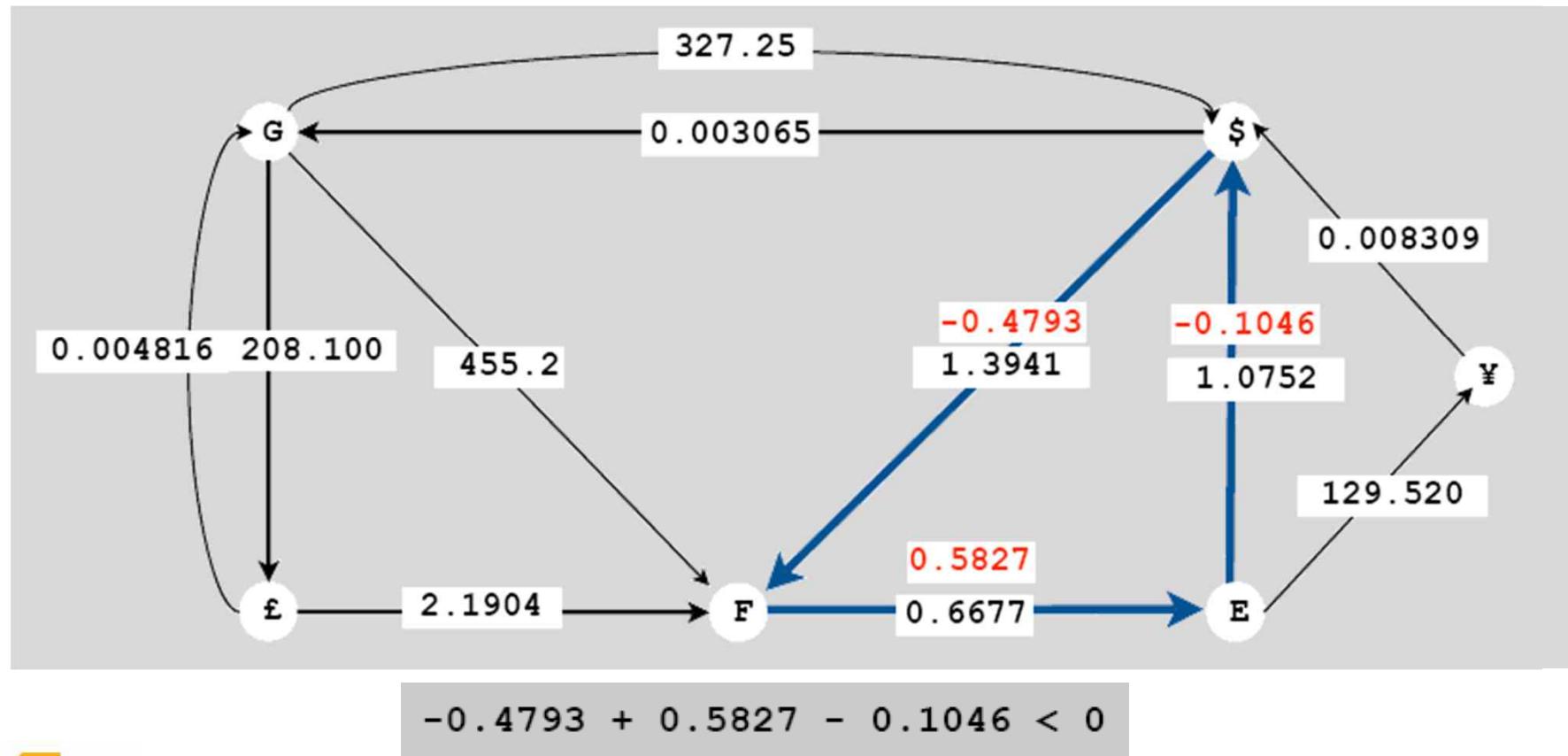
Single Source Shortest Paths Implementation: Cost Summary

	algorithm	worst case	typical case
nonnegative costs	Dijkstra (classic)	V^2	V^2
	Dijkstra (heap)	$E \lg E$	(E)
no negative cycles	Dynamic programming	EV	EV
	Bellman-Ford-Moore	EV	(E)

- Remark 1. Negative weights makes the problem harder.
- Remark 2. Negative cycles makes the problem intractable.

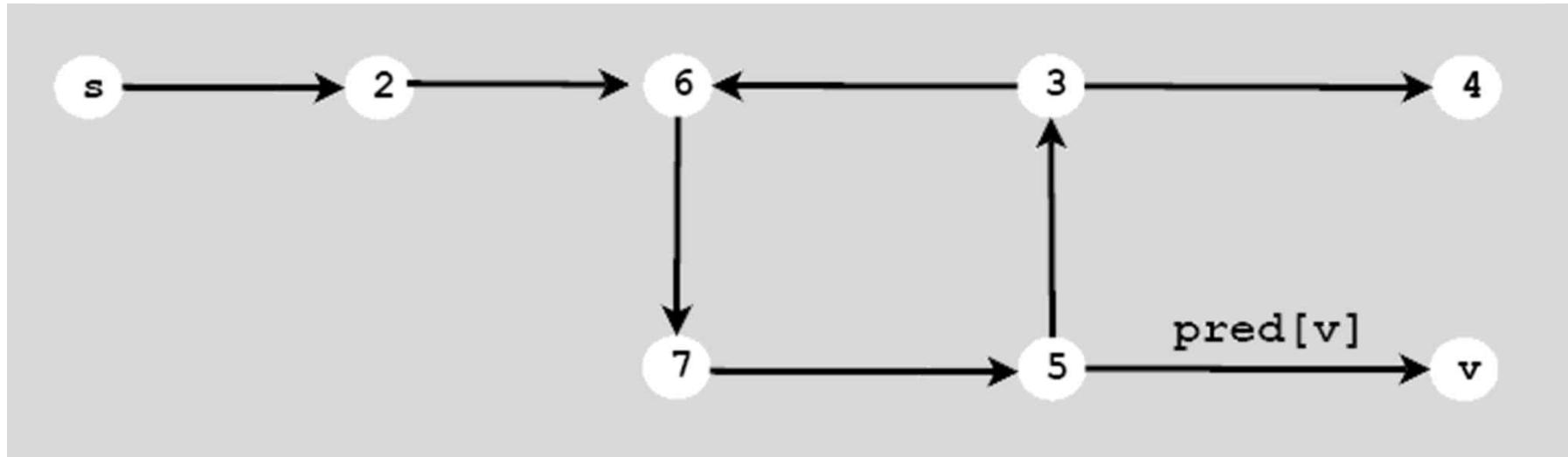
Shortest paths application: arbitrage

- ☐ Is there an arbitrage opportunity in currency graph?
 - ◆ Ex: \$1 → 1.3941 Francs → 0.9308 Euros → \$1.00084.
 - ◆ Is there a negative cost cycle?
 - ◆ Fastest algorithm is valuable!



Negative cycle detection

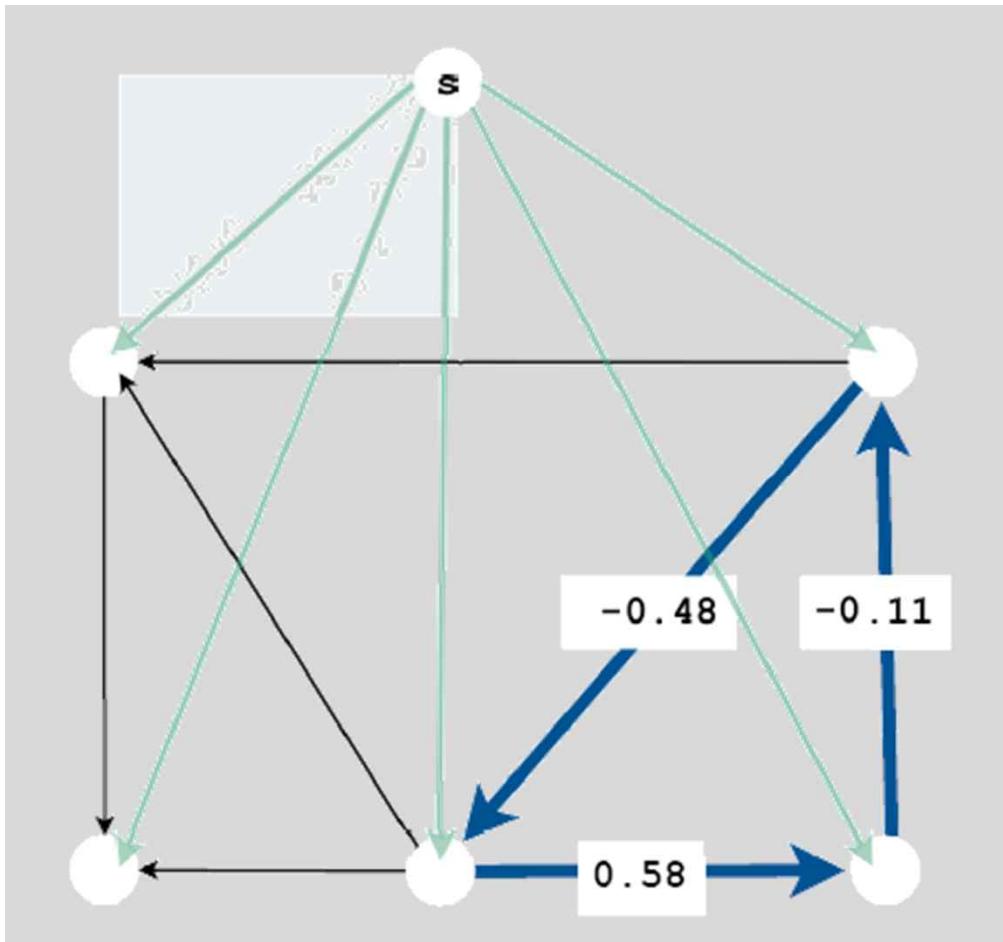
- If there is a negative cycle reachable from s .
 - ◆ Bellman-Ford-Moore gets stuck in loop, updating vertices in cycle.



- Finding a negative cycle.
 - ◆ If any vertex v is updated in phase V , there exists a negative cycle, and we can trace back $\text{pred}[v]$ to find it.

Negative cycle detection

- Goal. Identify a negative cycle (reachable from any vertex).
- Solution. Add **0-weight edge** from artificial source s to each vertex v . Run Bellman-Ford from vertex s .



Shortest paths summary

- Dijkstra's algorithm
 - ◆ easy and optimal for dense digraphs
 - ◆ PQ/ST data type gives near optimal for sparse graphs
- Priority-first search
 - ◆ generalization of Dijkstra's algorithm
 - ◆ encompasses DFS, BFS, and Prim
 - ◆ enables easy solution to many graph-processing problems
- Negative weights
 - ◆ arise in applications
 - ◆ make problem intractable in presence of negative cycles (!)
 - ◆ easy solution using old algorithms otherwise
- Shortest-paths is a broadly useful problem-solving model

Android App Programming

August 4, 2015

Prof. Kyosun Kim
Incheon National University



Outline

1. **Android Studio Installation**
2. **Tutorial I**
3. **Tutorial II**
4. **Bluetooth Chat**
5. **Infra-Red Remote Control V1**
6. **Infra-Red Remote Control V2**

□ **참조: The C Programming Language**

- ◆ <https://www.youtube.com/playlist?list=PLmngpJVWmdCwQueHIid7BSLJj0at7yEx9>

□ **참조: Atmega128 Micro-Controller**

- ◆ https://www.youtube.com/watch?v=kvnc2L5pA78&list=PLmngpJVWmdCyMUuNNDRMIVcfgfYj_YIq9

1. Android Studio Installation

March 28, 2015

Prof. Kyosun Kim
Incheon National University

Outline

1. References for Android Studio Installation
2. Java Compiler Download
3. Java Compiler Installation
4. Environment Variables for Java Compiler
5. Java Setup Test
6. Android Studio Download
7. Android Studio Installation
8. Android Phone Developer's Environment Setting
9. Android Studio Tour
 - ◆ Project Creation
 - ◆ SDK Download
 - ◆ Virtual Devices, Emulation
 - ◆ GUI Builder
 - ◆ Key Commands
10. Mobizen Installation

References for Android Studio Installation

□ Android Development Environment Setup Reference Sites

- ◆ <http://prolite.tistory.com/456> <= How to Install Android Studio
- ◆ <http://www.oracle.com/technetwork/java/javase/downloads/index.html?sourceSiteId=oocomen> <= Java Installation
- ◆ <http://developer.android.com> <= Android Studio Installation

Java Compiler Download

[http://www.oracle.com/technetwork/java/javase/downloads/index.html
?ssSourceSiteId=ocomen](http://www.oracle.com/technetwork/java/javase/downloads/index.html?ssSourceSiteId=ocomen) <= Java Installation Site.

The screenshot shows the Oracle Java SE Downloads page. Step 1 highlights the 'Java Platform (JDK) 8u25' download button with a red box and the word 'Click'. Step 2 highlights the 'Accept License Agreement' radio button with a red box and the number '2'. A large blue arrow points from the JDK download section to the license agreement section. A red box also highlights the list of download links for various operating systems.

Java SE Downloads

1

Java Platform (JDK) 8u25

DOWNLOAD

NetBeans

DOWNLOAD

Java Platform, Standard Edition

Java SE 8u25

This release includes important security fixes. Oracle strongly recommends that all Java SE 8 users upgrade to this release.

Learn more

2

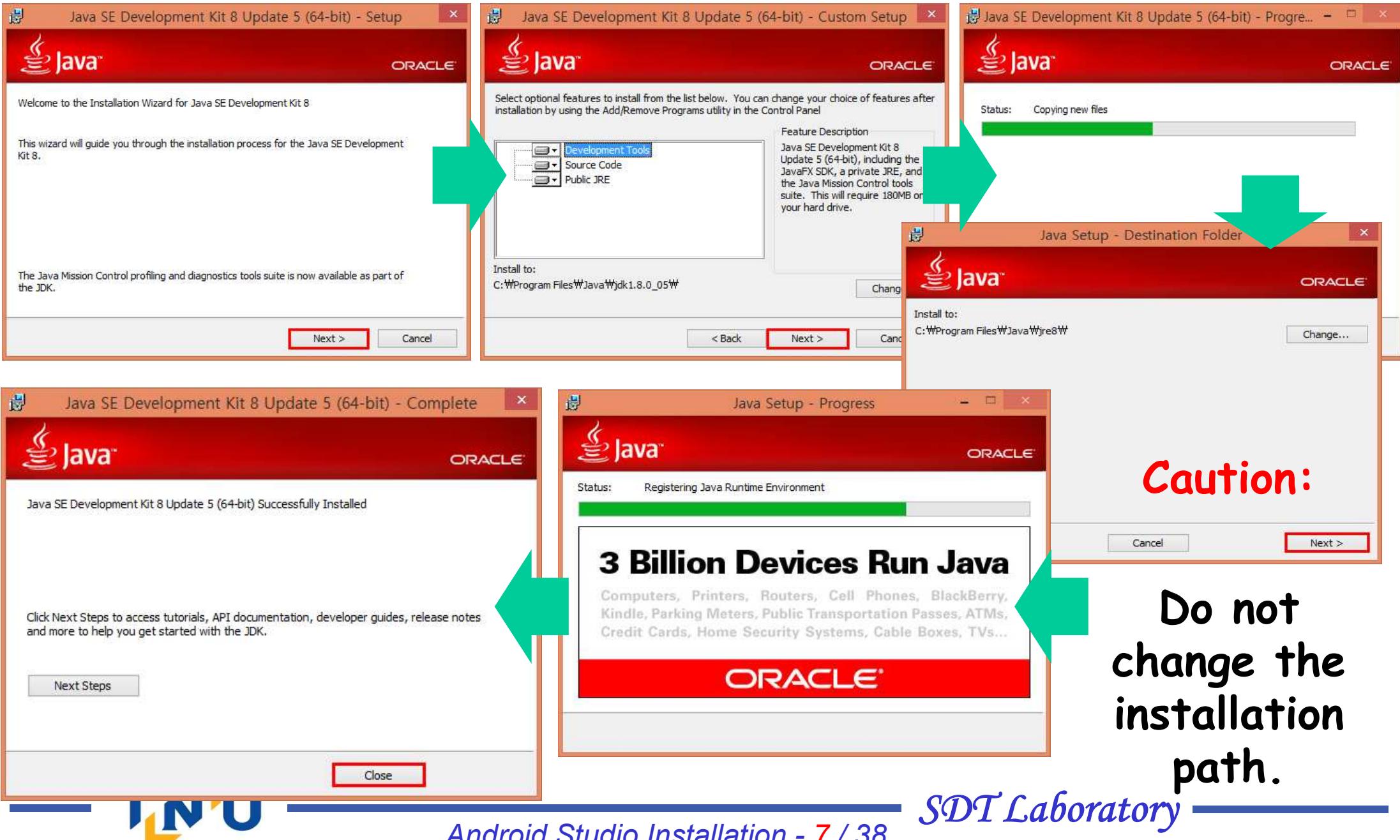
Accept License Agreement Decline License Agreement

Product / File Description File Size Download

Linux x86	135.24 MB	jdk-8u25-linux-i586.rpm
Linux x86	154.88 MB	jdk-8u25-linux-i586.tar.gz
Linux x64	135.6 MB	jdk-8u25-linux-x64.rpm
Linux x64	153.42 MB	jdk-8u25-linux-x64.tar.gz
Mac OS X x64	209.13 MB	jdk-8u25-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	137.01 MB	jdk-8u25-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	97.14 MB	jdk-8u25-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	137.11 MB	jdk-8u25-solaris-x64.tar.Z
Solaris x64	94.24 MB	jdk-8u25-solaris-x64.tar.gz
Windows x86	157.26 MB	jdk-8u25-windows-i586.exe
Windows x64	169.62 MB	jdk-8u25-windows-x64.exe

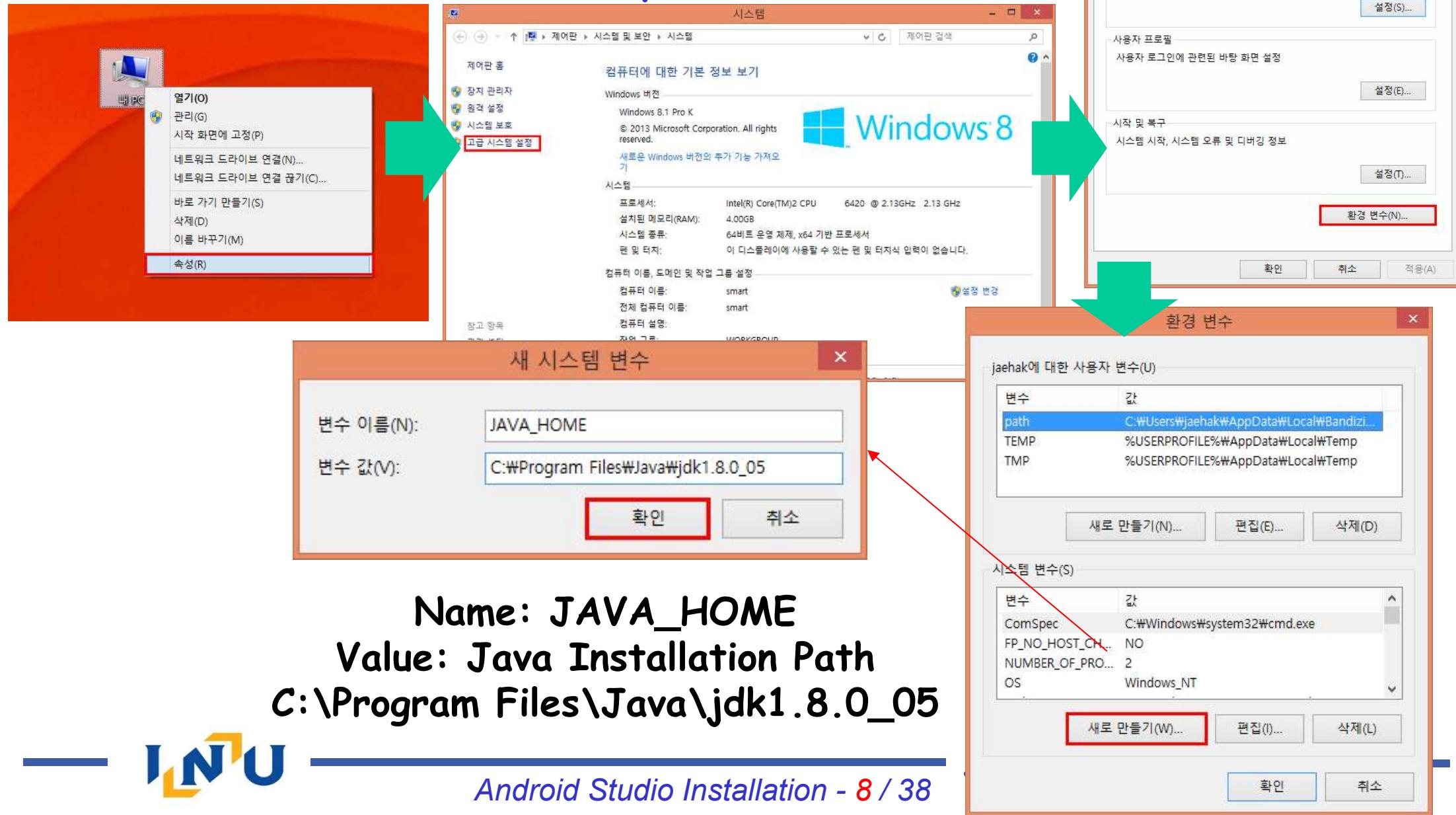
Download a version suitable for your PC.

Java Compiler Installation



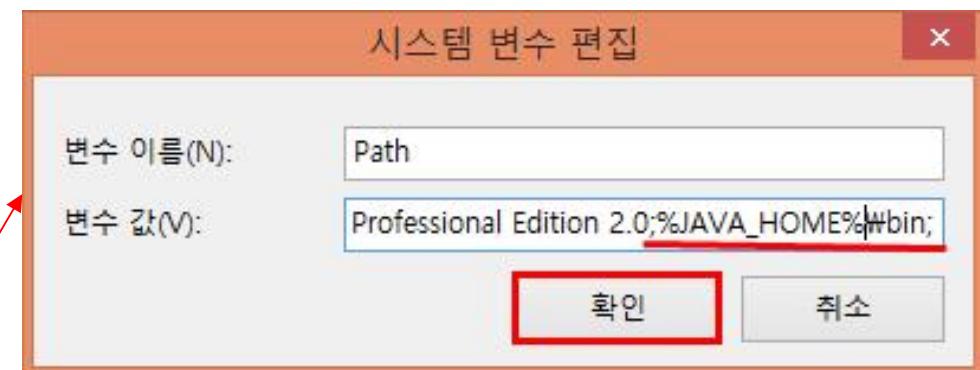
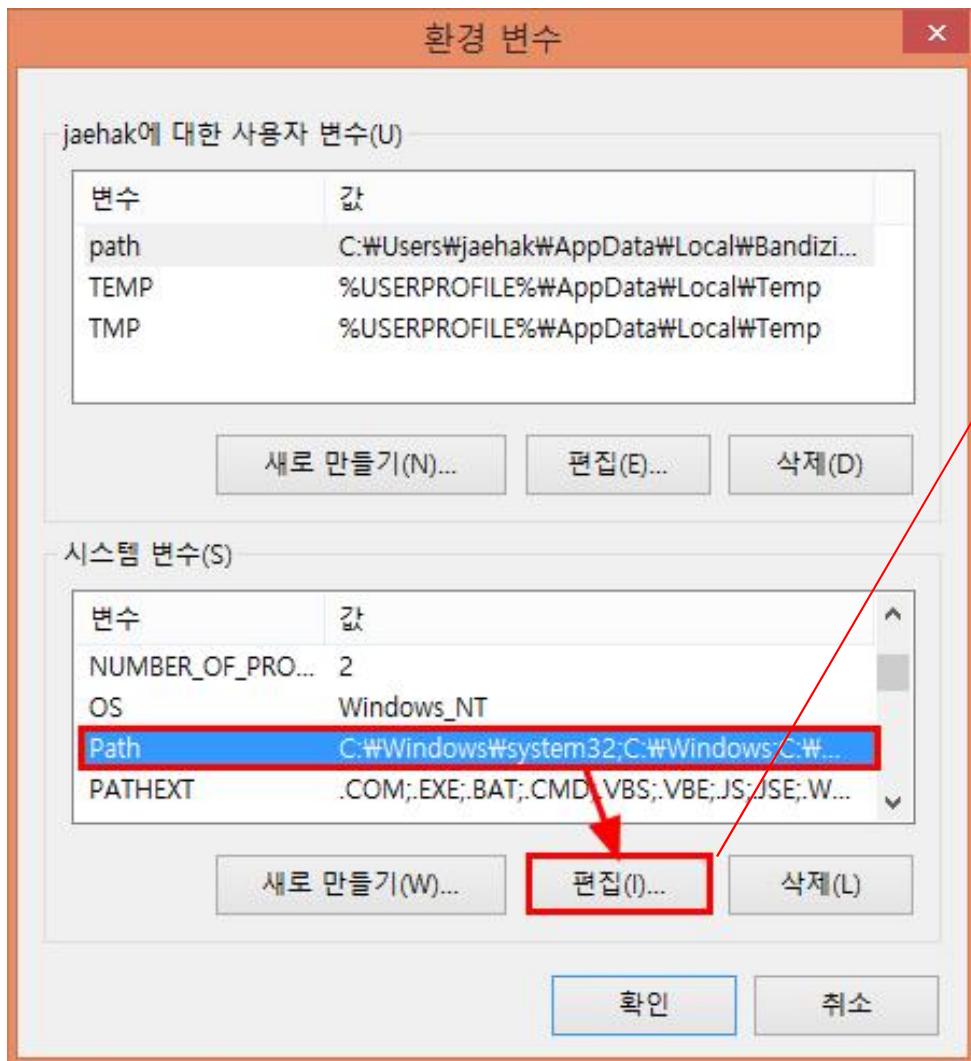
Environment Variables for Java Compiler (1 / 2)

□ Environment Variable Setup



Environment Variables for Java Compiler (2/3)

□ Environment Variable Setup



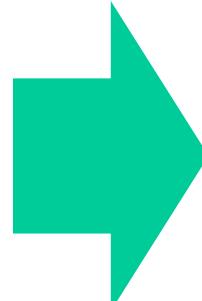
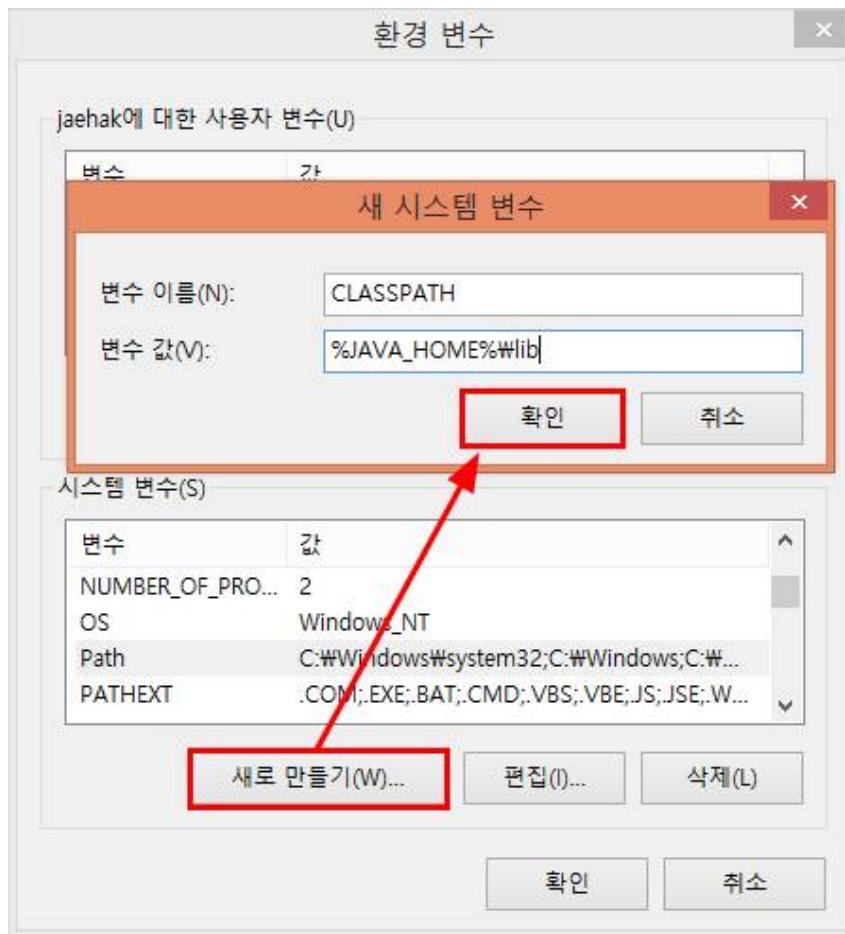
Value: ;%JAVA_HOME%\bin;
Make sure ; at both ends.

Environment Variables for Java Compiler (3/3)

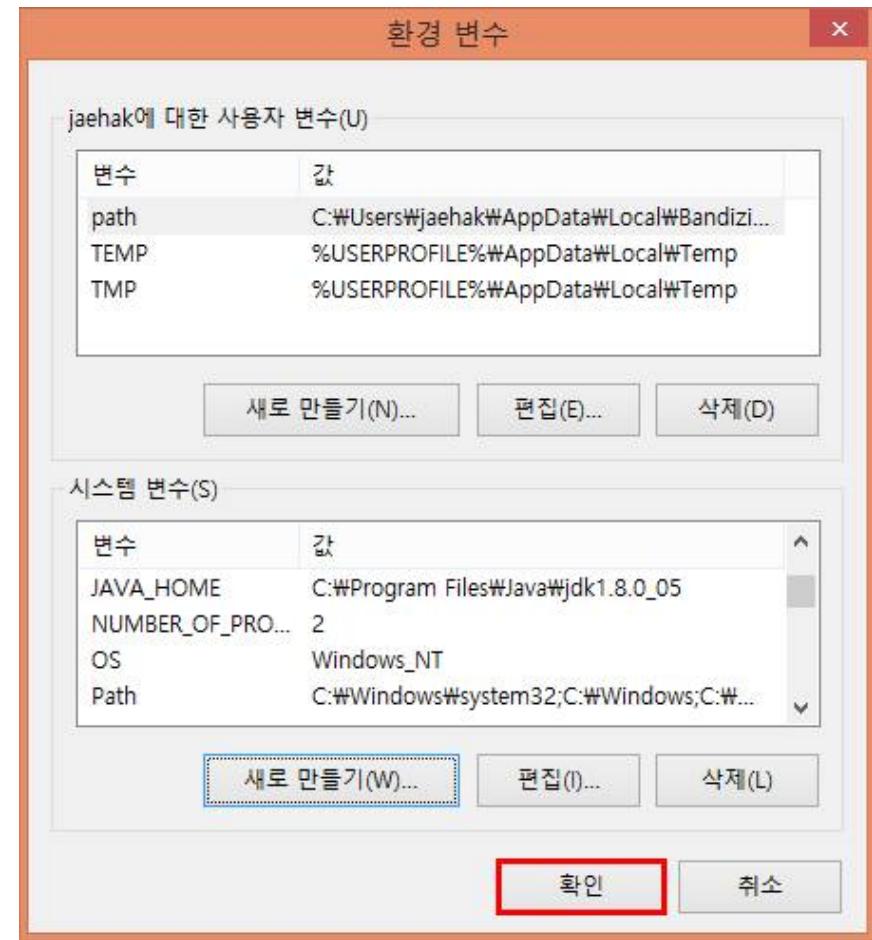
□ 환경변수 설정

Name: CLASSPATH

Value: %JAVA_HOME%\lib



최종적으로 확인!
확인 안 누르면 변경 값 저장 안됨!



Java Setup Test

□ cmd (명령 프롬프트)를 열어 환경변수 설치 확인

The image shows three separate command-line windows from a Windows operating system. Each window has a title bar '관리자: 명령 프롬프트'.

- Left Window (java):** Displays the usage information for the 'java' command. It includes options like '-d32', '-d64', '-server', and various classpath (-cp, -classpath) and verbose (-verbose) options.
- Middle Window (javac):** Displays the usage information for the 'javac' command. It includes options like '-g', '-cp', and various sourcepath (-sourcepath), bootclasspath (-bootclasspath), and extdirs (-extdirs) options.
- Right Window (java -version):** Displays the output of the 'java -version' command, which shows the Java version (1.8.0_05), the Java SE Runtime Environment (build 1.8.0_05-b13), and the Java HotSpot(TM) 64-Bit Server VM (build 25.5-b02, mixed mode).

위와 같이 안 뜨면 경로설정을 다시 한번 확인해 봄.
그리고 자바가 잘 실행되면 cmd 창은 신경 쓸 필요 없음.

Android Studio Download

□ Download

◆ <http://developer.android.com/sdk/index.html> < 설치 사이트로 접속.

The screenshot shows the 'Download' section of the Android Developers website. On the left, there's a sidebar with various links like 'Installing the SDK', 'Adding SDK Packages', and 'Android Studio'. The main content area has a large image of a laptop displaying the Android Studio interface. A red arrow points from the number '1' to the green 'Download Android Studio for Windows' button. Another red arrow points from the number '2' to the checkbox labeled 'I have read and agree with the above terms and conditions'. A third red arrow points from the number '3' to the blue 'Download Android Studio for Windows' button. The right side of the screen shows the 'Download' page with the heading 'Before installing Android Studio or the standalone SDK tools, you must agree to the following terms and conditions.' It includes sections for '1. Introduction', '2. Accepting this License Agreement', and a summary of the license terms.

1

2

3

클릭

Developers ▾ Design Develop Distribute

Training API Guides Reference Tools Google Services Samples

Download

Installing the SDK

Adding SDK Packages

Android Studio

Workflow

Tools Help

Build System

Support Library

Revisions

NDK

ADK

Eclipse with ADT

1

Download Android Studio for Windows

2

I have read and agree with the above terms and conditions

3

Download Android Studio for Windows

3. Before installing Android Studio or the standalone SDK tools, you must agree to the following terms and conditions.

1.1 The Android Software Development Kit (referred to in this License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of this License Agreement. This License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.

1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: <http://source.android.com/>, as updated from time to time.

1.3 "Google" means Google Inc., a Delaware corporation with principal place of business at 1600 Amphitheatre Parkway, Mountain View, CA 94043, United States.

2. Accepting this License Agreement

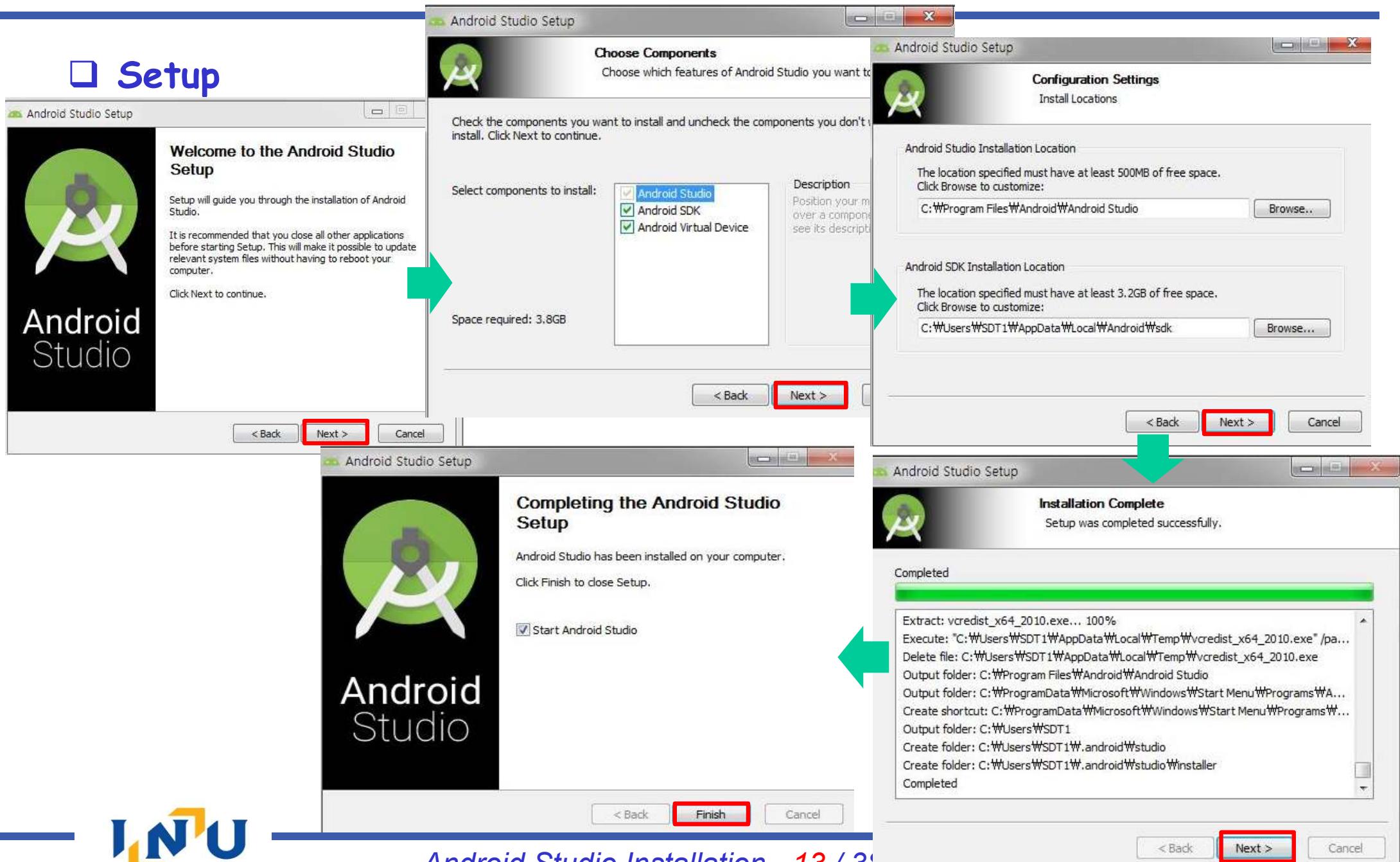
2.1 In order to use the SDK, you must first agree to this License Agreement. You may not use the SDK if you do not accept this License Agreement.

2.2 By clicking to accept, you hereby agree to the terms of this License Agreement.

Download

Android Studio Installation

Setup



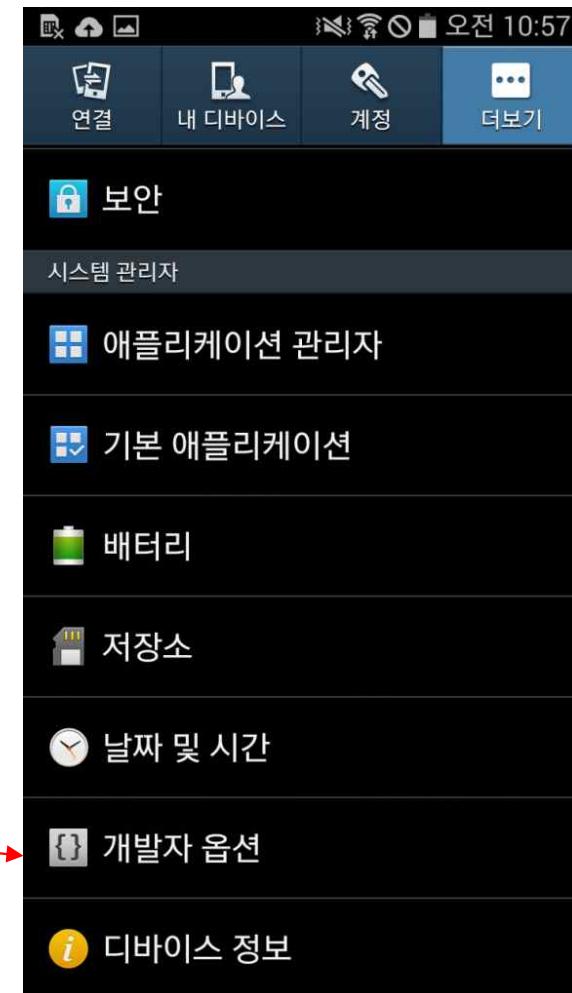
Android Phone Developer's Environment Setting

□ 휴대폰 설정 방법

◆ 환경설정 => 일반/더보기 => 디바이스 정보

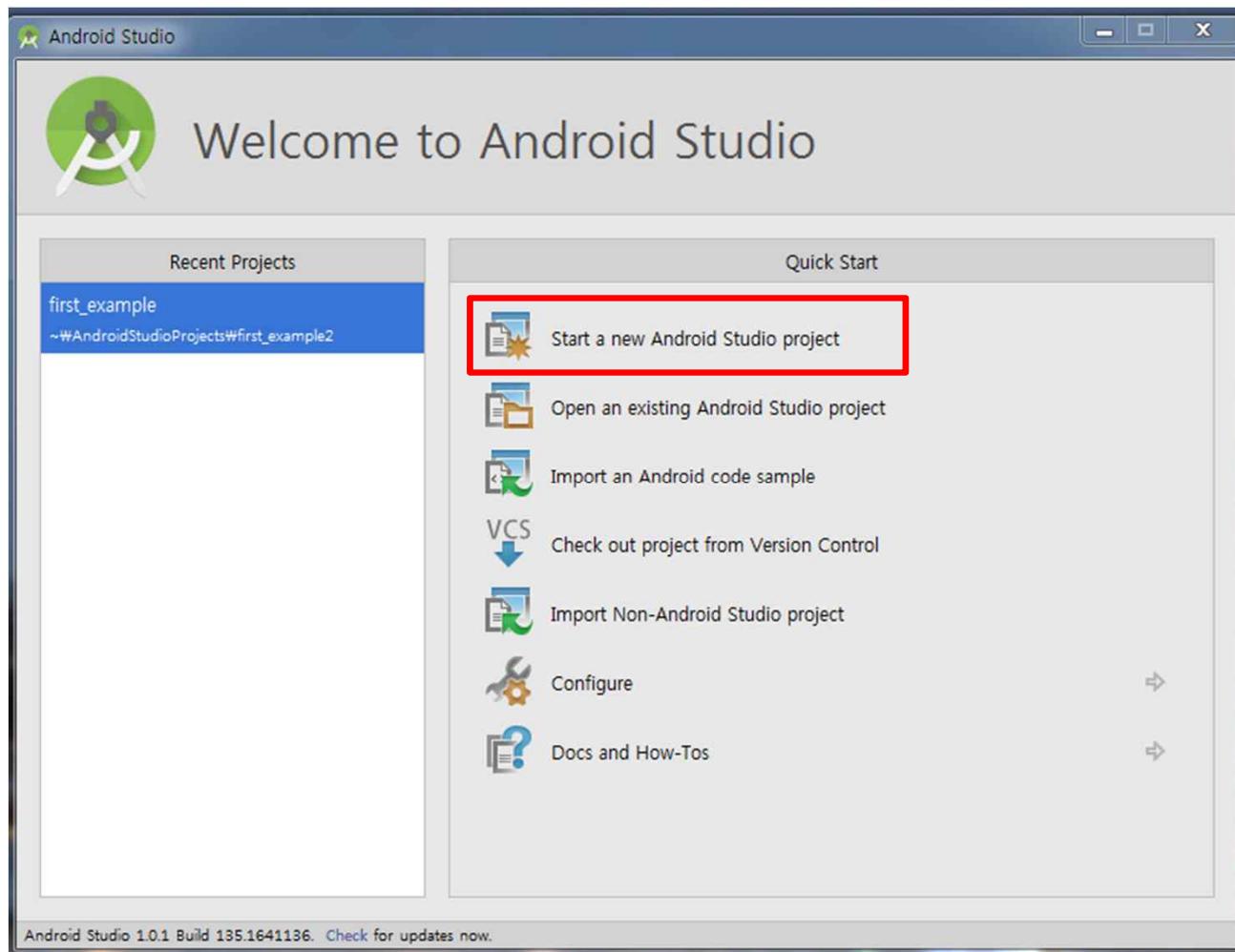


이 부분을 여러
번 누르면
개발자 모드가
실행 됨.



Android Studio Tour: Project Creation (1/5)

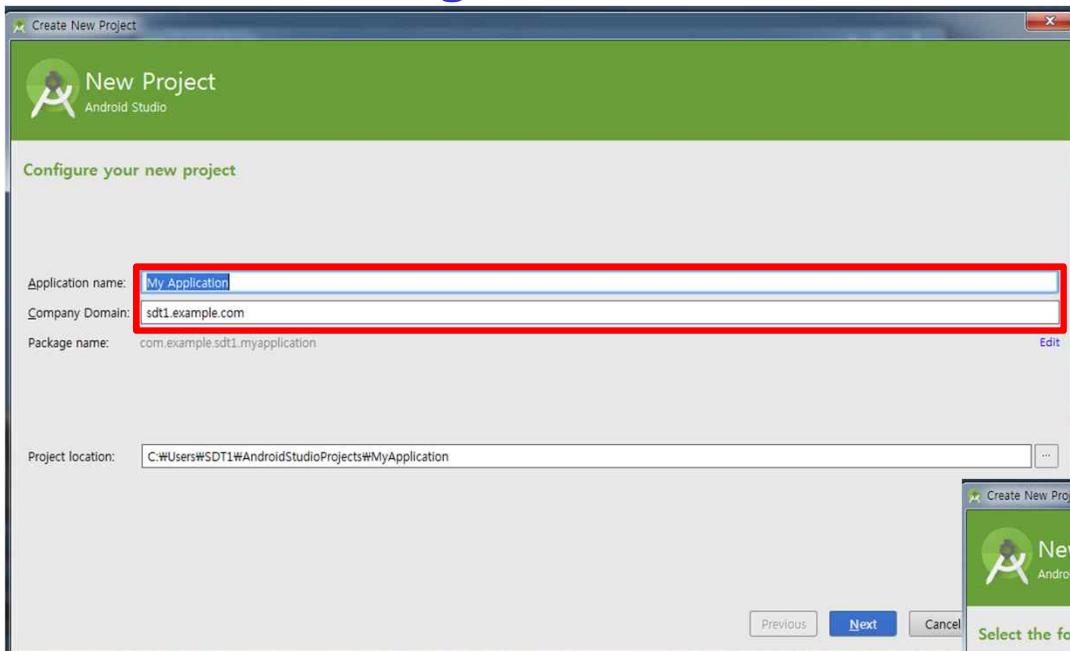
□ Creating a New Project



Project 만들기:
Click Start a new
Android Studio Project

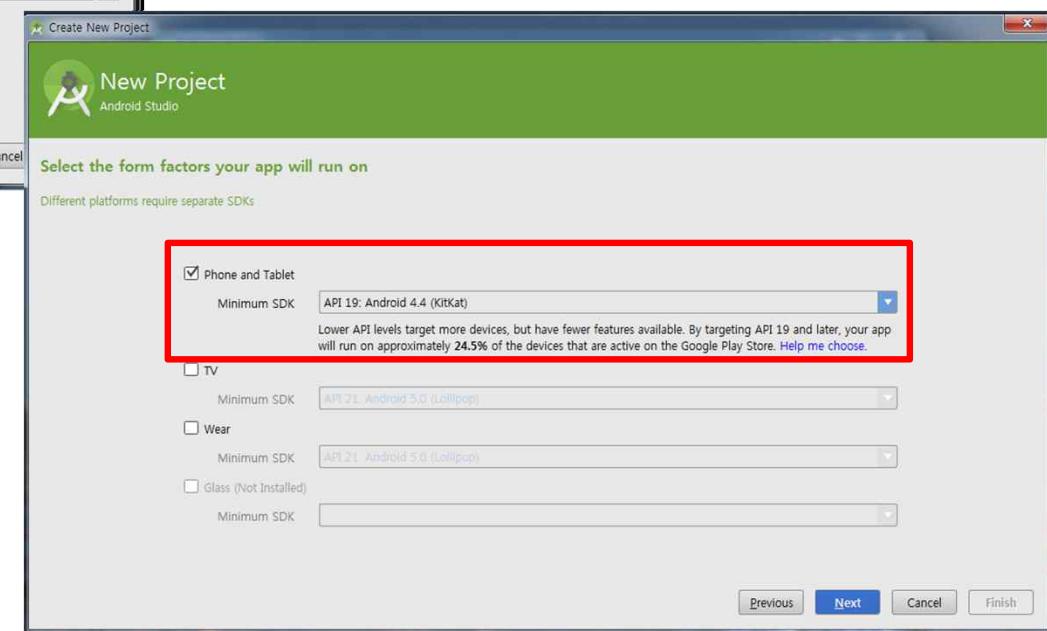
Android Studio Tour: Project Creation (2/5)

□ Android Training



Application 이름 설정

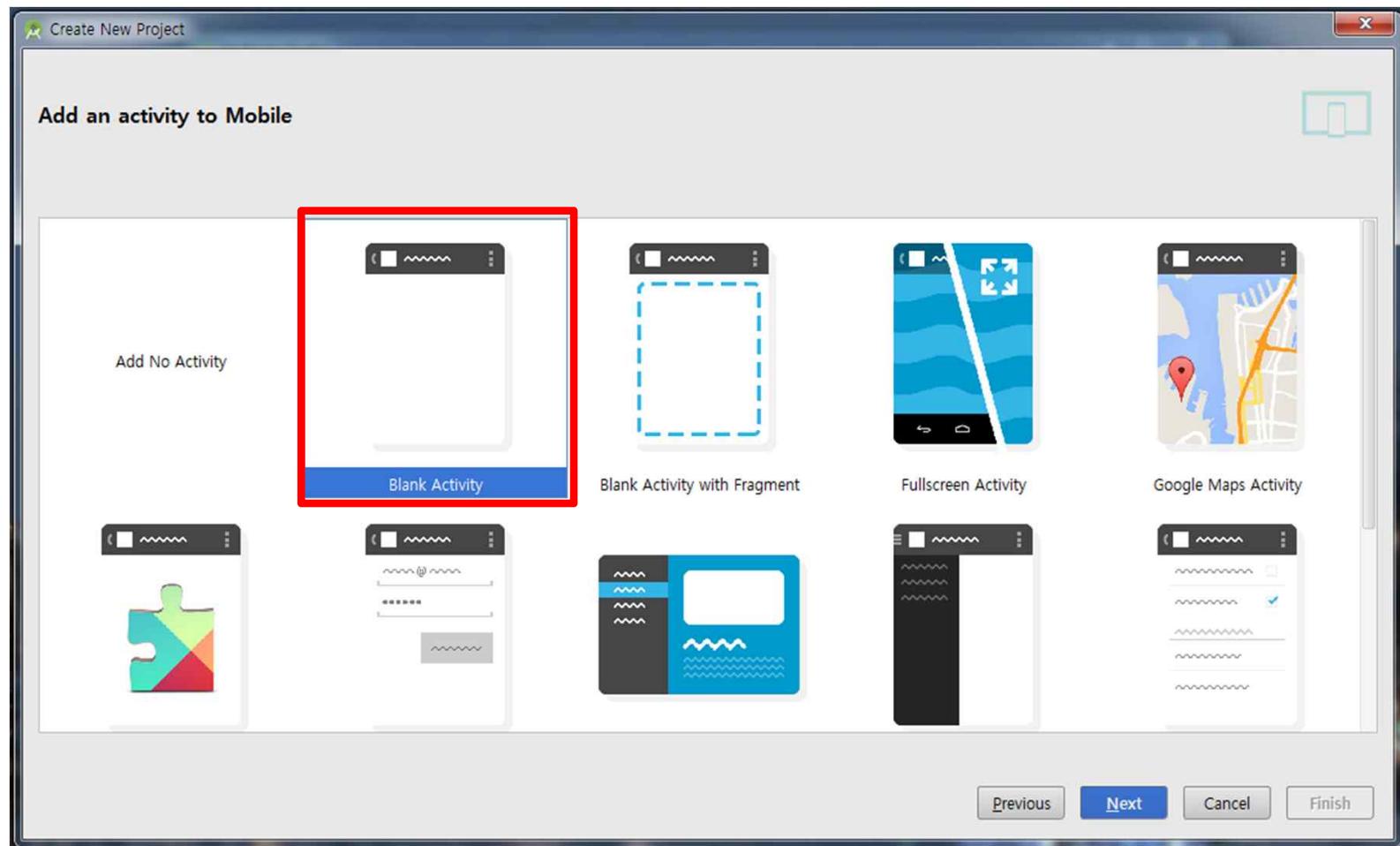
사용할 Android 버전을 설정한다.
2.2 (Froyo)로 한 이유: 가장
가볍기 때문 (시뮬레이션 시 많은
양의 메모리 소모 때문에, 간편한
테스트 시 2.2로 이용)



Android Studio Tour: Project Creation (3/5)

□ Android Training

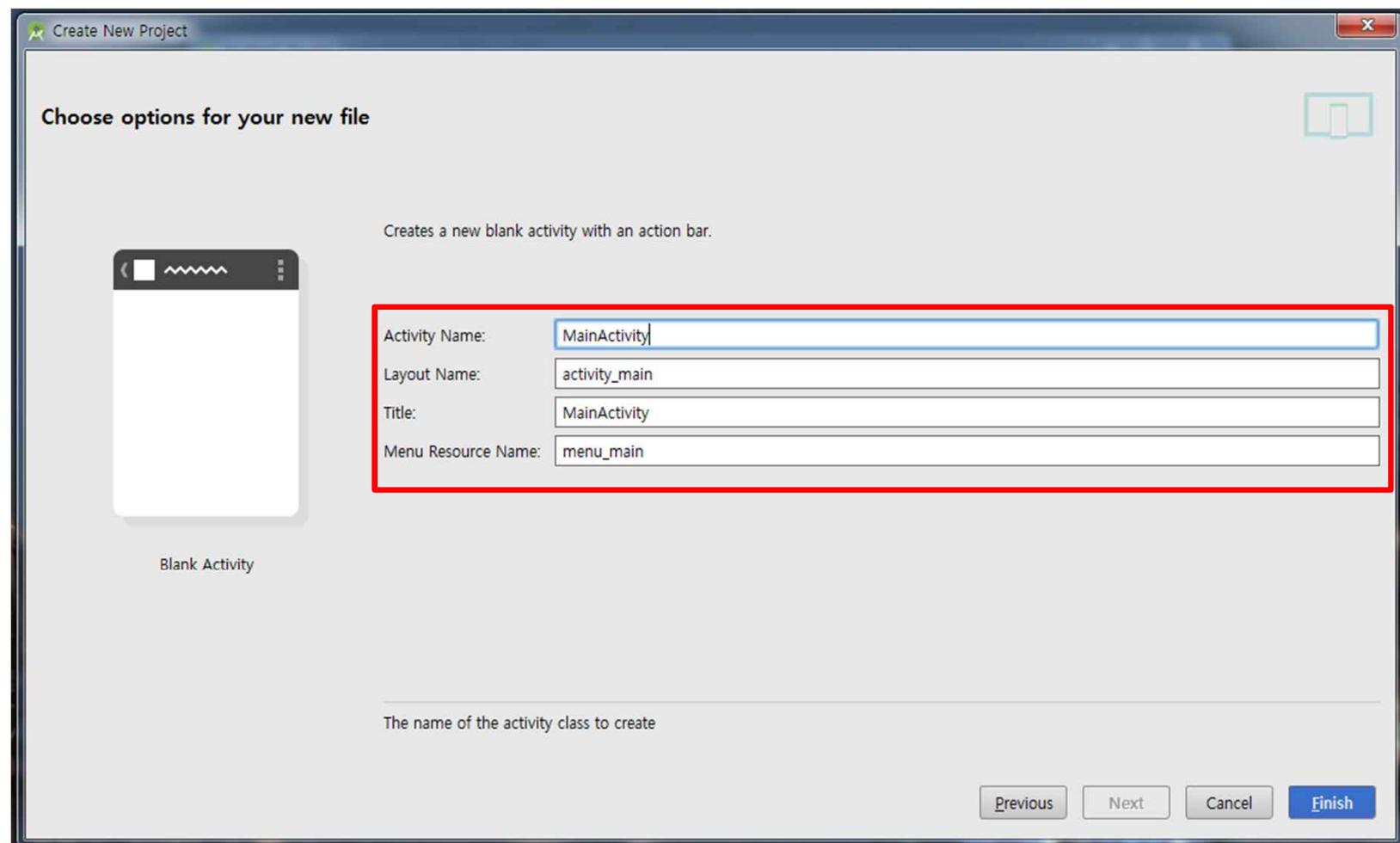
사용할 기본 Activity를 설정 할 수 있다.



Android Studio Tour: Project Creation (4/5)

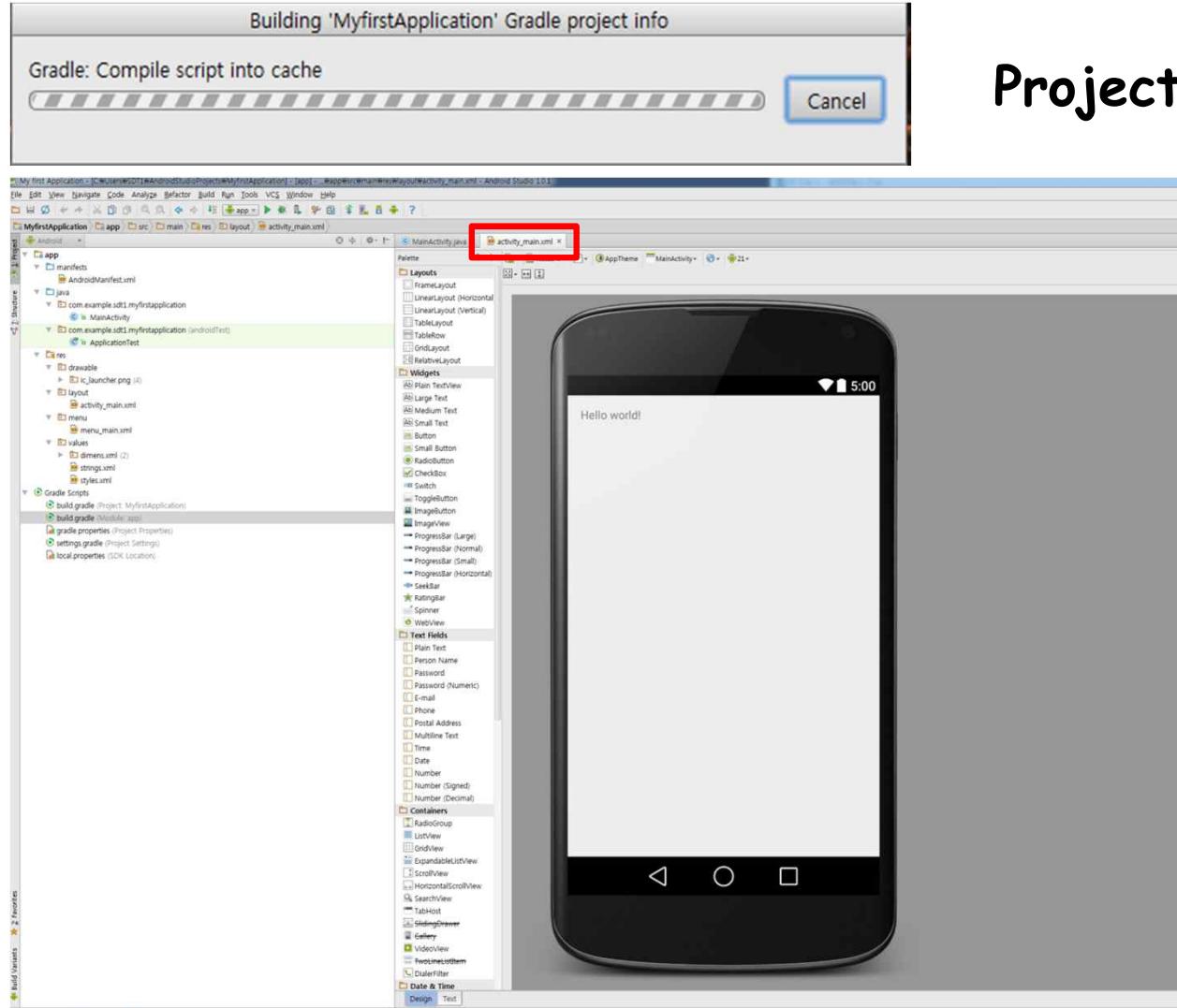
□ Android Training

Activity 이름을 설정한다.



Android Studio Tour: Project Creation (5/5)

□ Android Training

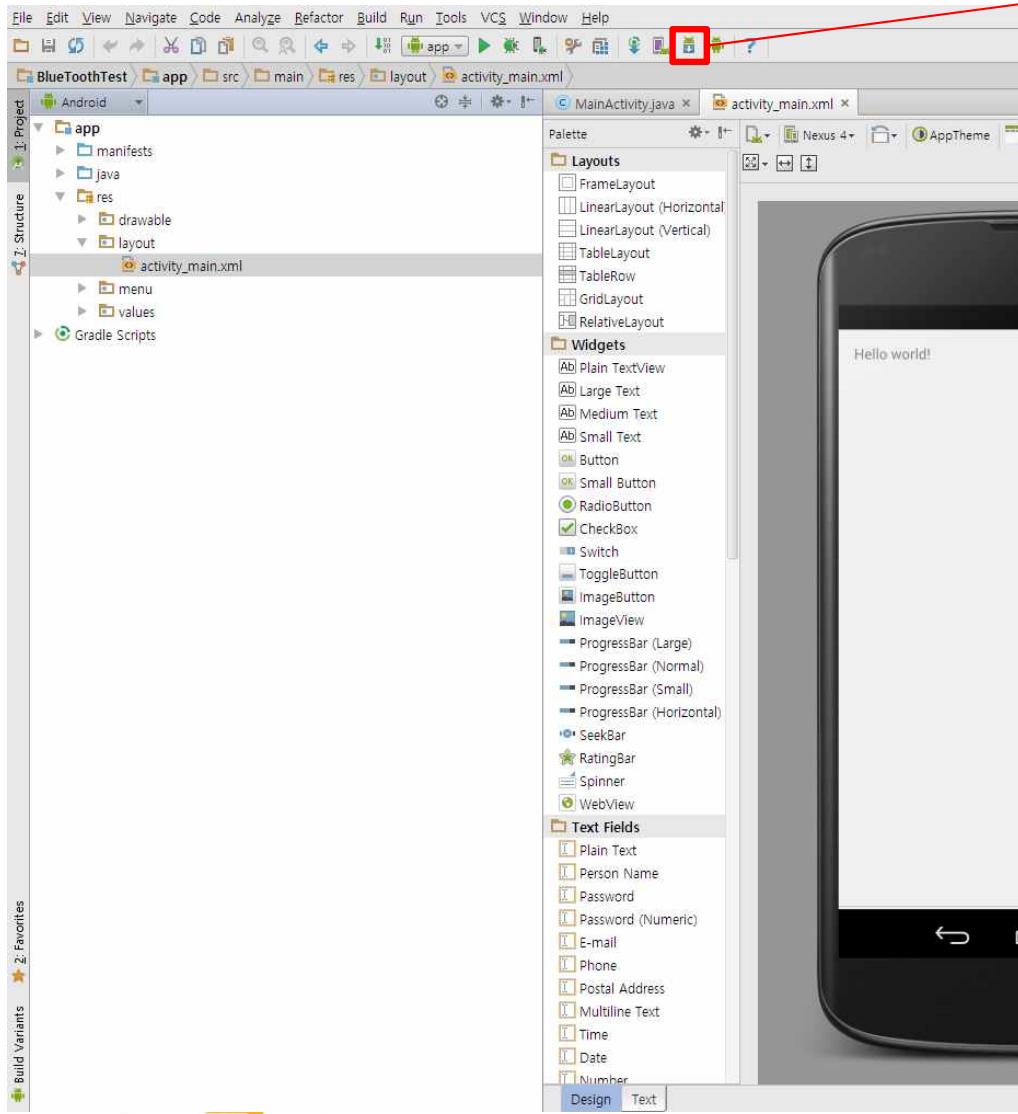


Project 생성중...

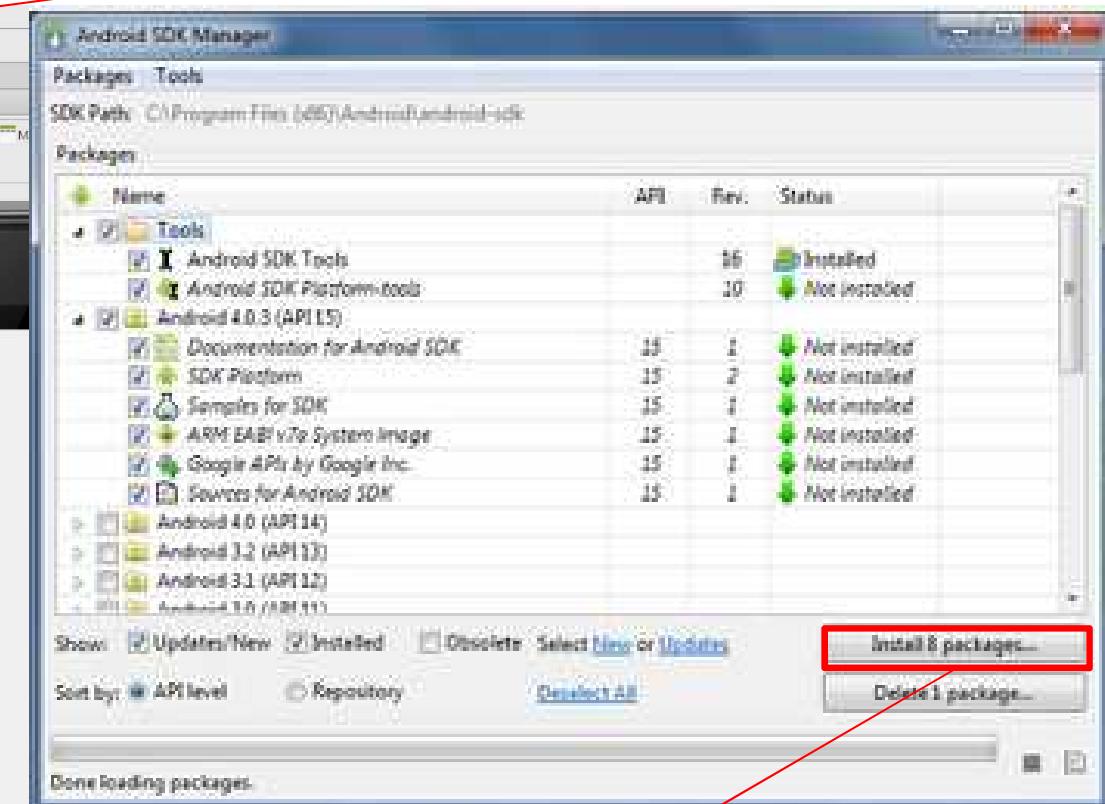
다음과 같은 Project
창이 발생 Simulator를
확인 할 수 있는
activity_프로젝트
명.xml 창이 중앙에
뜨는 것을 확인 할 수
있다.

Android Studio Tour: SDK Download

□ Android Training



SDK Download

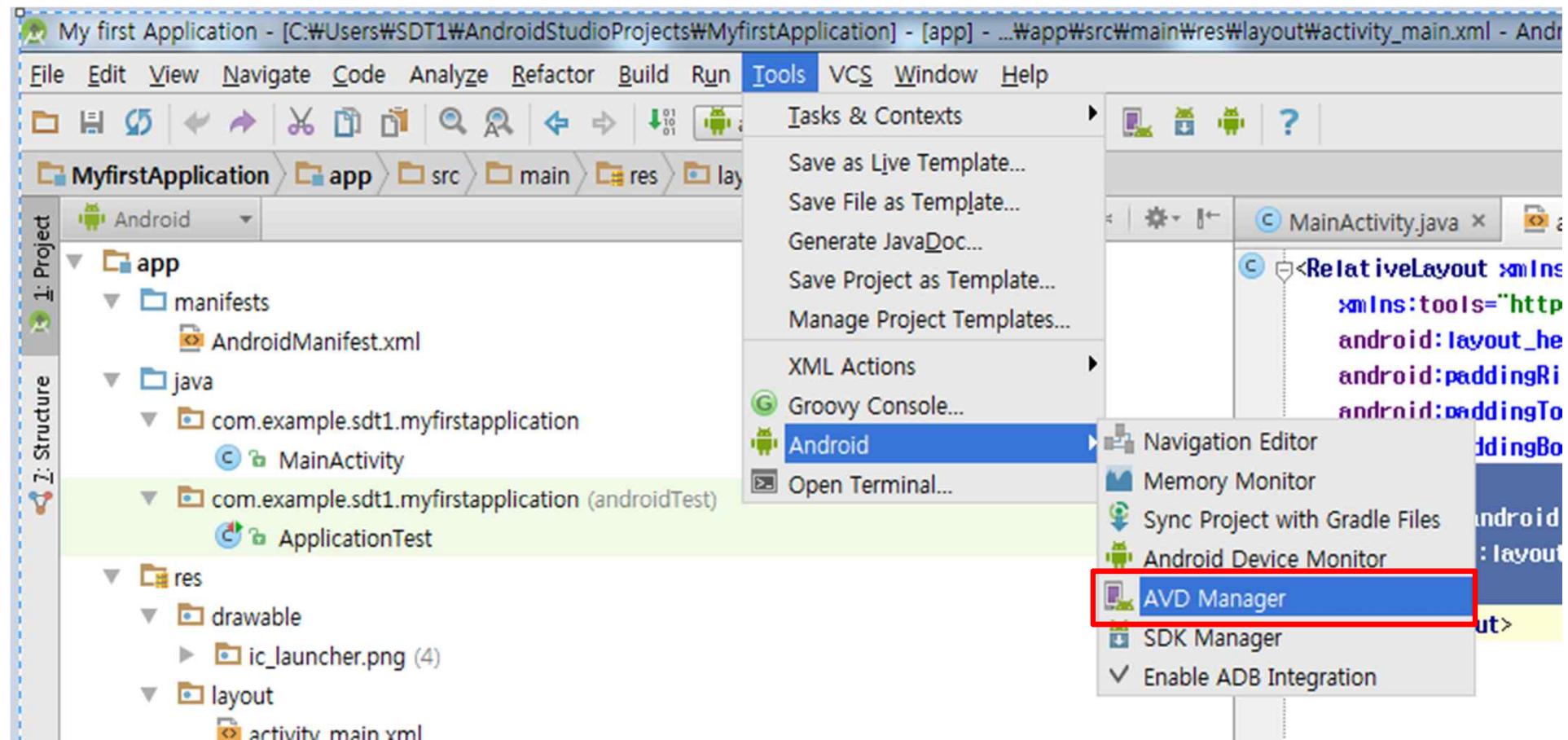


최초로 들어갔을 때
기본적으로 선택된 항목들은
반드시 Install 해야 함.

Android Studio Tour: Virtual Devices (1/3)

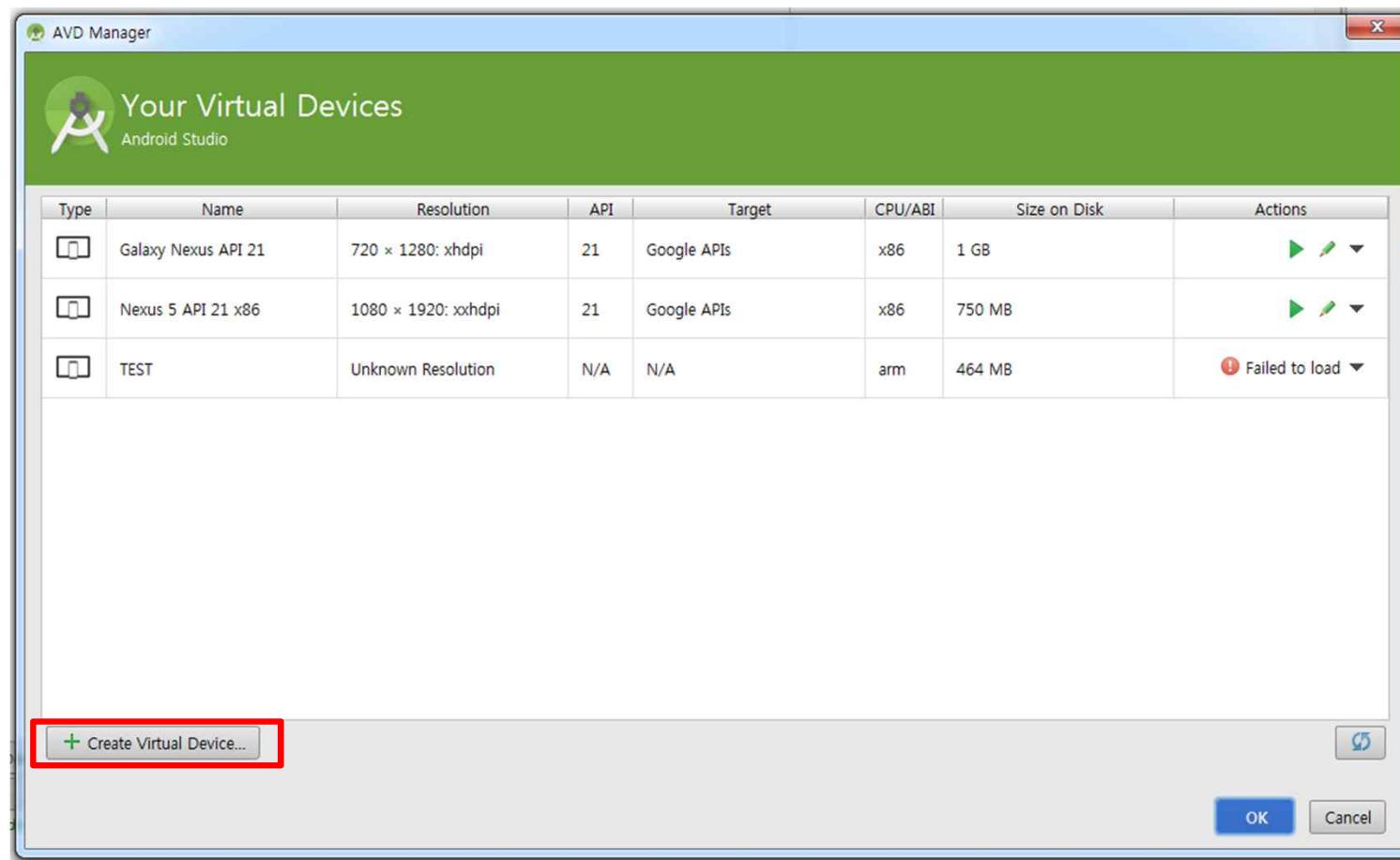
□ Android Training

Simulation을 위한 Emulator (장치
Device 기종)를 추가한다.



Android Studio Tour: Virtual Devices (2/3)

□ Android Training

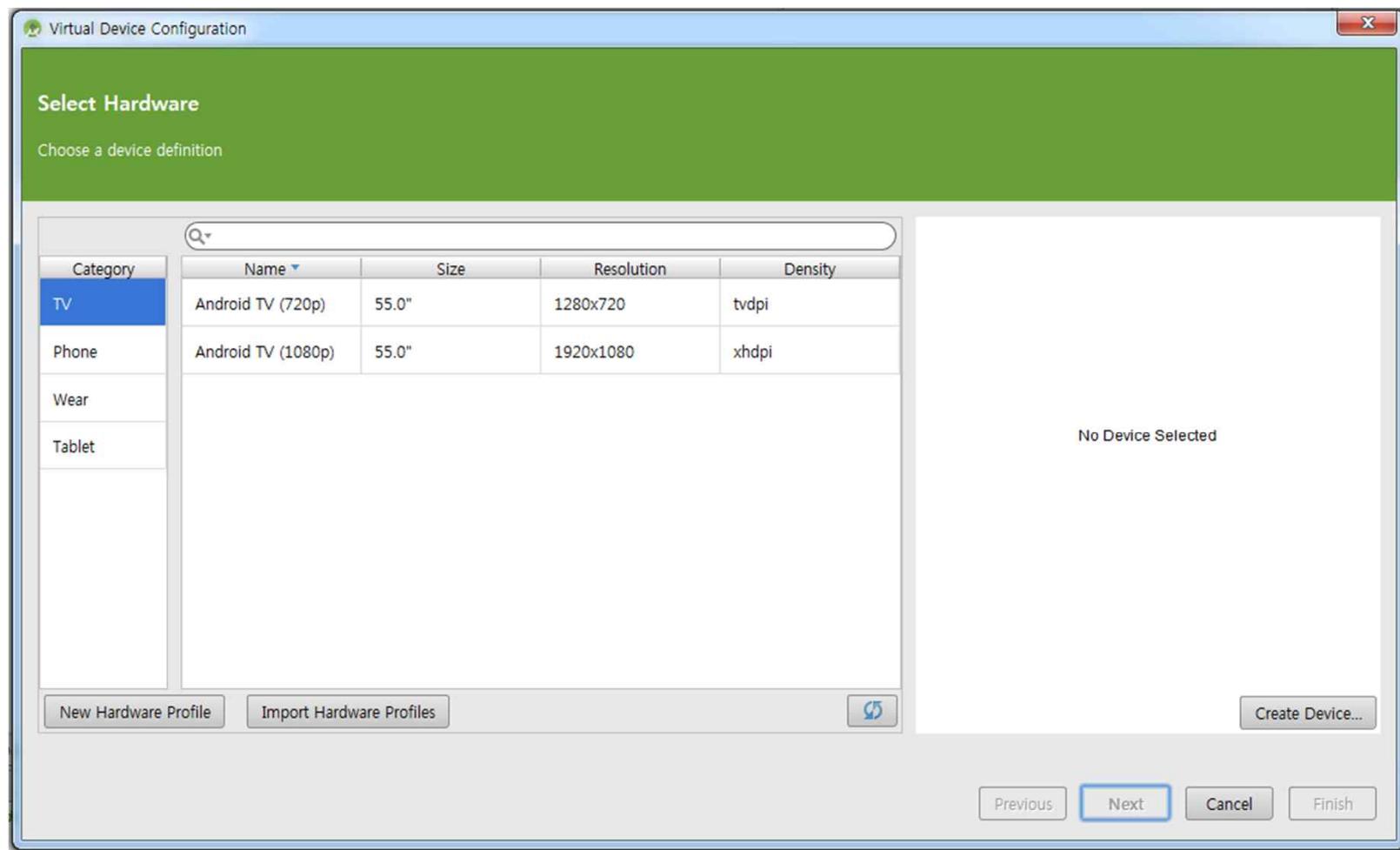


화면에 기존에
추가된 Device가
표시되고 있으며,
Emulator로
사용할 Device를
원하는 기종으로
추가할 수 있다.

Android Studio Tour: Virtual Devices (3/3)

□ Android Training

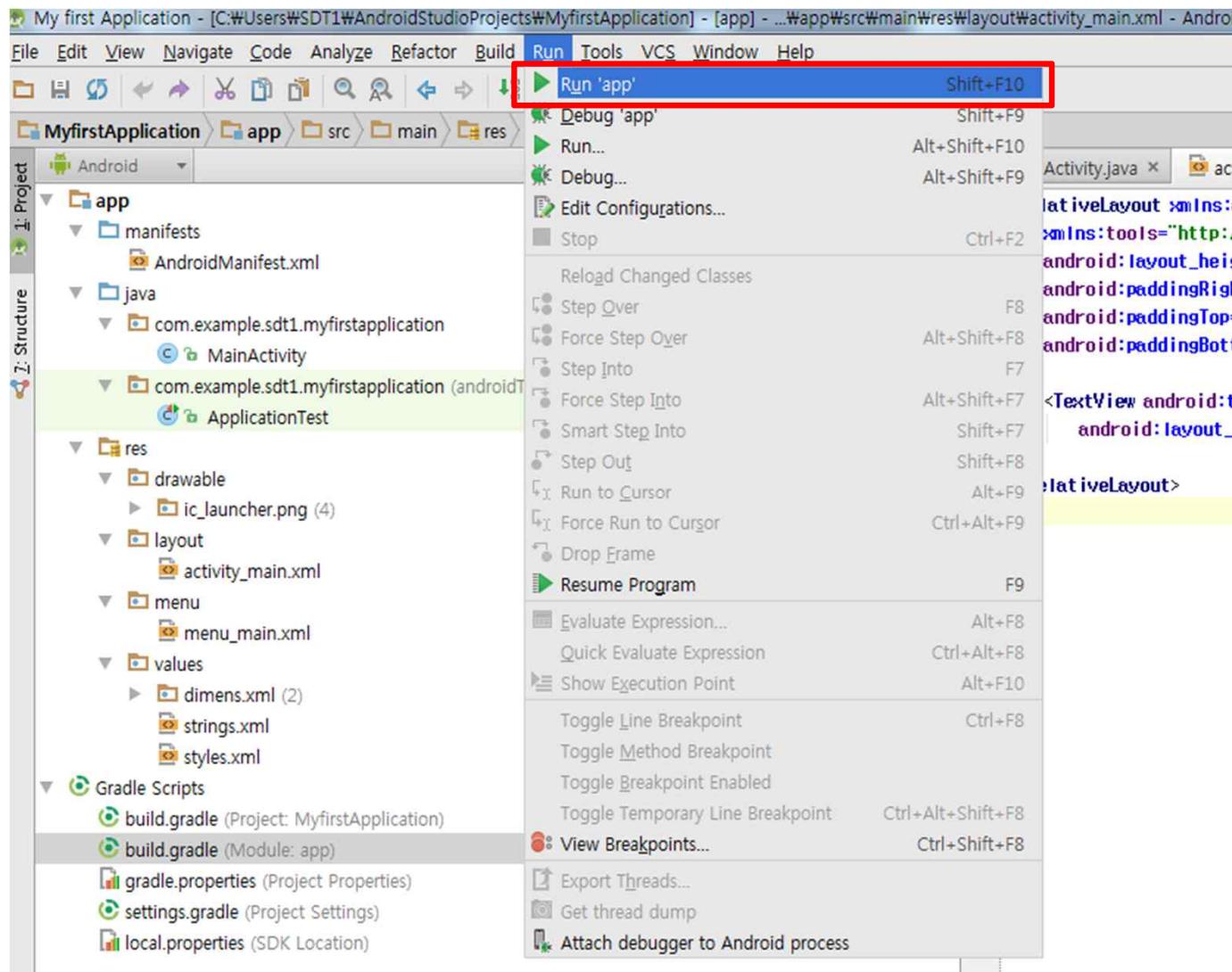
추가될 장비는 TV ~ Wearable Device 등등 가능하다.



Android Studio Tour: Emulation (1/3)

□ Android Training

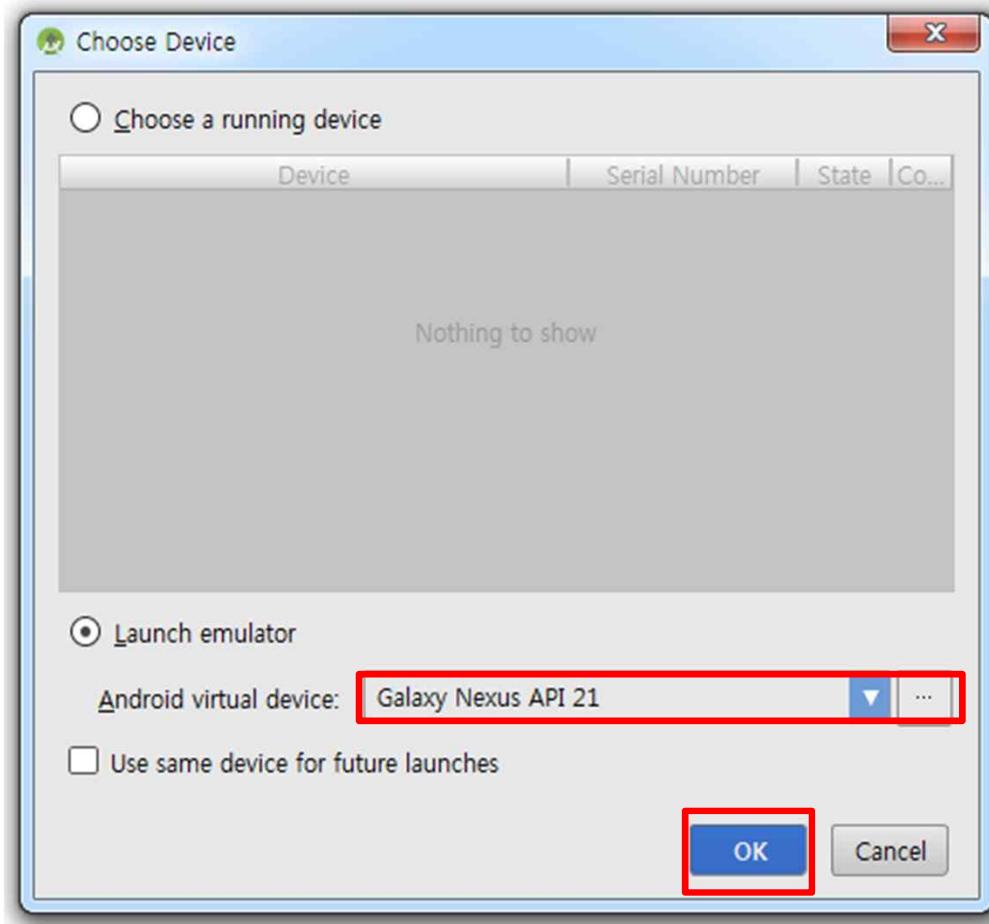
Simulation을 동작시킨다.



Android Studio Tour: Emulation (2/3)

□ Android Training

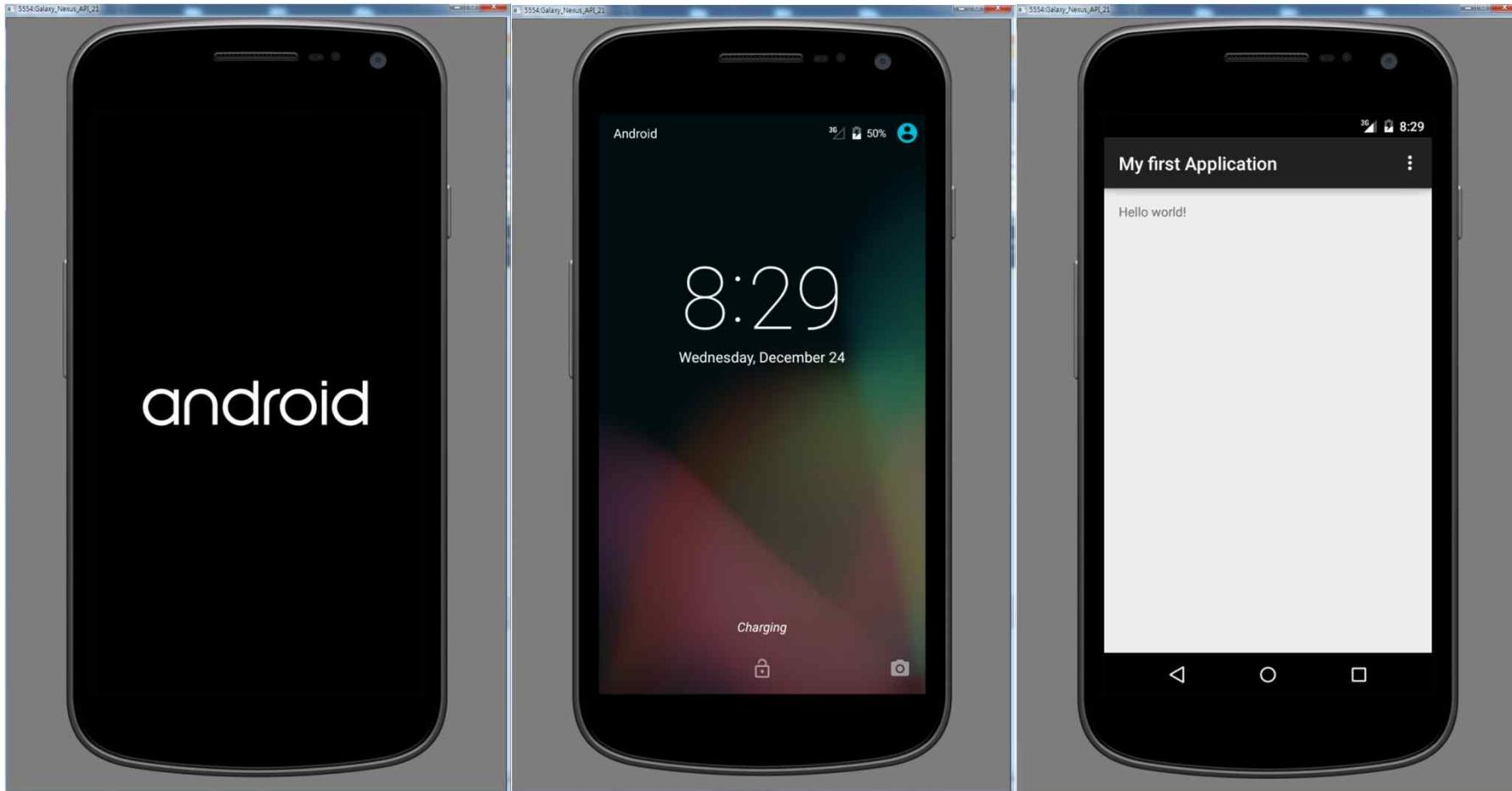
Emulator로 사용할 장비 선택.



Android Studio Tour: Emulation (3/3)

□ Android Training

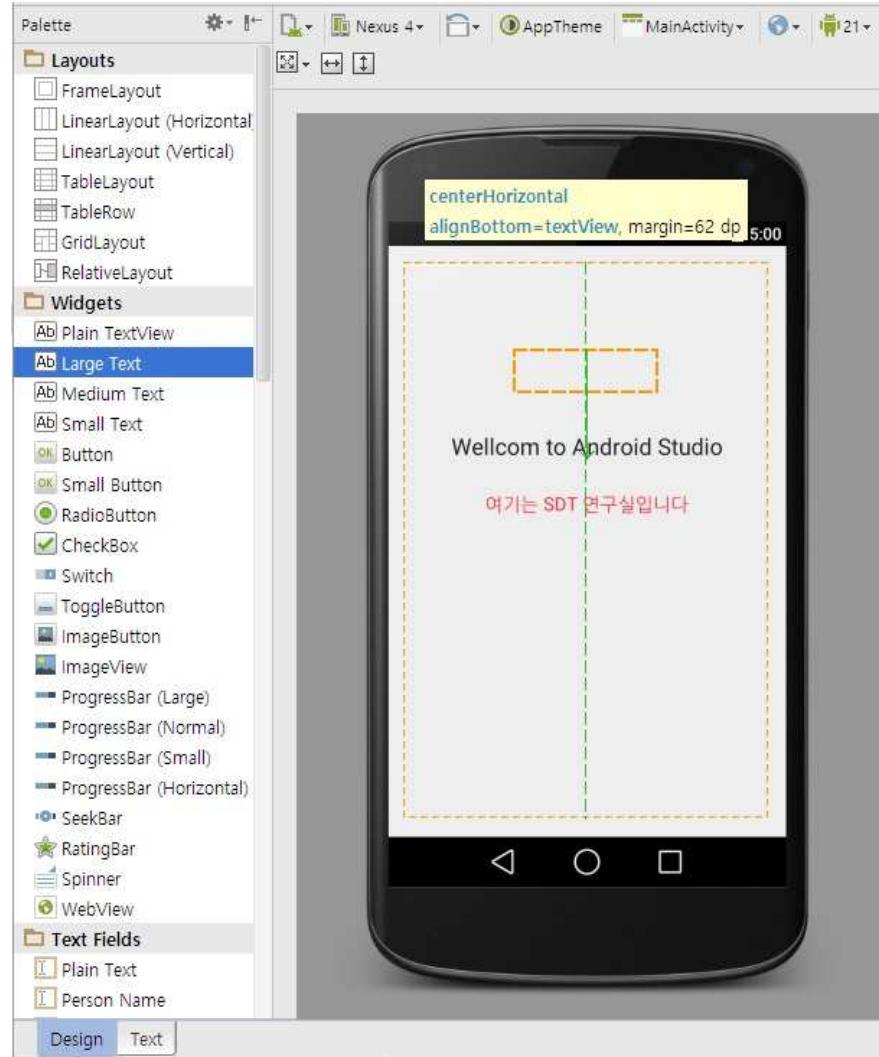
다음과 같은 창이 뜨며 실제 스마트 폰과 같은 동작을 한다.



앞에 정의된 창이 열리는 것을 확인할 수 있다.

Android Studio Tour: GUI Builder (1/4)

□ Android Training

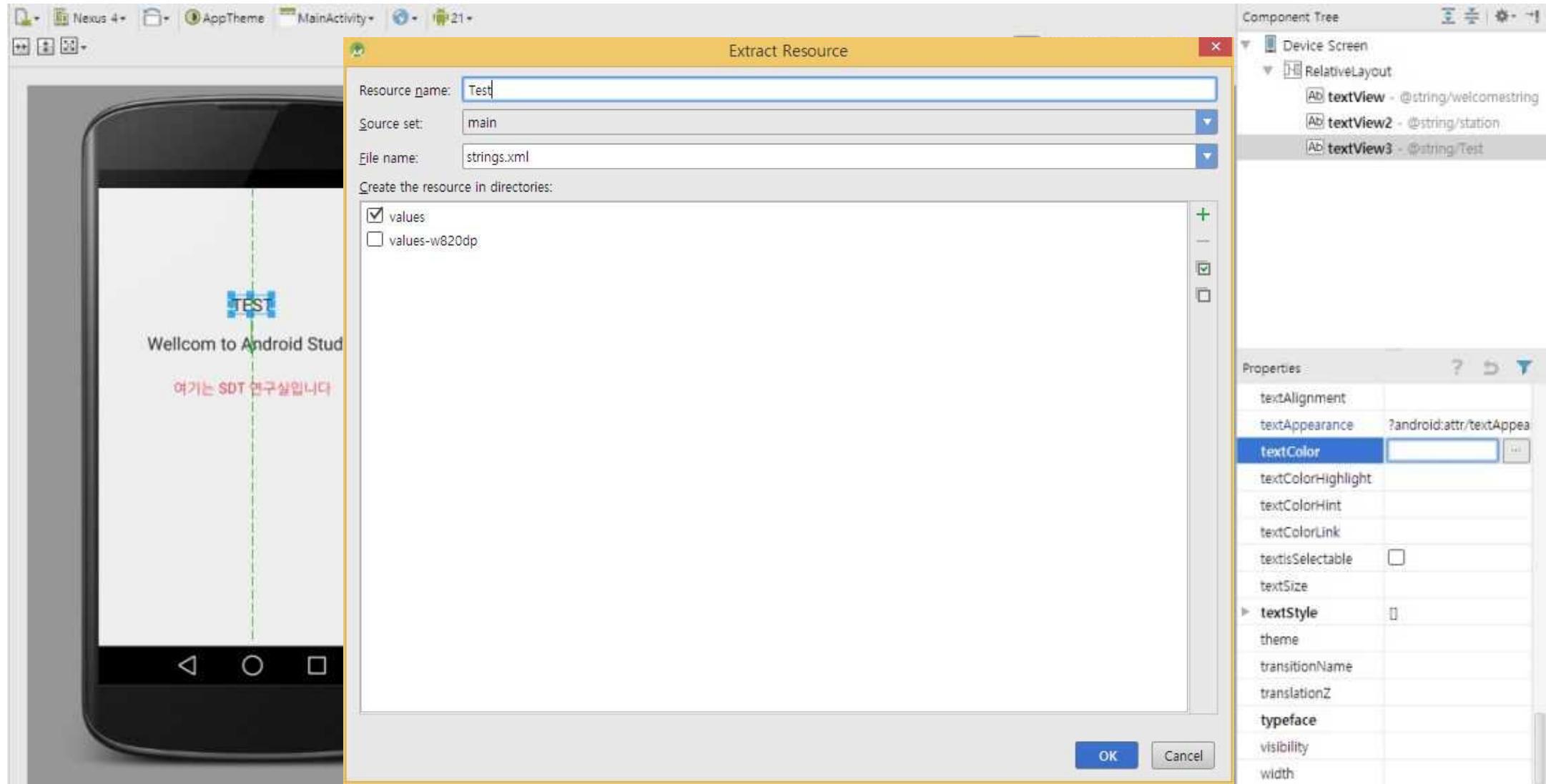


글자 색과 크기 등등 변경하여 표시



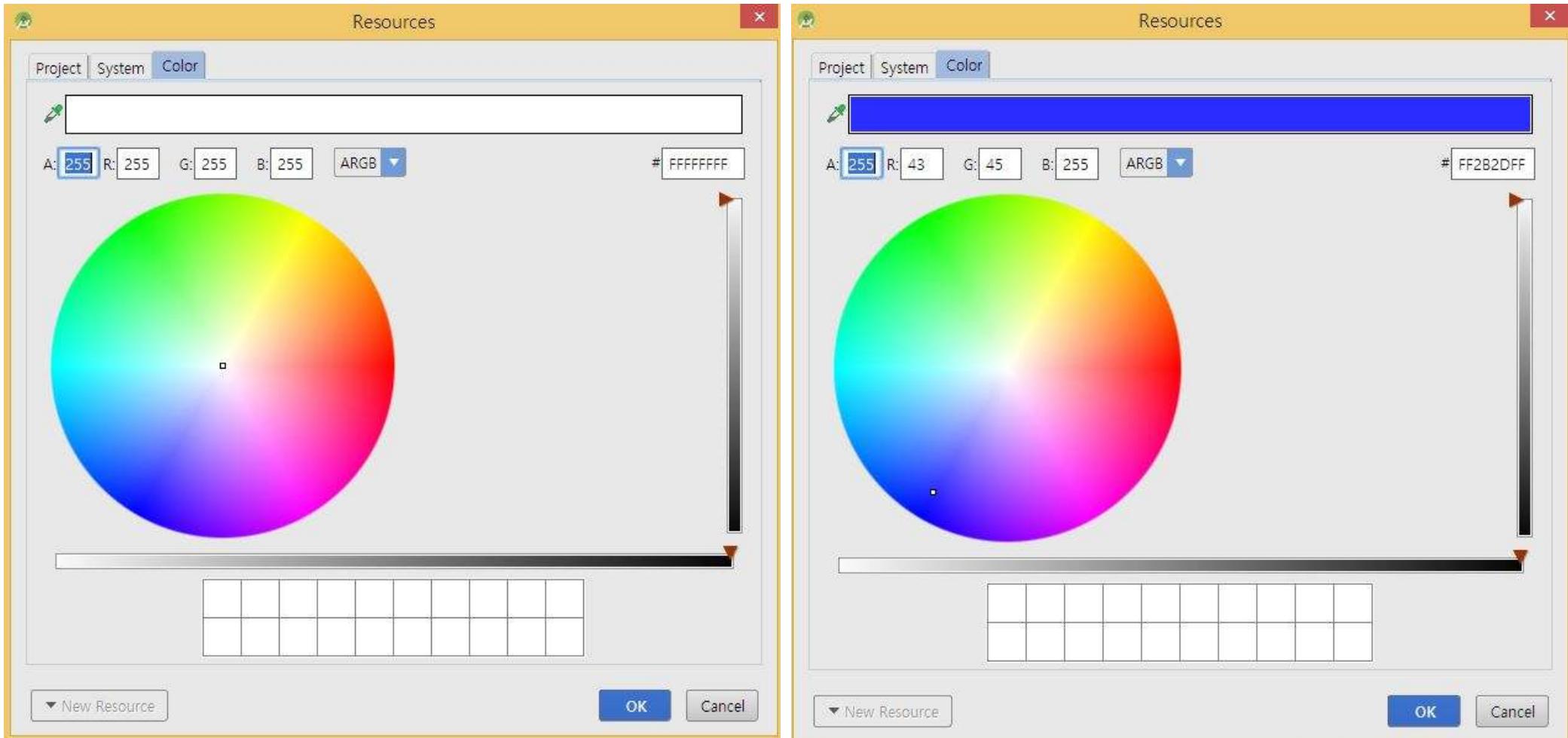
Android Studio Tour: GUI Builder (2/4)

□ Android Training



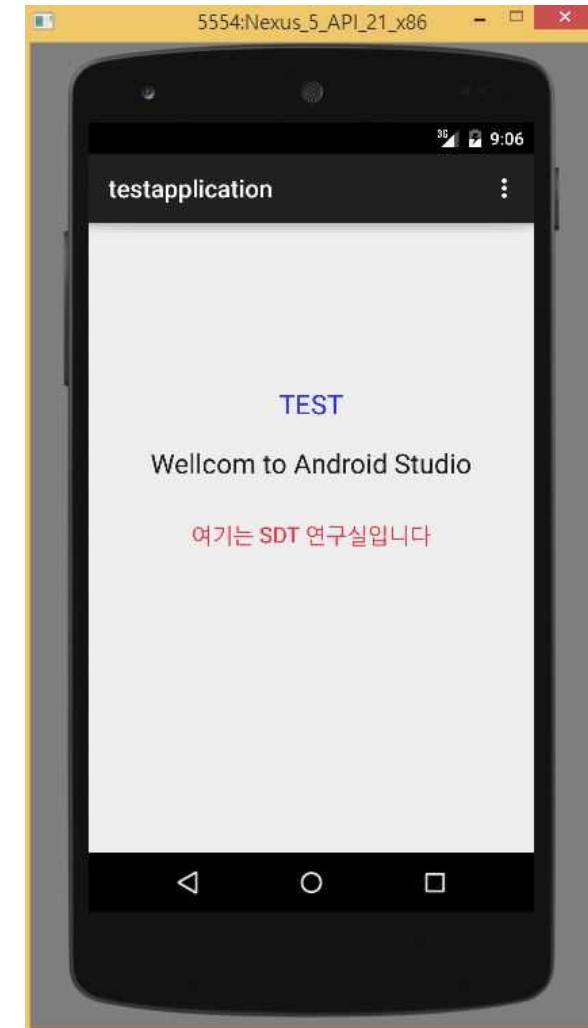
Android Studio Tour: GUI Builder (3/4)

□ Android Training



Android Studio Tour: GUI Builder (4/4)

□ Android Training



Android Studio Tour: Key Commands

Table 1. Programming key commands

Action	Android Studio Key Command
Command look-up (autocomplete command name)	CTRL + SHIFT + A
Project quick fix	ALT + ENTER
Reformat code	CTRL + ALT + L (Win) OPTION + CMD + L (Mac)
Show docs for selected API	CTRL + Q (Win) F1 (Mac)
Show parameters for selected method	CTRL + P
Generate method	ALT + Insert (Win) CMD + N (Mac)
Jump to source	F4 (Win) CMD + down-arrow (Mac)
Delete line	CTRL + Y (Win) CMD + Backspace (Mac)
Search by symbol name	CTRL + ALT + SHIFT + N (Win) OPTION + CMD + O (Mac)

Android Studio Tour: Key Commands (Cont'd)

Table 2. Project and editor key commands

Action	Android Studio Key Command
Build	CTRL + F9 (Win) CMD + F9 (Mac)
Build and run	SHIFT + F10 (Win) CTRL + R (Mac)
Toggle project visibility	ALT + 1 (Win) CMD + 1 (Mac)
Navigate open tabs	ALT + left-arrow; ALT + right-arrow (Win) CTRL + left-arrow; CTRL + right-arrow (Mac)

참조 사이트

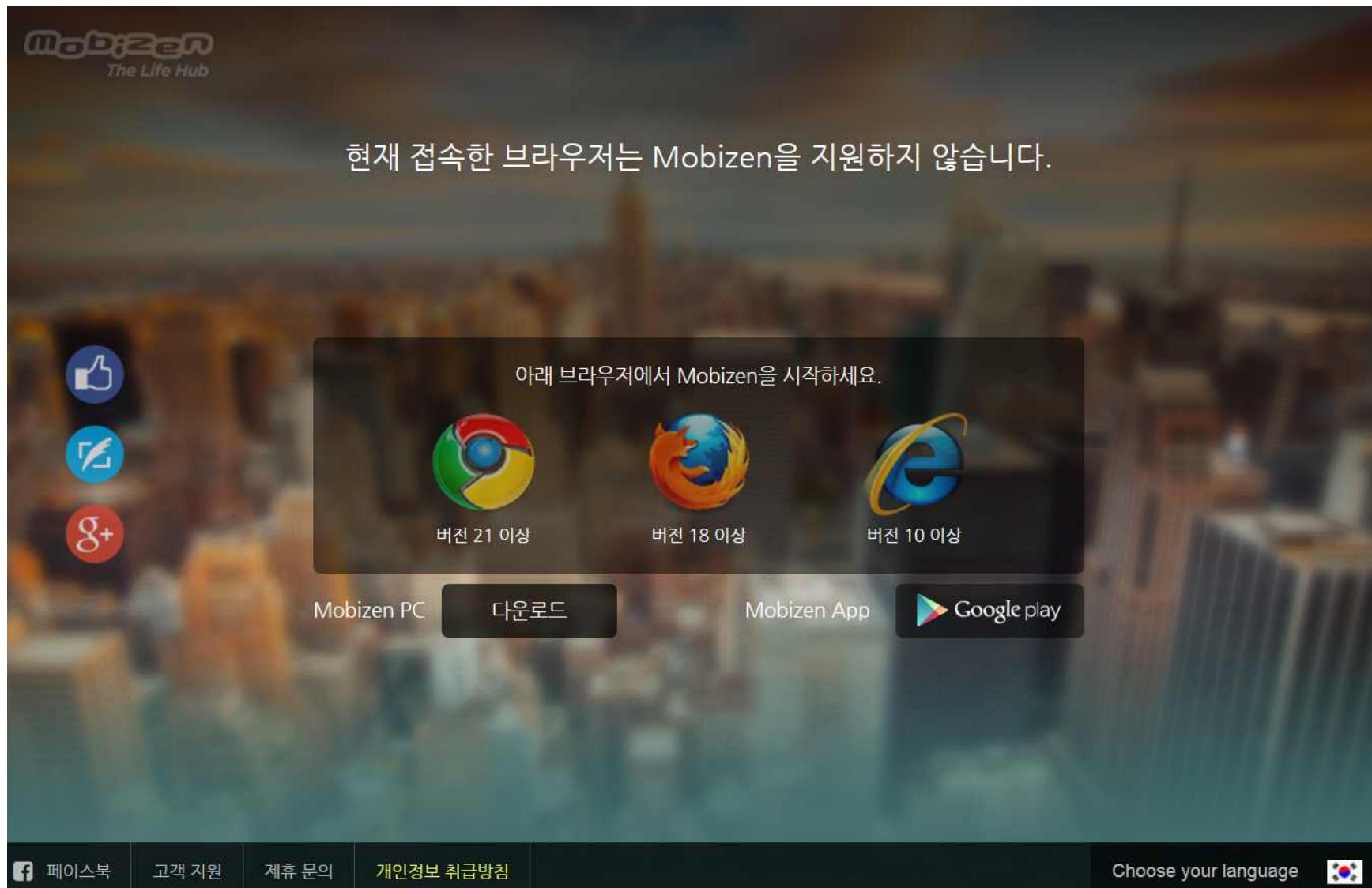
<https://developer.android.com/sdk/installing/studio-tips.html>



Mobizen: Search for Mobizen

The screenshot shows the NAVER search interface. The search bar at the top contains the query "mobizen". Below the search bar is a navigation menu with links to various categories like 통합검색, 블로그, 이미지, etc. Further down are filters for 정렬, 기간, 영역, and 옵션유지 (with options '꺼짐' and '켜짐'). A sidebar on the left lists related search terms under "연관검색어". The main content area displays search results for "사이트" (websites). The first result is for "Mobizen" (www.mobizen.pe.kr), which is described as providing mobile content information, reviews, and download links. This result has a red box around the "모비즌" link. The second result is for "Mobizen" (www.mobizen.com), described as a service for creating and sharing files. The third result is for "Mobizen" (www.mobizen.fr), described as a professional company for card rental, usage methods, videos, and fees. A "사이트 더보기 >" link is located at the bottom right of the website section.

Mobizen: Unsupported Web Browser



Mobizen: Search for Chrome

The screenshot shows the NAVER search interface. The search bar at the top contains the query "chrome". Below the search bar is a navigation menu with links to "통합검색", "어학사전", "이미지", "블로그", "지식iN", "지식백과", "실시간검색", "뉴스", and "더보기". Underneath this menu are filters for "정렬", "기간", "영역", "옵션유지" (with "꺼짐" and "켜짐" buttons), and "상세검색". A section titled "연관검색어" lists related terms: "한층 개선된 구글크롬 다운로드", "한층 업그레이드 된 chrome", "한층 개선된 chrome", "chrome 다운로드", "한층 개선된 chrome의 최신버전...", "핸드폰 해킹", and "chrome 삭제". There are "신고" and "X" buttons next to the last two items. A "더보기" button is also present. Below this is a "사이트" section featuring a link to "구글 크롬" (www.google.com/chrome) which is highlighted with a red box. Below the link are buttons for "속도", "간편함", "보안", "맞춤설정", and "개인정보 보호". A descriptive text follows: "구글 오픈소스 웹브라우저, 사이트 썸네일, 시크릿모드 등 기능 소개, 다운로드 제공.".

Mobizen: Downloading Chrome



브라우저

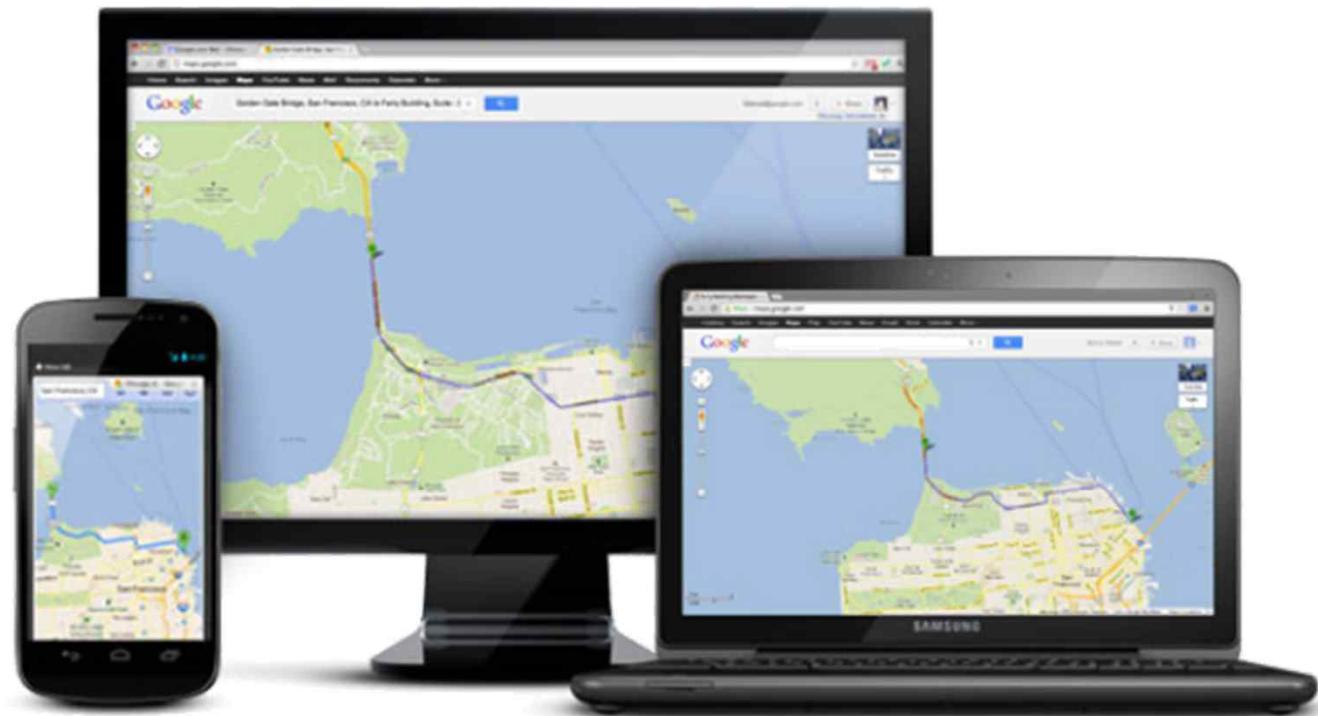
CHROMECAST

웹 스토어

Chrome 브라우저 탐색하기

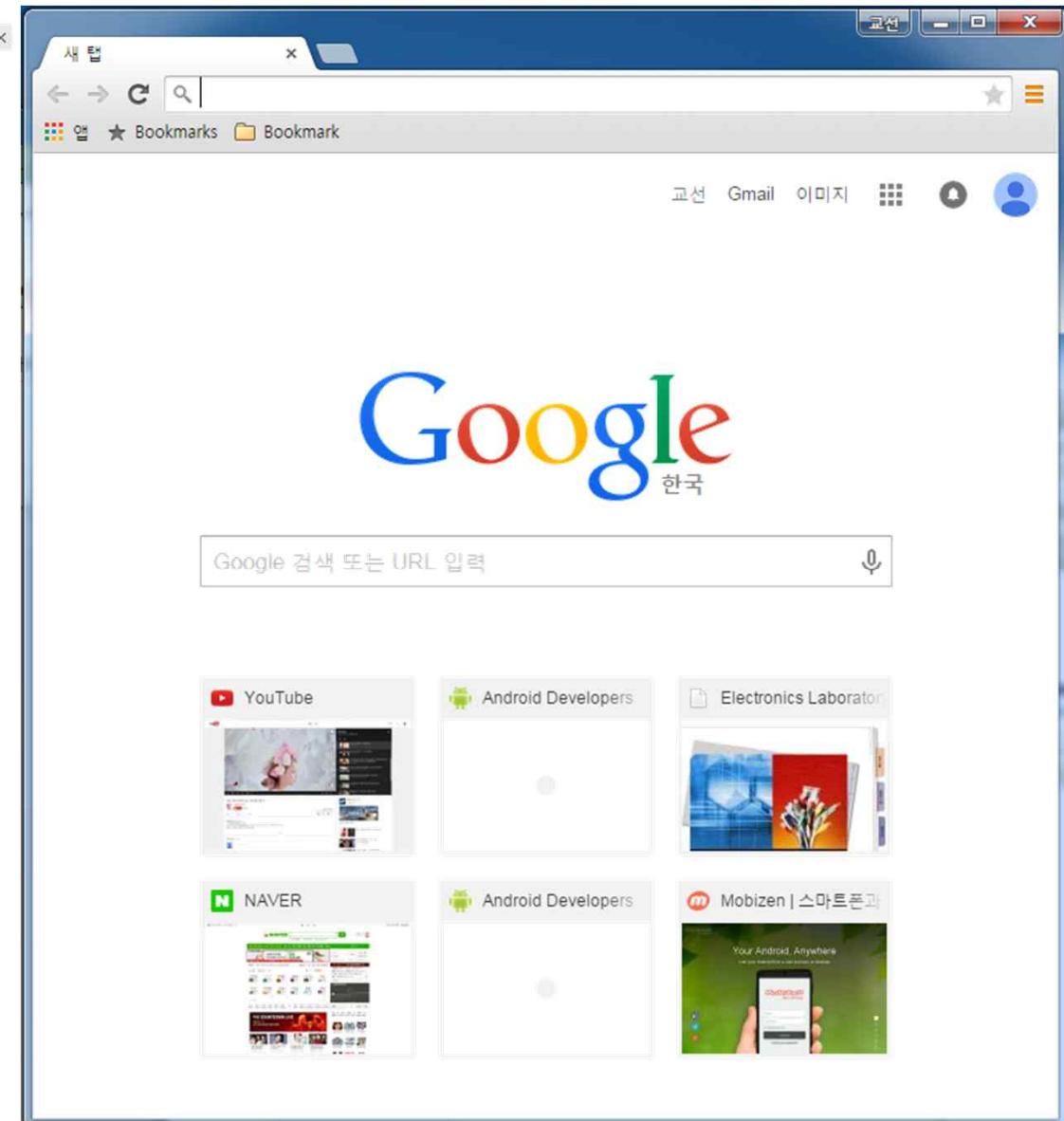
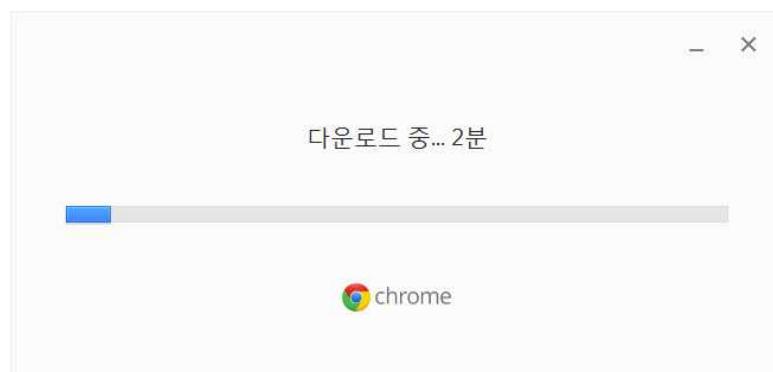
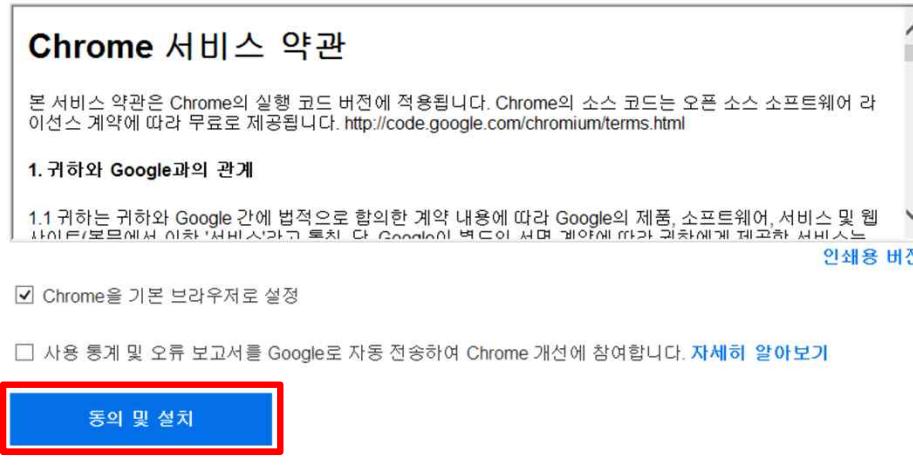
Chrome은 최신 웹에 최적화된 신속하고 간단하며 안전한 웹 브라우저입니다.

Chrome 다운로드



Mobizen: Downloading Chrome

차원이 다른 웹 서핑을 즐겨보세요.



Mobizen: Downloading Mobizen

The left screenshot shows a Google search results page for "mobizen". The search bar at the top contains "mobizen". Below it, there are tabs for "웹문서" (Web pages), "동영상" (Videos), "이미지" (Images), "지도" (Maps), "앱" (Apps), and "더보기" (More). The search results section shows a result for "파일전송은 플라잉파일 - flying-file.com" with a link to www.flying-file.com/. Below this, there are sections for "관련검색" (Related searches) and "바로가기" (Direct links). A red box highlights the link "Mobizen | Your Android, Anywhere" with the URL <https://mobizen.com/>.

The right screenshot shows the official Mobizen website at <https://mobizen.com>. The page has a dark background with a hand holding a smartphone displaying the Mobizen app interface. The app screen shows fields for "이메일" (Email) and "비밀번호" (Password), a checked checkbox for "이메일 저장" (Save email), and a "연결하기" (Connect) button. Below the phone, there are navigation icons. At the bottom, there are links for "Mobizen App 다운로드" (Download Mobizen App), "Mobizen PC 설치" (Install Mobizen PC), and "연결" (Connect). The footer includes links for Facebook, customer service, and legal information.

Android Studio Installation

March 28, 2015

Prof. Kyosun Kim
Incheon National University

Outline

1. References for Android Studio Installation
2. Java Compiler Download
3. Java Compiler Installation
4. Environment Variables for Java Compiler
5. Java Setup Test
6. Android Studio Download
7. Android Studio Installation
8. Android Phone Developer's Environment Setting
9. Android Studio Tour
 - ◆ Project Creation
 - ◆ SDK Download
 - ◆ Virtual Devices, Emulation
 - ◆ GUI Builder
 - ◆ Key Commands
10. Mobizen Installation

References for Android Studio Installation

□ Android Development Environment Setup Reference Sites

- ◆ <http://prolite.tistory.com/456> <= How to Install Android Studio
- ◆ <http://developer.android.com> <= Android Studio Installation
- ◆ <http://www.oracle.com/technetwork/java/javase/downloads/index.html?sourceSiteId=oocomen> <= Java Installation

Java Compiler Download

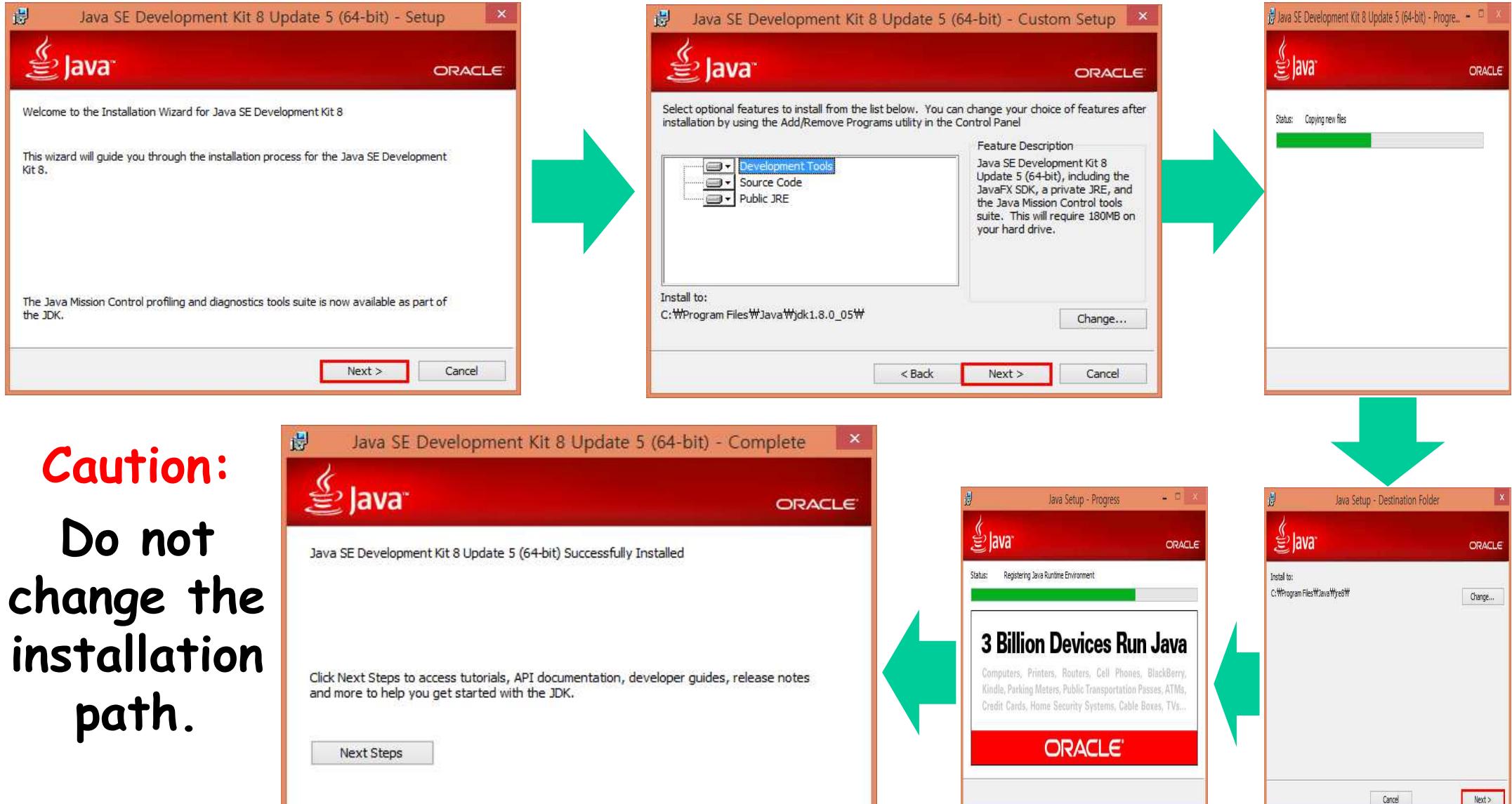
[http://www.oracle.com/technetwork/java/javase/downloads/index.html
?ssSourceSiteId=ocomen](http://www.oracle.com/technetwork/java/javase/downloads/index.html?ssSourceSiteId=ocomen) <= Java Installation Site.

The screenshot shows the Oracle Java SE Downloads page. Step 1 highlights the 'Java Platform (JDK) 8u25' download button with a red box and a red arrow pointing to it. Step 2 highlights the 'Accept License Agreement' radio button with a red box and a red arrow pointing to it. A large blue arrow points from the JDK download section to the license agreement section. The license agreement section contains the text: 'You must accept the Oracle Binary Code License Agreement for Java SE to download this software.' with two radio buttons: 'Accept License Agreement' (selected) and 'Decline License Agreement'. Below this is a table of download links for various Java versions and architectures.

Product / File Description	File Size	Download
Linux x86	135.24 MB	jdk-8u25-linux-i586.rpm
Linux x86	154.88 MB	jdk-8u25-linux-i586.tar.gz
Linux x64	135.6 MB	jdk-8u25-linux-x64.rpm
Linux x64	153.42 MB	jdk-8u25-linux-x64.tar.gz
Mac OS X x64	209.13 MB	jdk-8u25-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	137.01 MB	jdk-8u25-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	97.14 MB	jdk-8u25-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	137.11 MB	jdk-8u25-solaris-x64.tar.Z
Solaris x64	94.24 MB	jdk-8u25-solaris-x64.tar.gz
Windows x86	157.26 MB	jdk-8u25-windows-i586.exe
Windows x64	169.62 MB	jdk-8u25-windows-x64.exe

Download a version suitable for your PC.

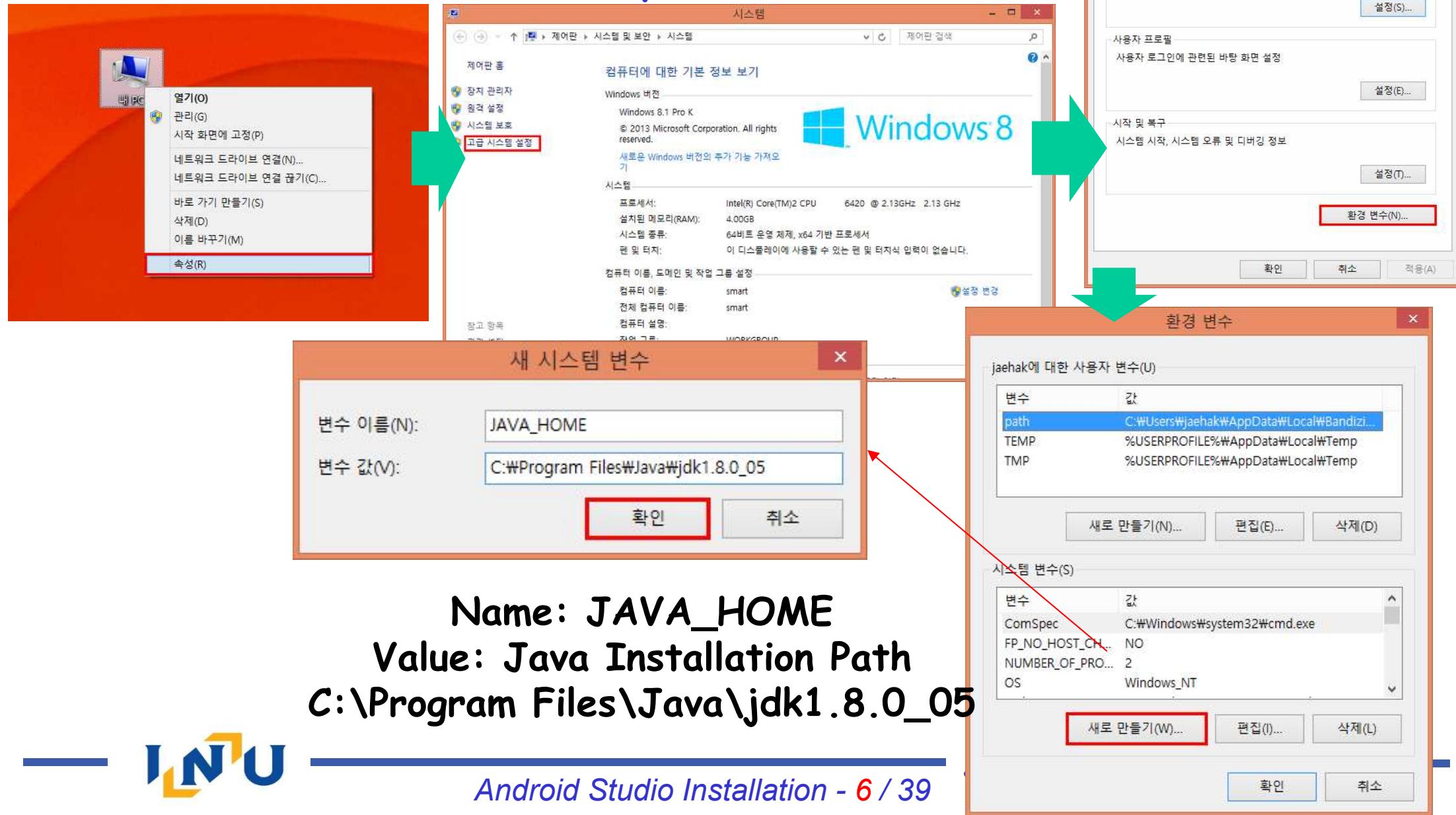
Java Compiler Installation



Caution:
Do not
change the
installation
path.

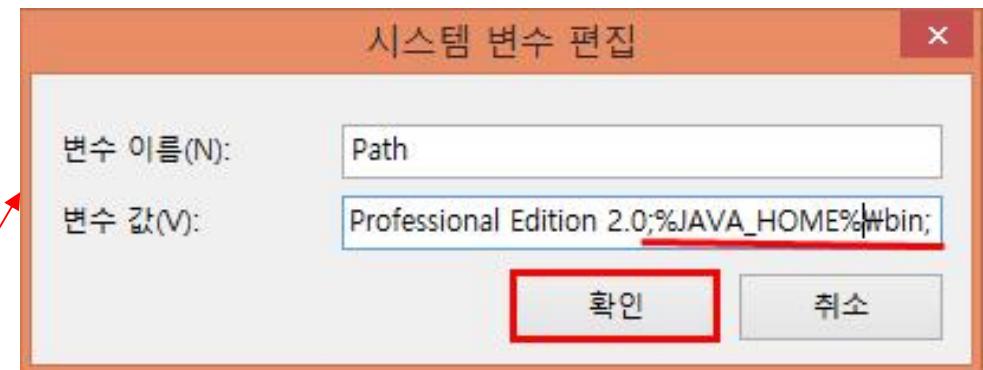
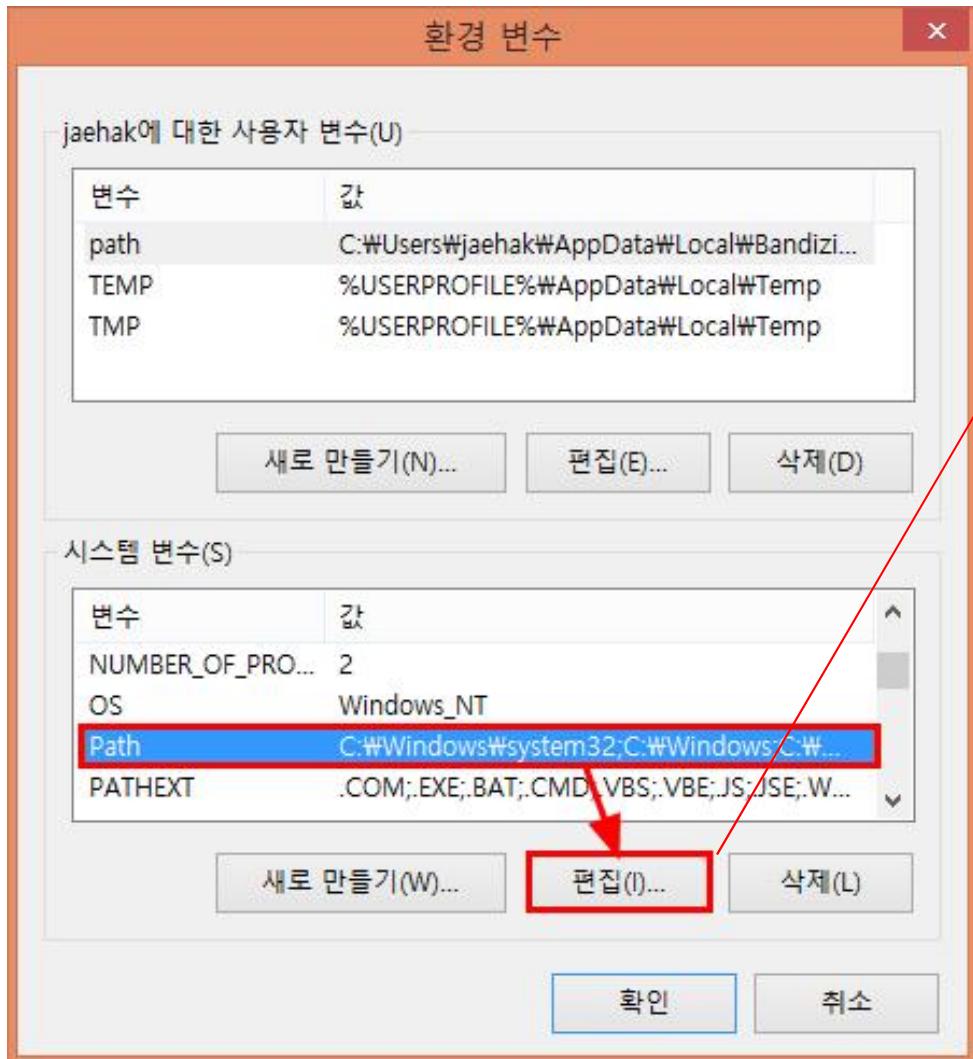
Environment Variables for Java Compiler (1 / 2)

□ Environment Variable Setup



Environment Variables for Java Compiler (2/3)

□ Environment Variable Setup



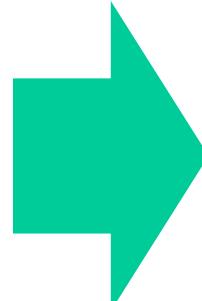
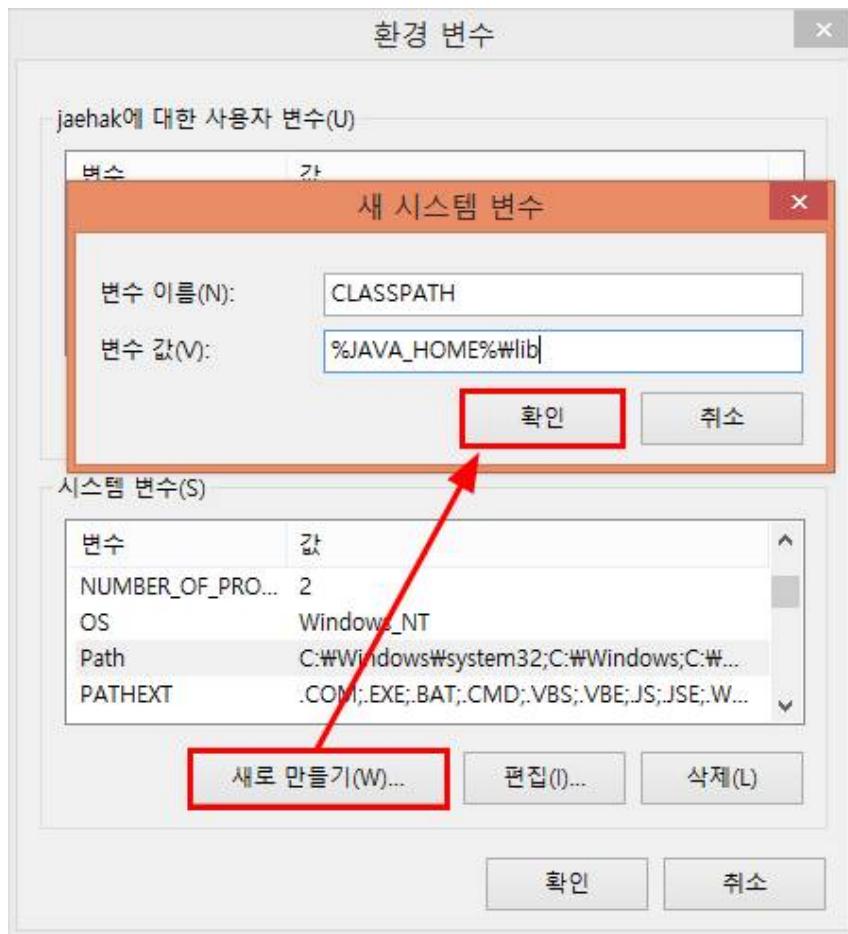
Value: ;%JAVA_HOME%\bin;
Make sure ; at both ends.

Environment Variables for Java Compiler (3/3)

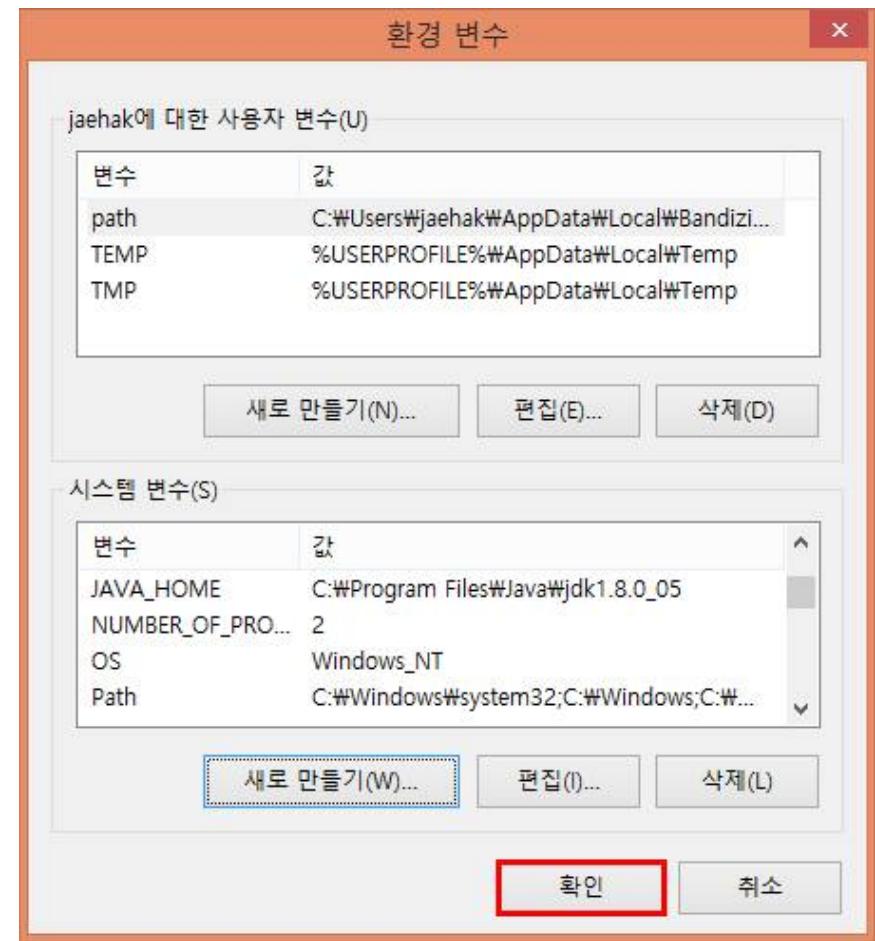
□ 환경변수 설정

Name: CLASSPATH

Value: %JAVA_HOME%\lib



최종적으로 확인!
확인 안 누르면 변경 값 저장 안됨!



Java Setup Test

□ cmd (명령 프롬프트)를 열어 환경변수 설치 확인

The image shows three separate command-line windows from a Windows operating system. Each window has a title bar '관리자: 명령 프롬프트'.

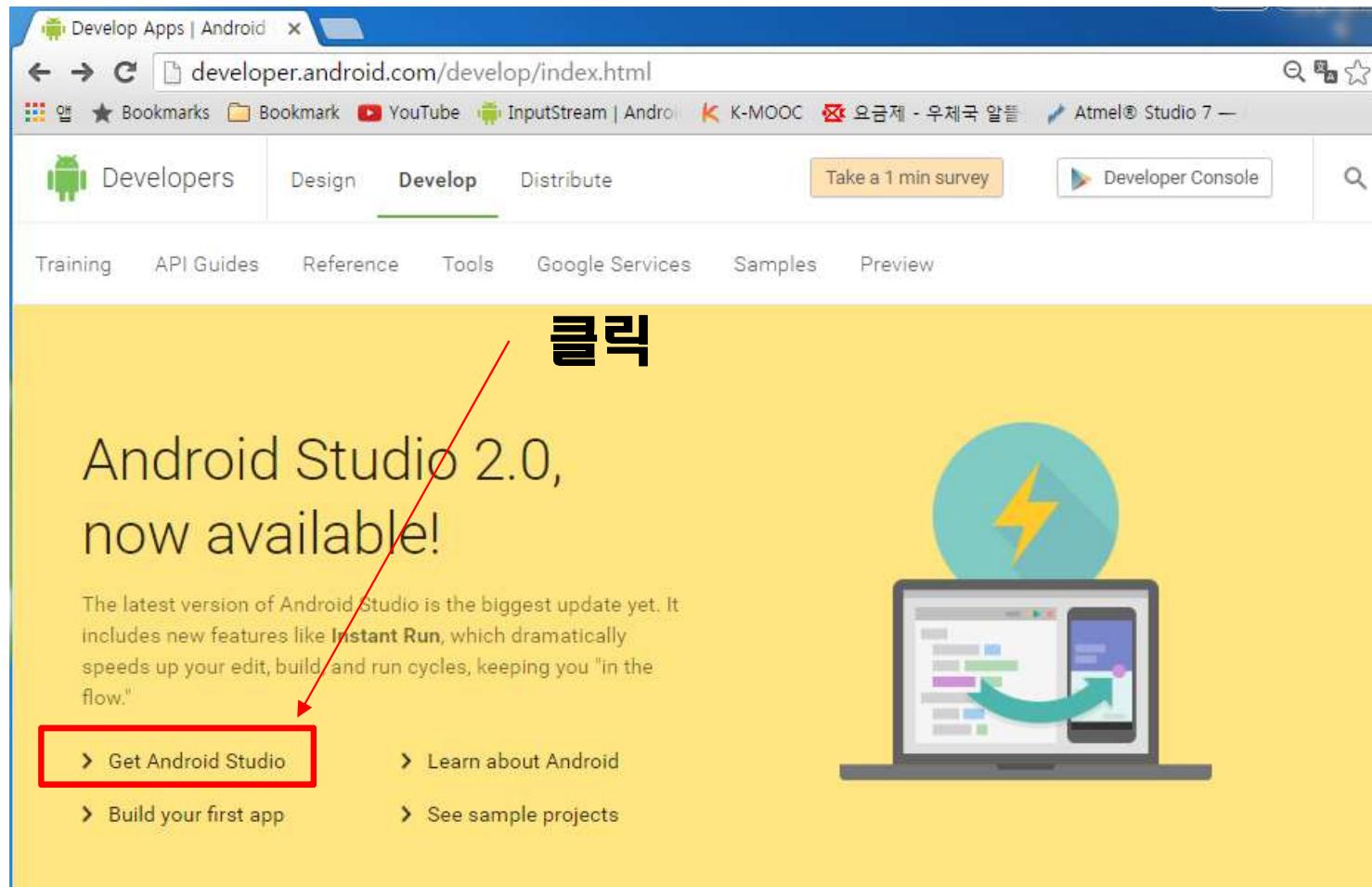
- Left Window (java):** Displays the usage information for the 'java' command. It includes options like '-d32', '-d64', and '-server' for VM selection, and '-cp' for classpath.
- Middle Window (javac):** Displays the usage information for the 'javac' command. It includes options for generating debugging info (-g), specifying classpath (-classpath), and bootclasspath (-bootclasspath).
- Right Window (java -version):** Displays the output of the 'java -version' command, which shows the Java version (1.8.0_05), the Java SE Runtime Environment (build 1.8.0_05-b13), and the Java HotSpot(TM) 64-Bit Server VM (build 25.5-b02, mixed mode).

위와 같이 안 뜨면 경로설정을 다시 한번 확인해 봄.
그리고 자바가 잘 실행되면 cmd 창은 신경 쓸 필요 없음.

Android Studio Download

□ Download

◆ <http://developer.android.com/sdk/index.html> < 설치 사이트로 접속.



Android Studio Download

□ Download

The screenshot shows the Android Developers website with the 'Develop' tab selected. A red arrow points from the 'Download Android Studio 2.1 FOR WINDOWS (1181 MB)' button on the left to the 'I have read and agree with the above terms and conditions' checkbox on the right. The steps are numbered 1, 2, and 3.

1 DOWNLOAD ANDROID STUDIO 2.1 FOR WINDOWS (1181 MB)

2 I have read and agree with the above terms and conditions

3 DOWNLOAD ANDROID STUDIO 2.1 FOR WINDOWS (1181 MB)

Android Studio
The Official IDE for Android

Android Studio provides the fastest tools for building apps on every type of Android device.

World-class code editing, debugging, performance tooling, a flexible build system, and an instant build/deploy system all allow you to focus on building unique and high quality apps.

Take a 1 min survey Developer Console Search

Training API Guides Reference Tools Google Services Samples Preview

Download the Android SDK Tools

Before downloading, you must agree to the following terms and conditions.

Terms and Conditions

This is the Android Software Development Kit License Agreement

1. Introduction

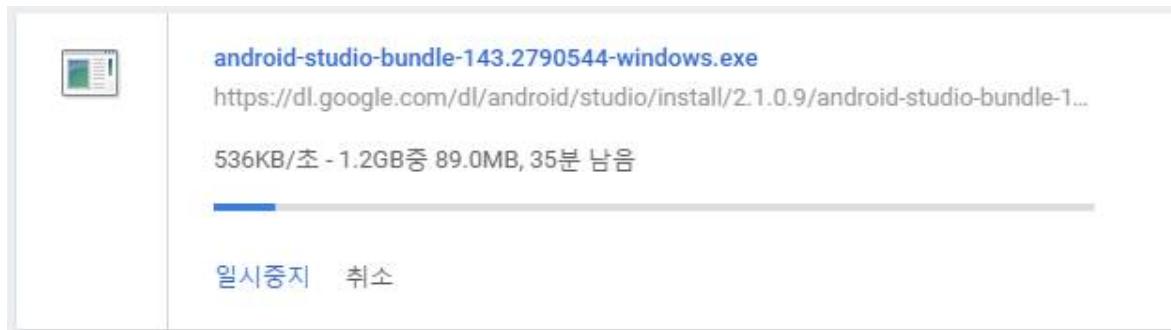
1.1 The Android Software Development Kit (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of the License Agreement. The License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.

1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: <http://source.android.com/>, as updated from time to time.

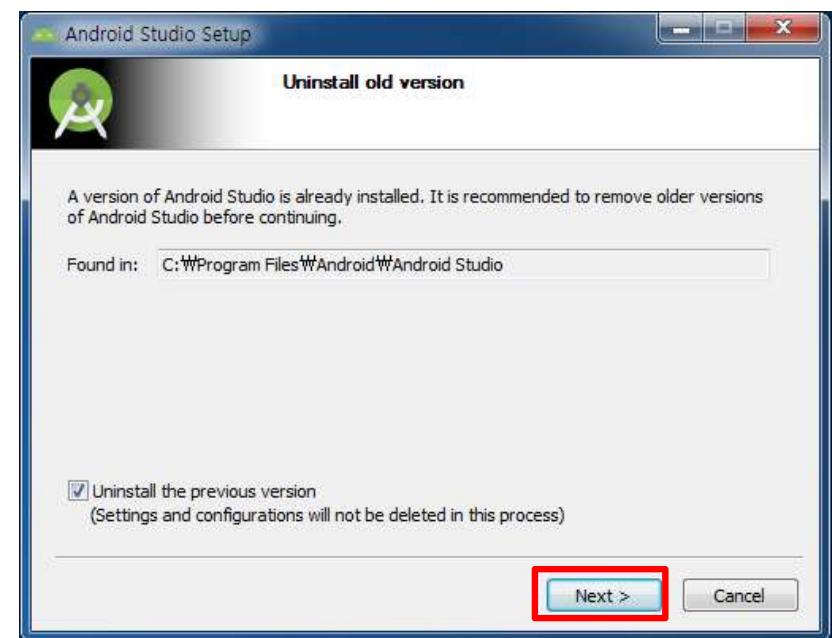
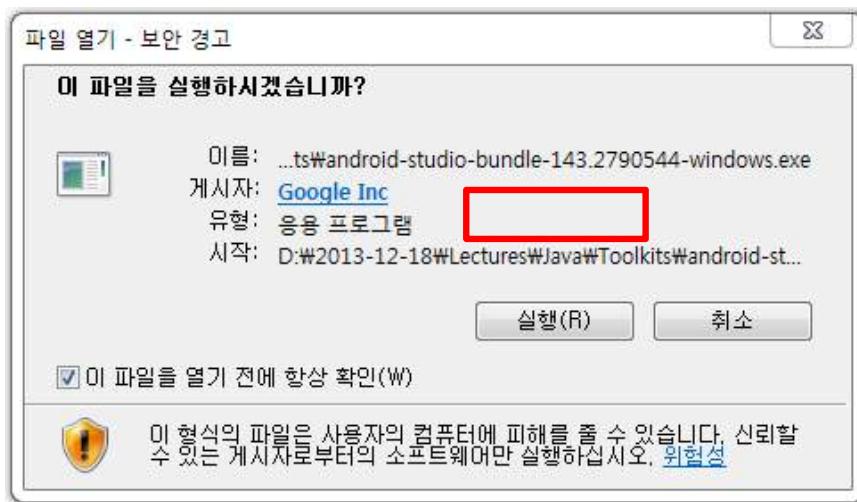
1.3 A "compatible implementation" means any Android device that (i) complies with the Android Compatibility

Android Studio Download

□ Download

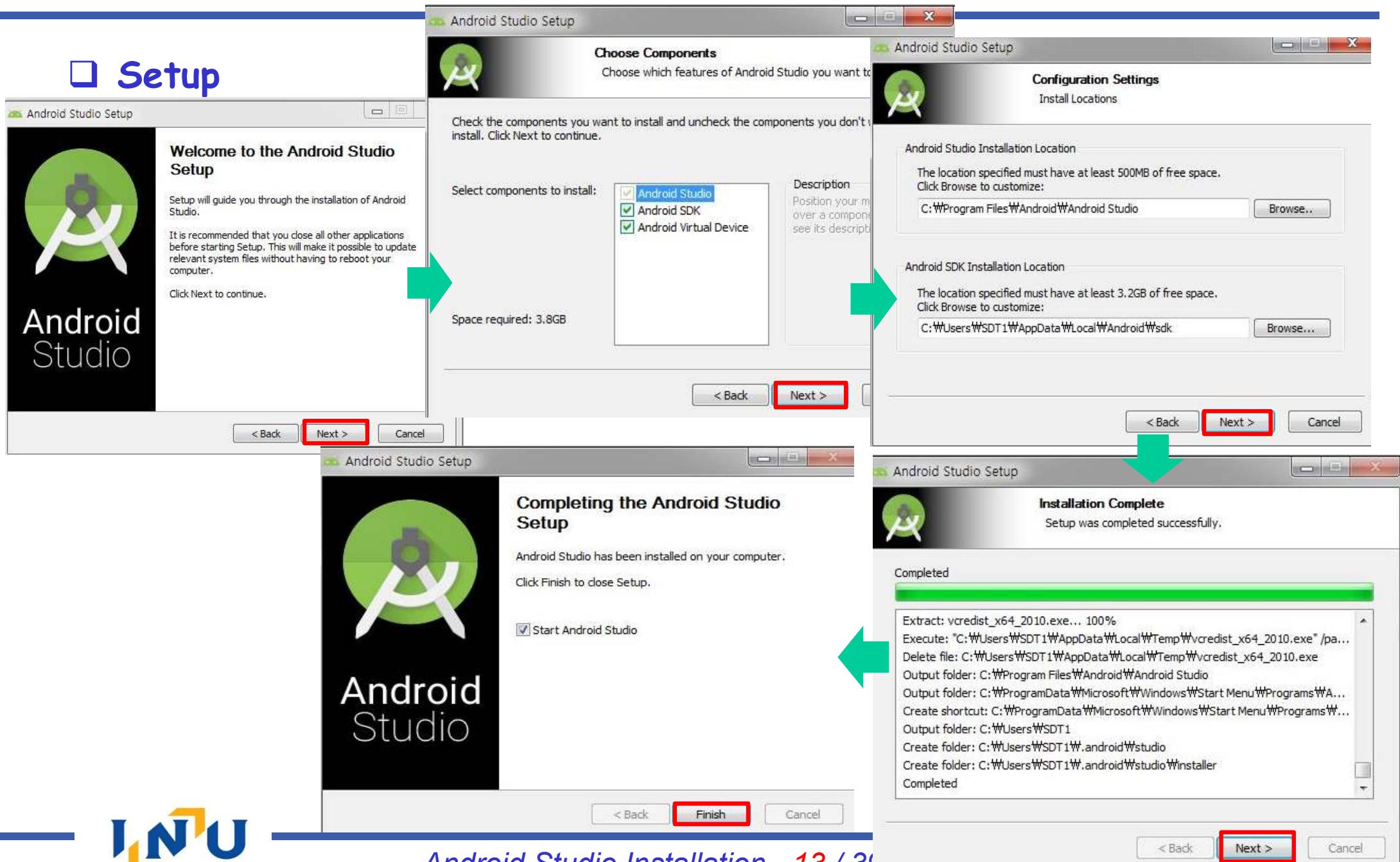


□ Install



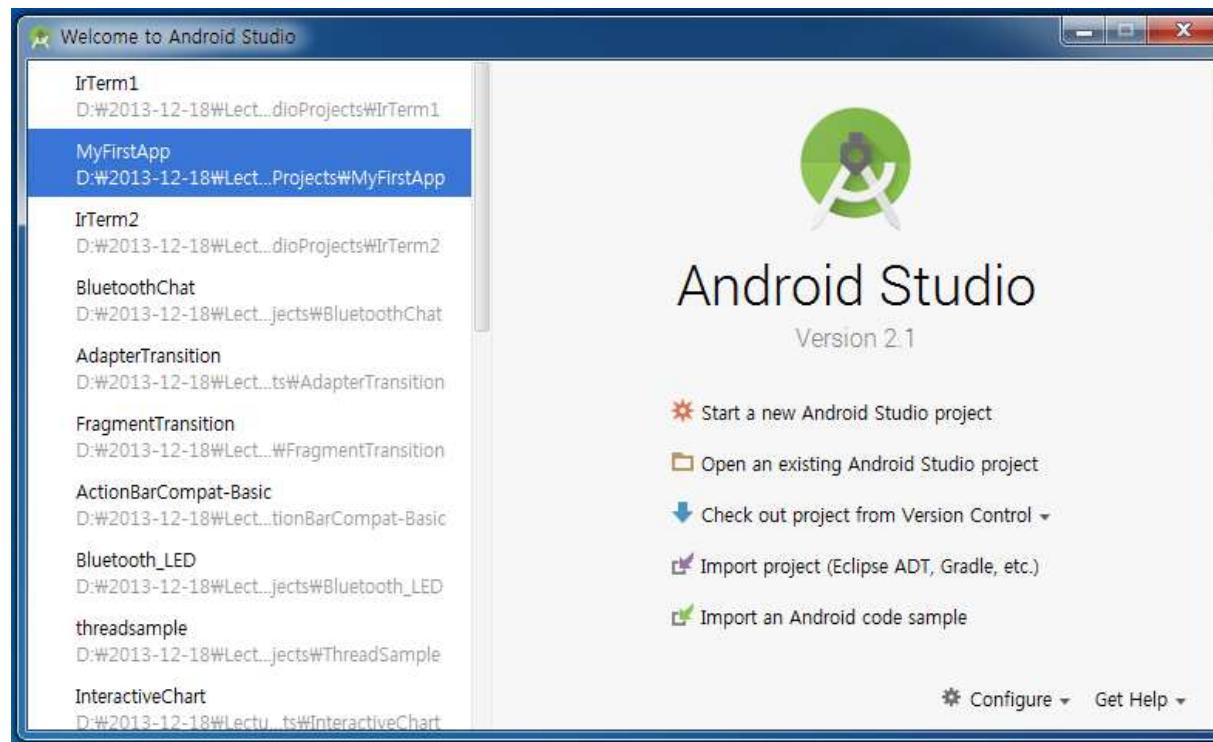
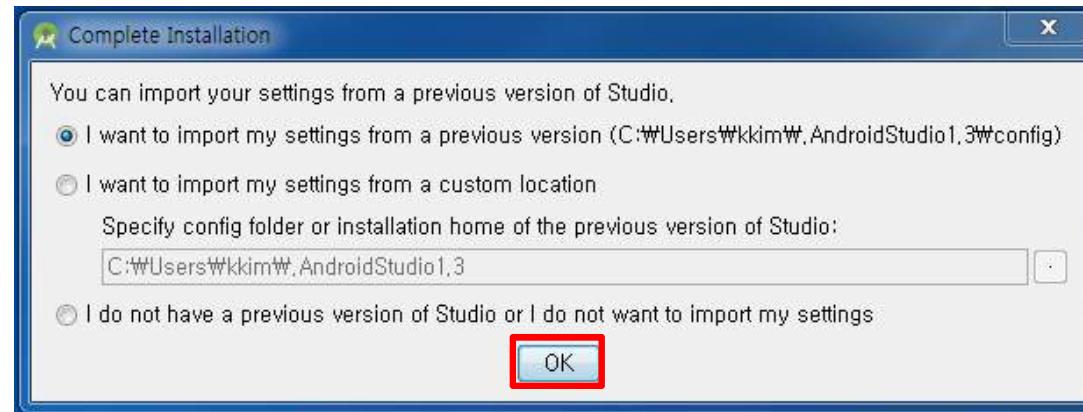
Android Studio Installation

Setup



Android Studio Installation

□ Setup



Android Phone Developer's Environment Setting

□ 휴대폰 설정 방법

◆ 환경설정 => 더보기 => 디바이스 정보

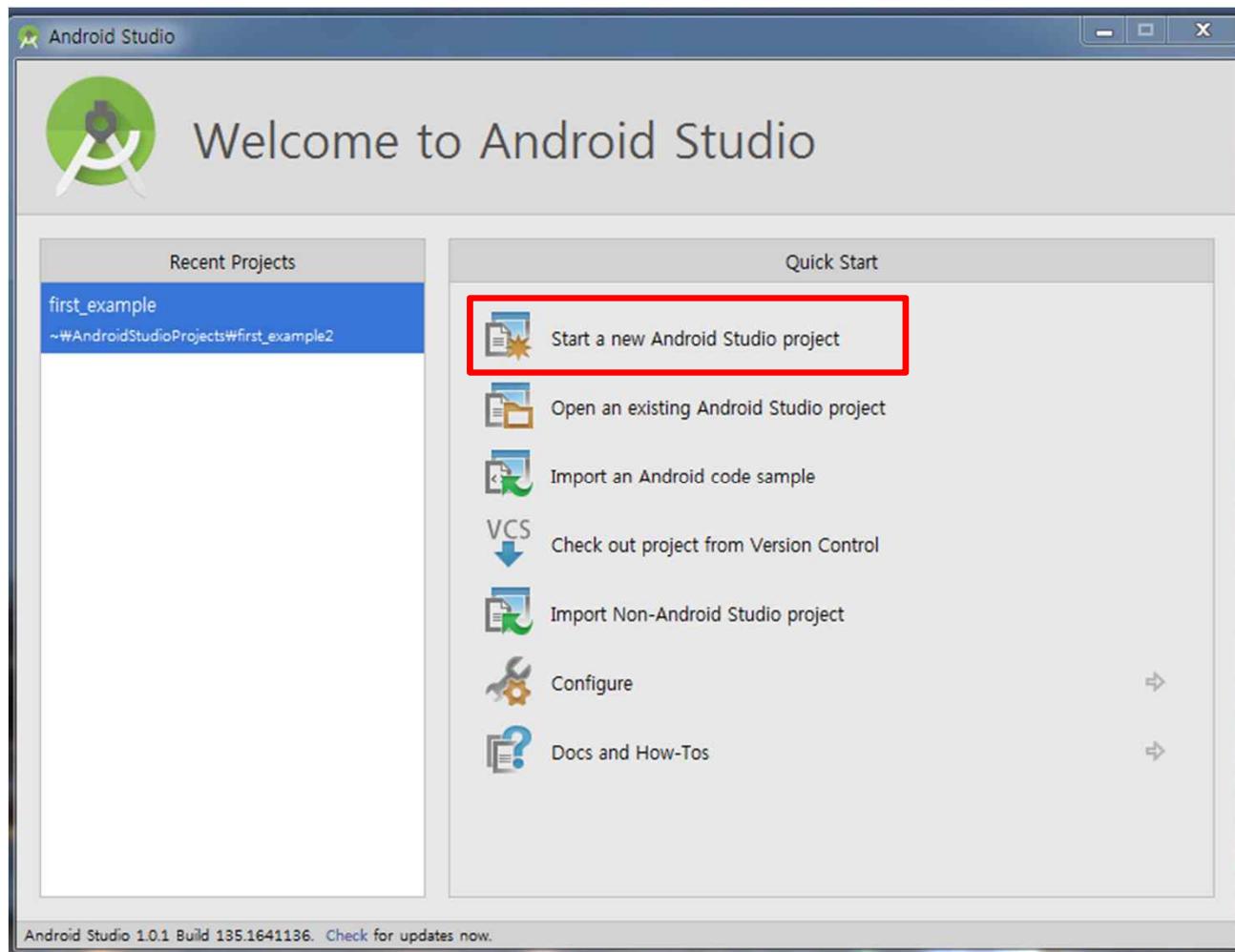


이 부분을 여러
번 누르면
개발자 모드가
실행 됨.



Android Studio Tour: Project Creation (1/5)

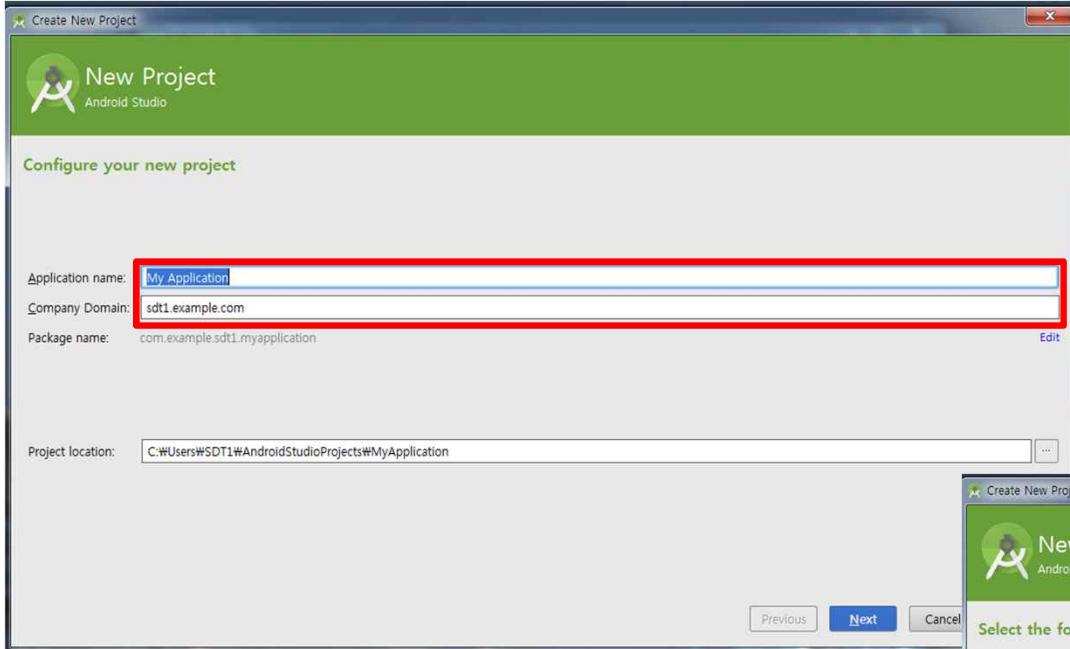
□ Creating a New Project



프로젝트 만들기
Start a new Android
Studio Project 클릭

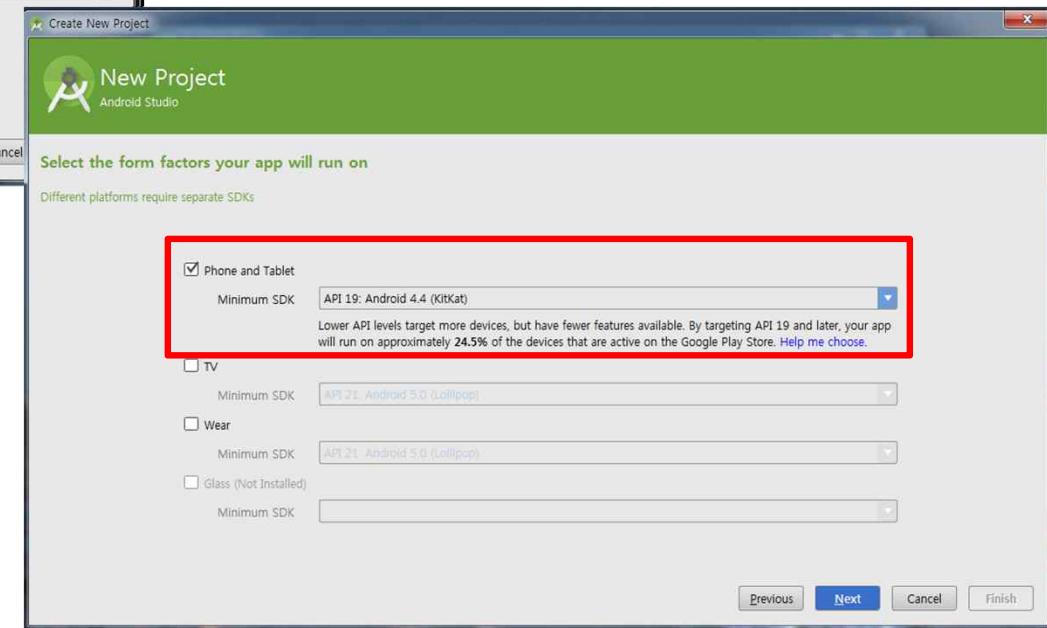
Android Studio Tour: Project Creation (2/5)

□ Android Training



Application 이름 설정

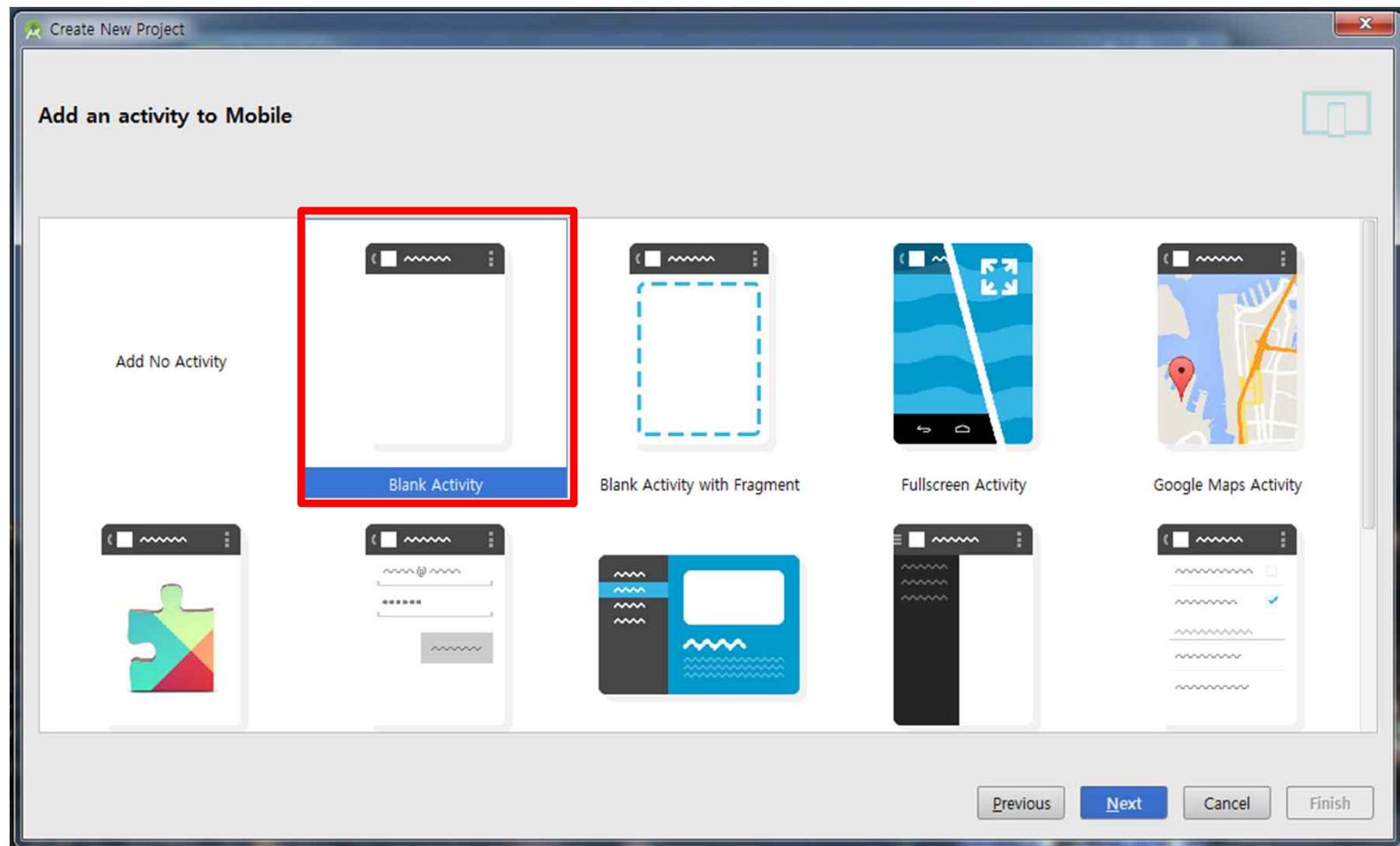
사용할 Android 버전을 설정한다.
2.2 (Froyo)로 한 이유: 가장
가볍기 때문 (시뮬레이션 시 많은
양의 메모리 소모 때문에, 간편한
테스트 시 2.2로 이용)



Android Studio Tour: Project Creation (3/5)

□ Android Training

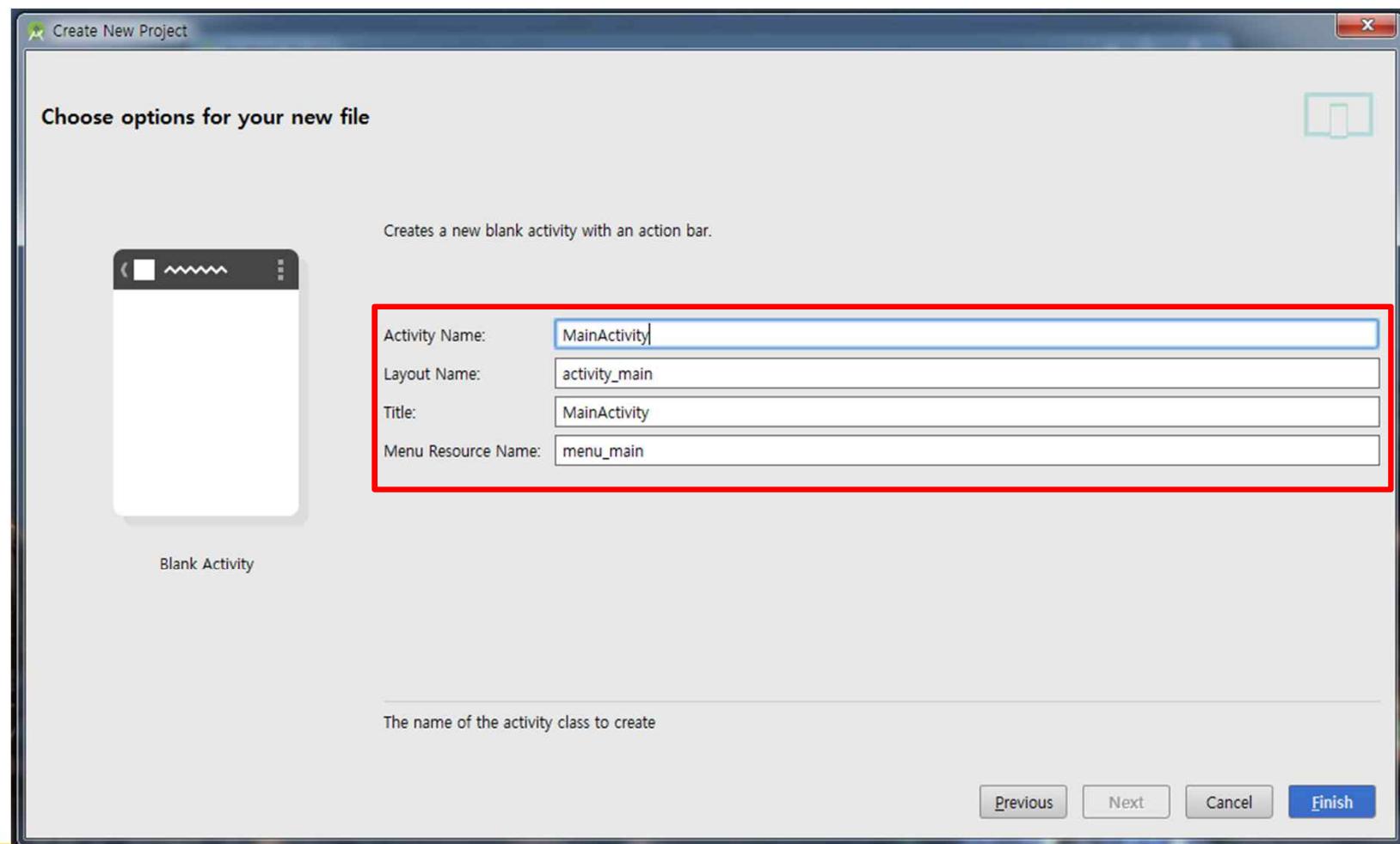
사용할 기본 Activity를 설정 할 수 있다.



Android Studio Tour: Project Creation (4/5)

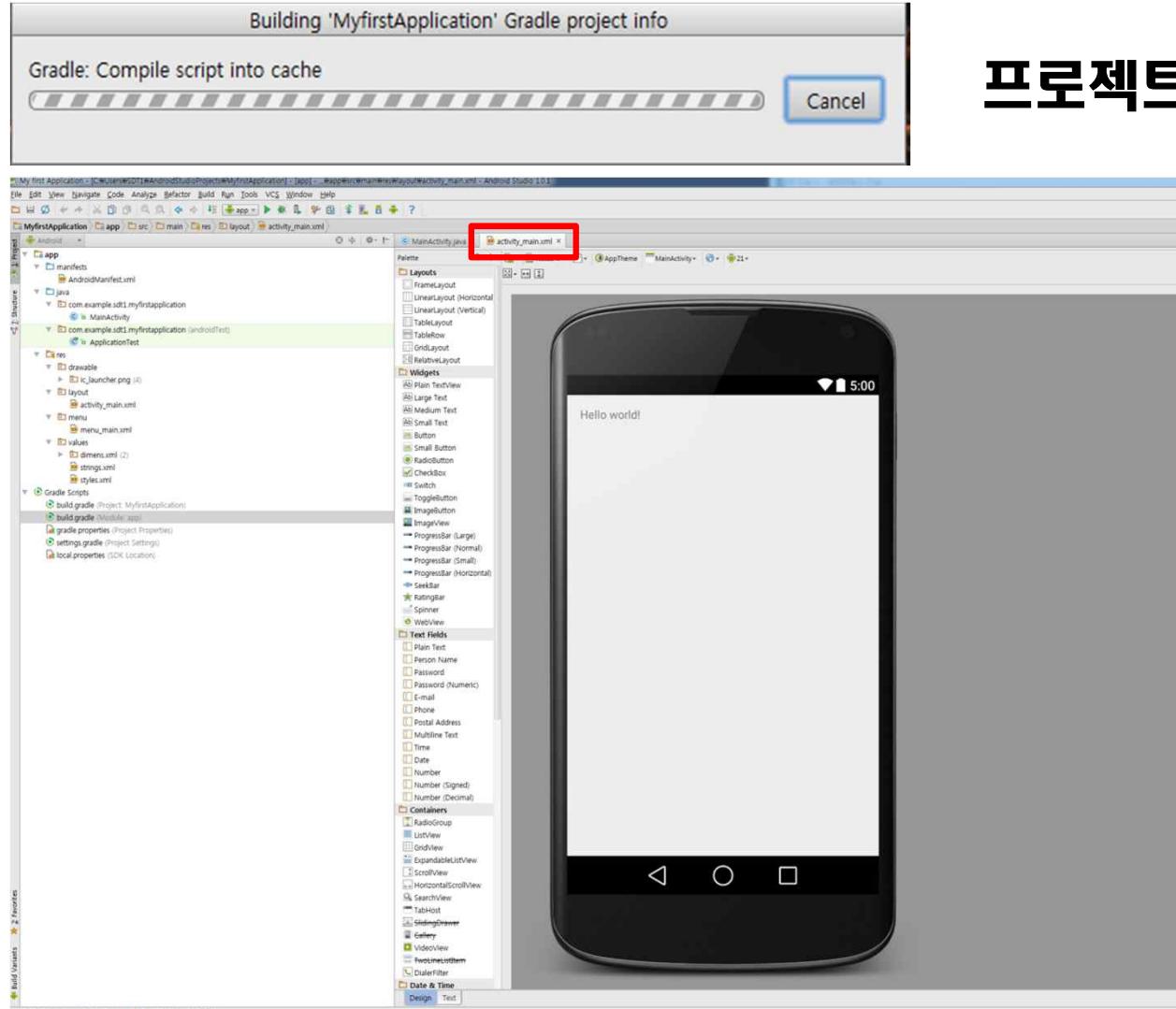
□ Android Training

Activity 이름을 설정한다.



Android Studio Tour: Project Creation (5/5)

□ Android Training

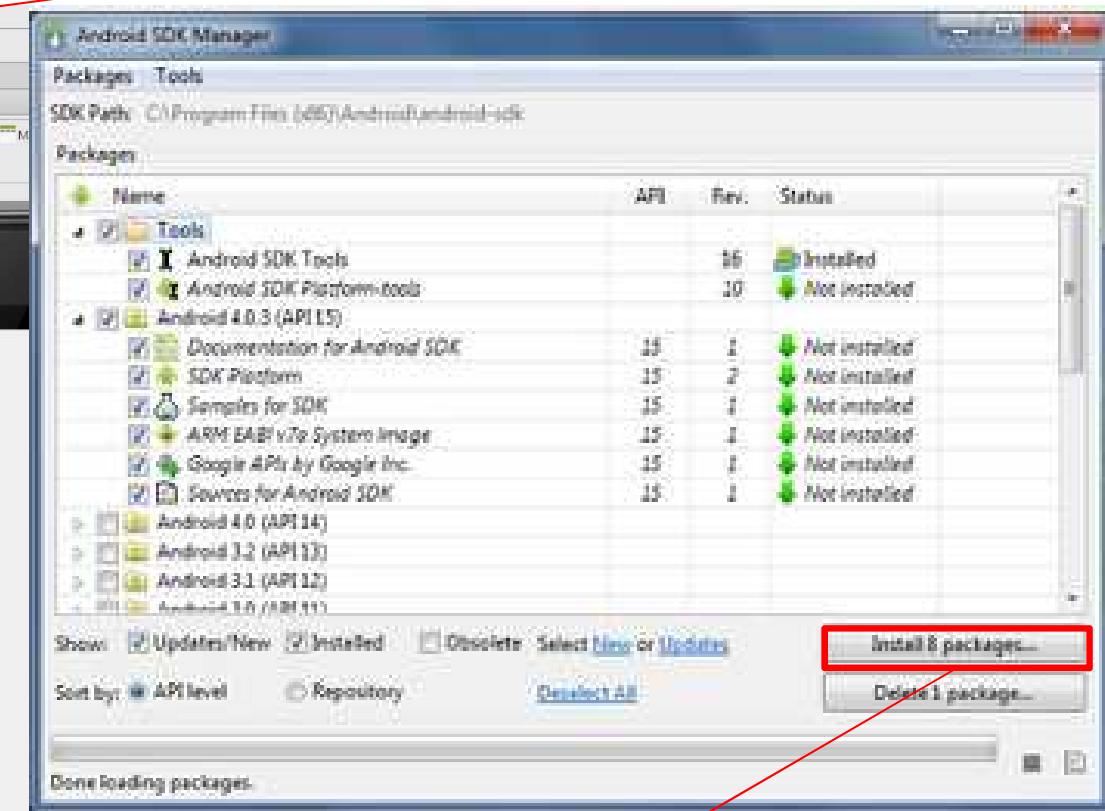
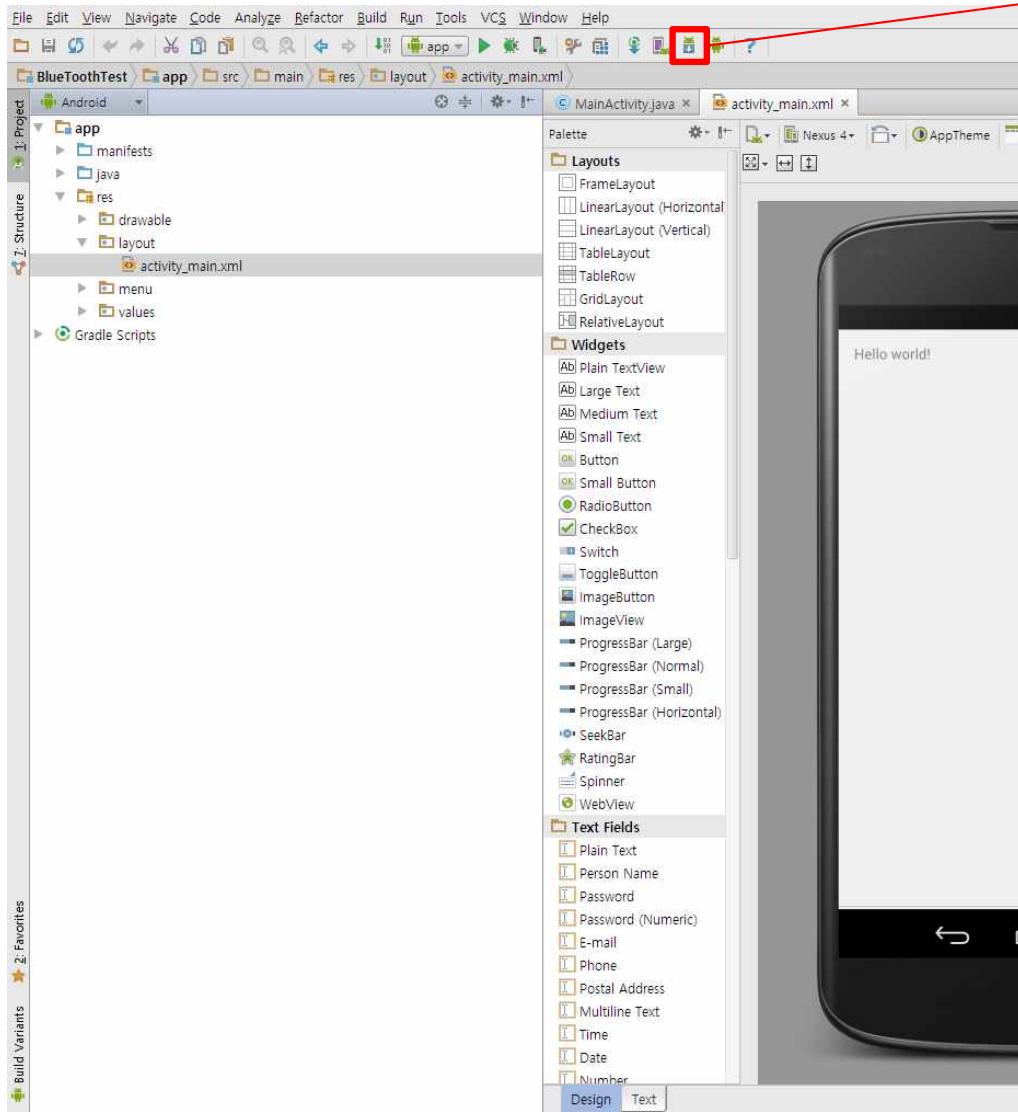


프로젝트 생성중...

다음과 같은 프로젝트
창이 발생 시뮬레이터를
확인 할 수 있는
**activity_프로젝트
명.xml** 창이 중앙에
뜨는 것을 확인 할 수
있다.

Android Studio Tour: SDK Download

□ Android Training

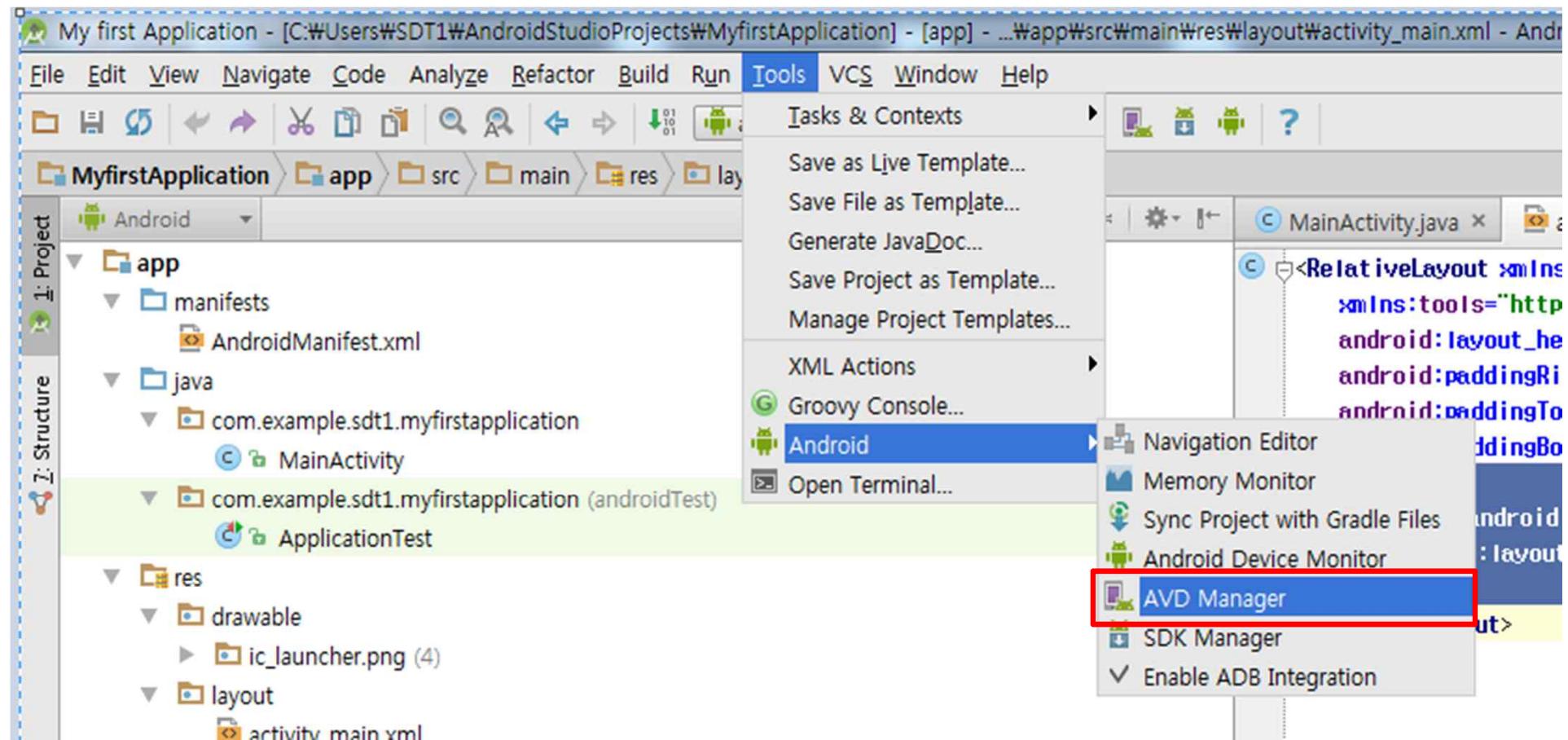


최초로 들어갔을 때
기본적으로 선택된 항목들은
반드시 인스톨 해야 함.

Android Studio Tour: Virtual Devices (1/3)

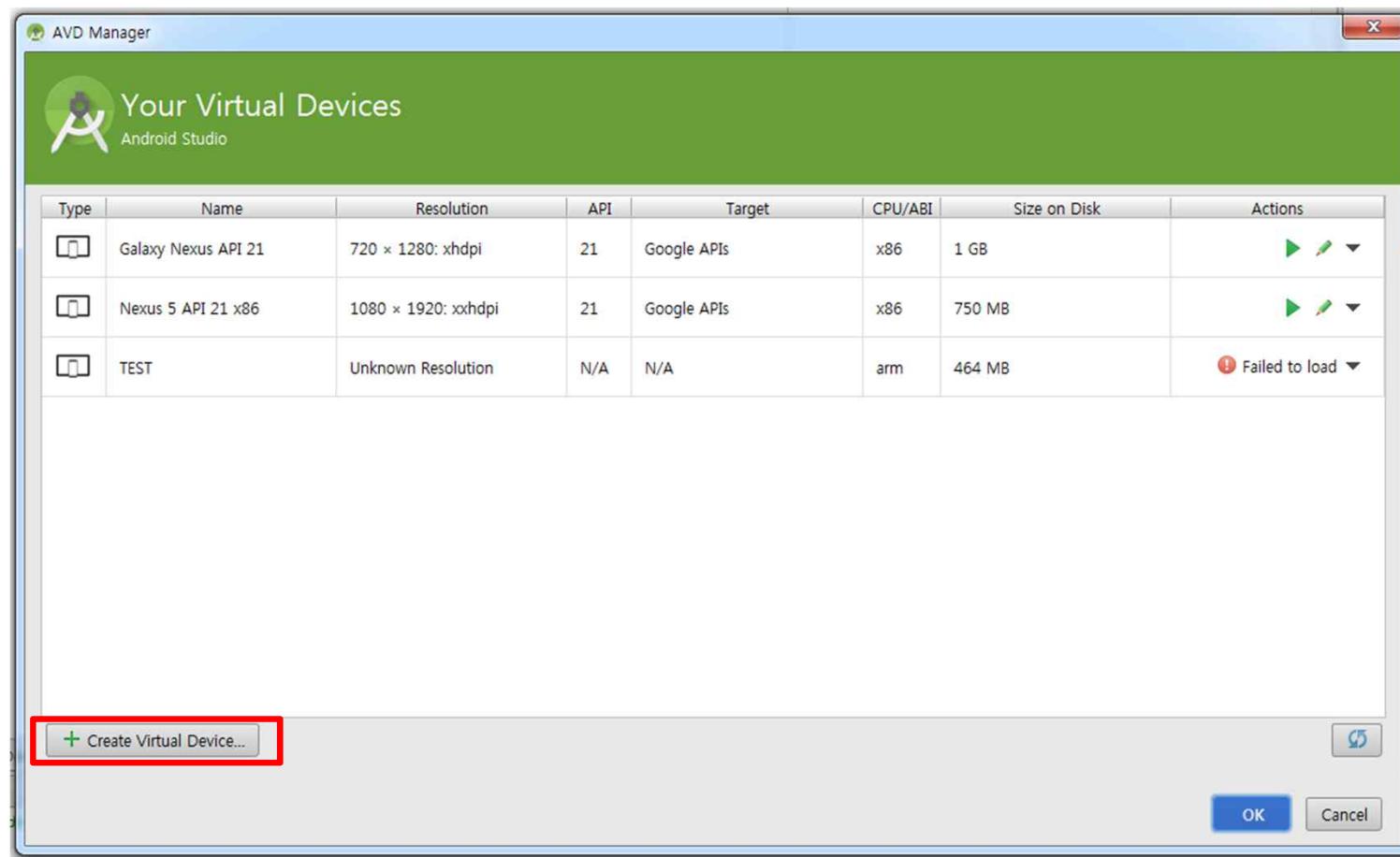
□ Android Training

시뮬레이션 동작을 위한 에뮬레이터
(장치디바이스 가종)를 추가한다.



Android Studio Tour: Virtual Devices (2/3)

□ Android Training

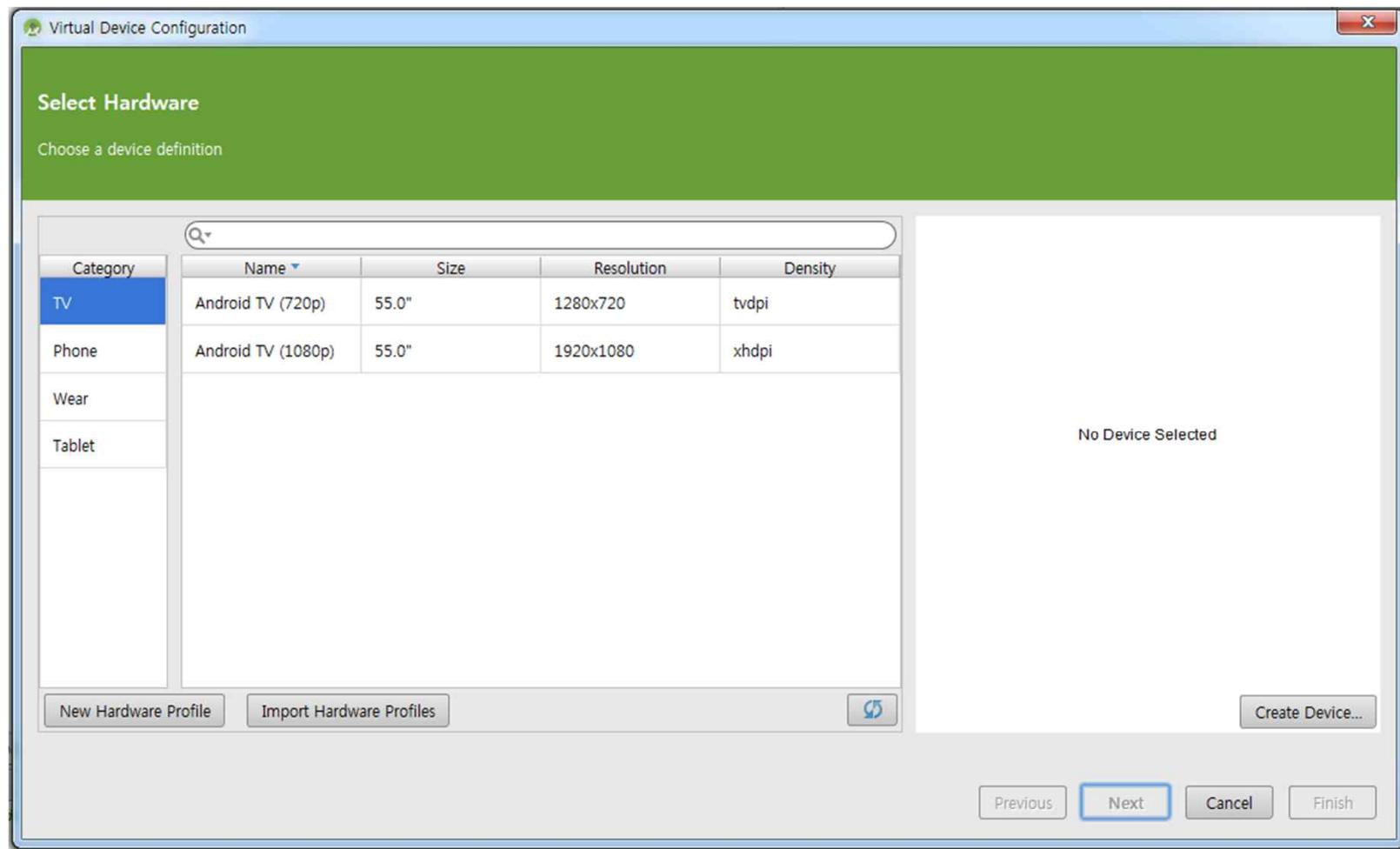


화면에 기존에
추가된 디바이스가
표시되고 있으며,
에뮬레이터로
사용할 디바이스를
원하는 기종으로
추가할 수 있다.

Android Studio Tour: Virtual Devices (3/3)

□ Android Training

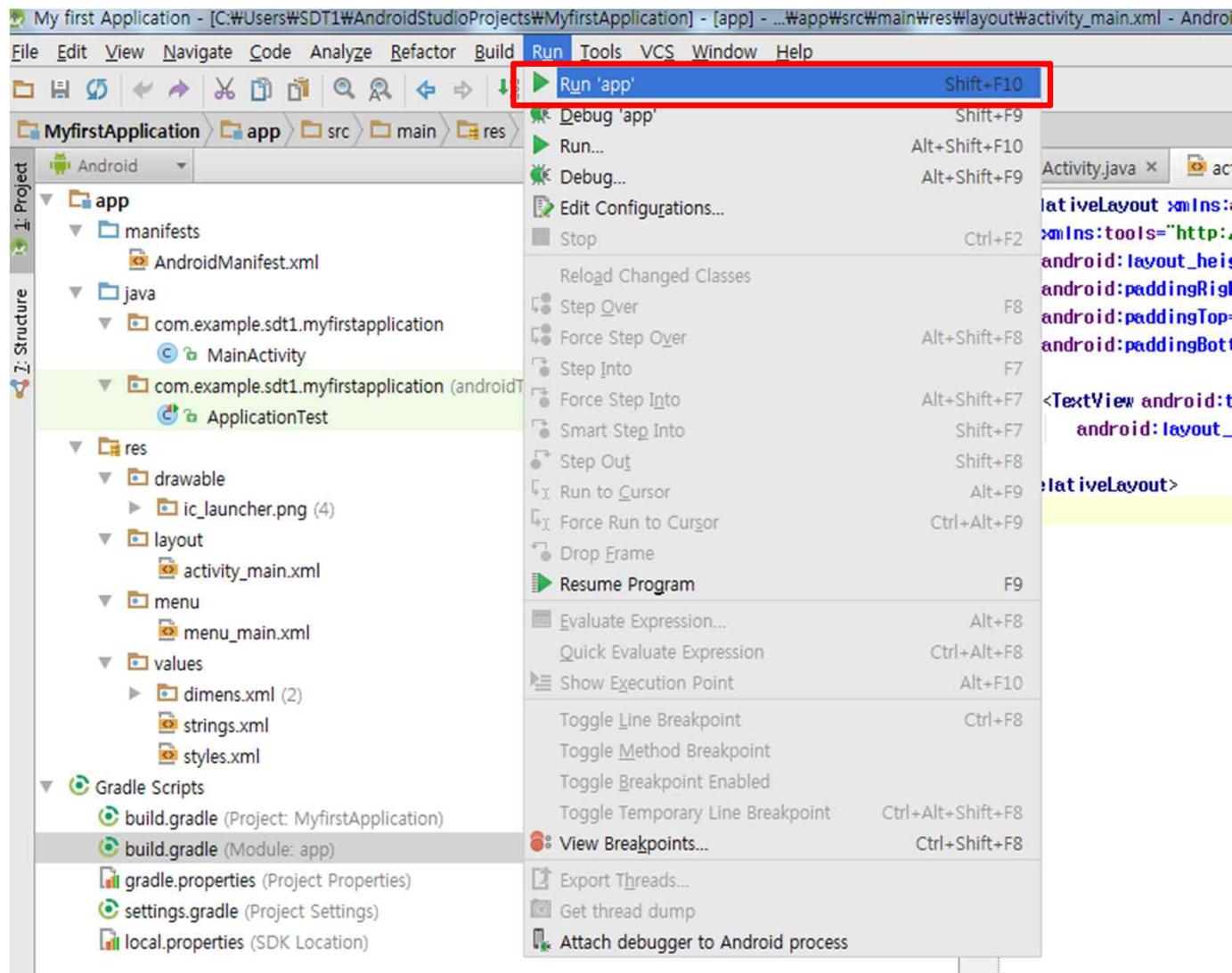
추가될 장비는 TV ~ wearable
디바이스 등등 가능하다.



Android Studio Tour: Emulation (1/3)

□ Android Training

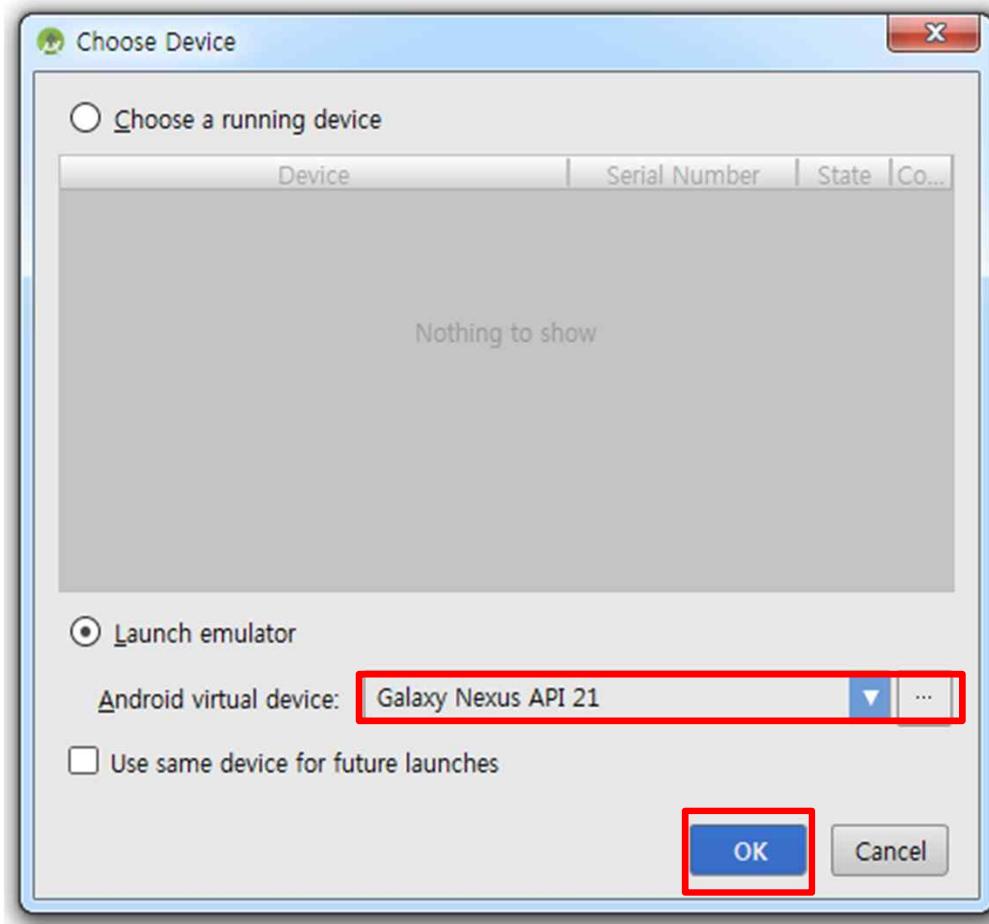
시뮬레이션을 동작시킨다.



Android Studio Tour: Emulation (2/3)

□ Android Training

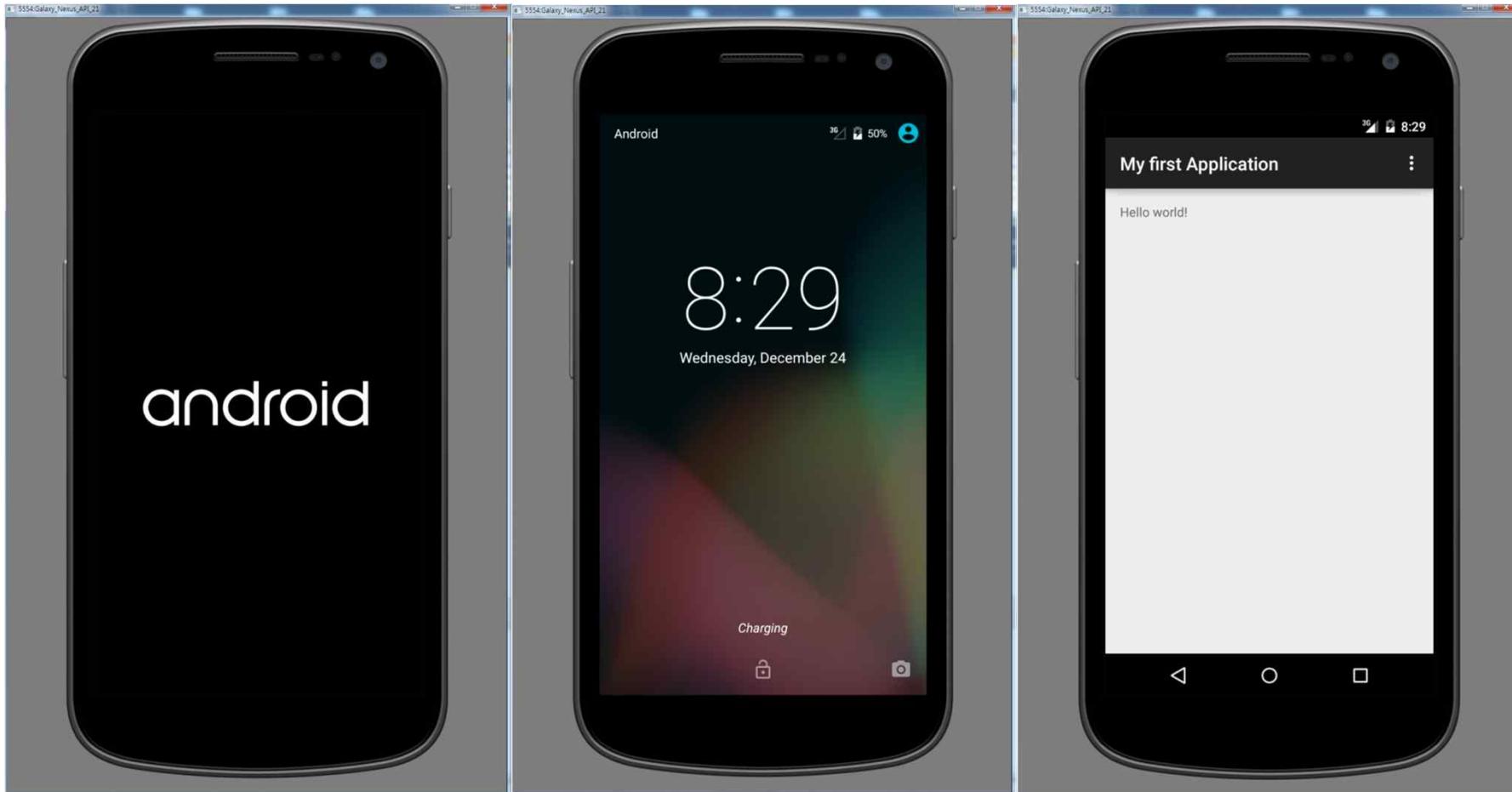
에뮬레이터로 사용할 장비 선택.



Android Studio Tour: Emulation (3/3)

□ Android Training

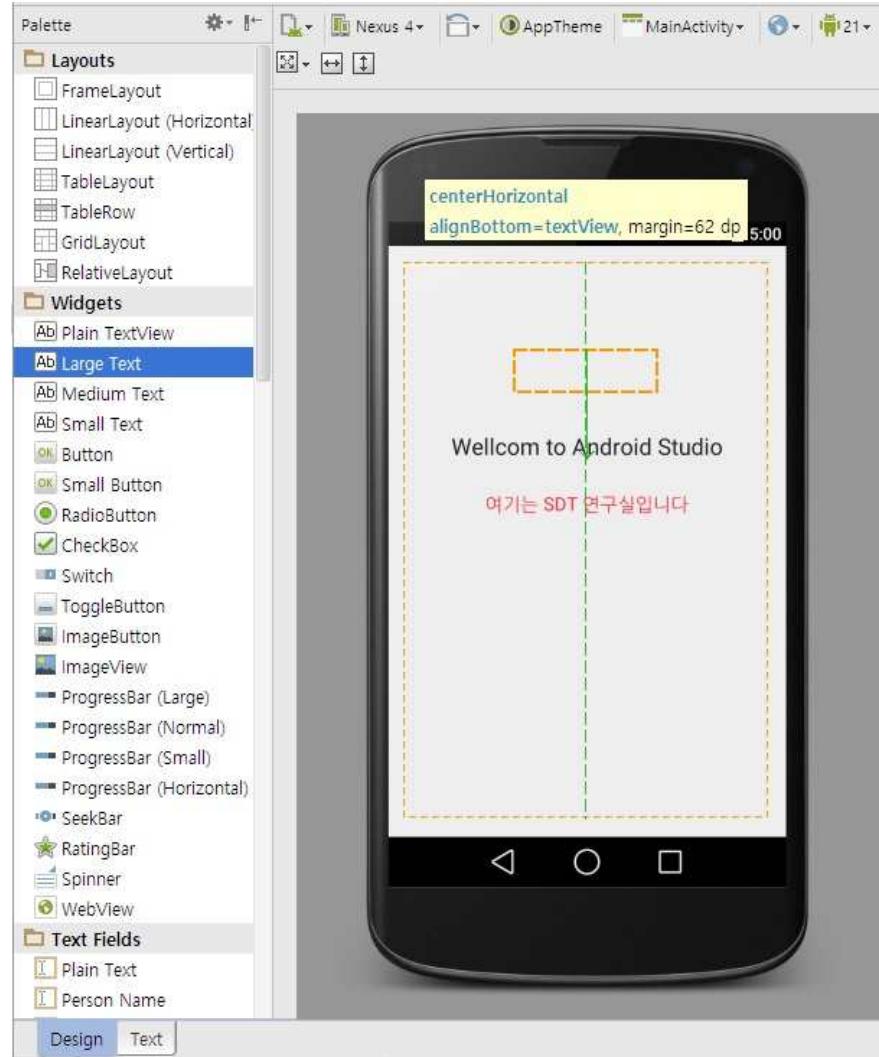
다음과 같은 창이 뜨며 실제 스마트 폰과 같은 동작을 한다.



앞에 정의된 창이 열리는 것을 확인할 수 있다.

Android Studio Tour: GUI Builder (1/4)

□ Android Training

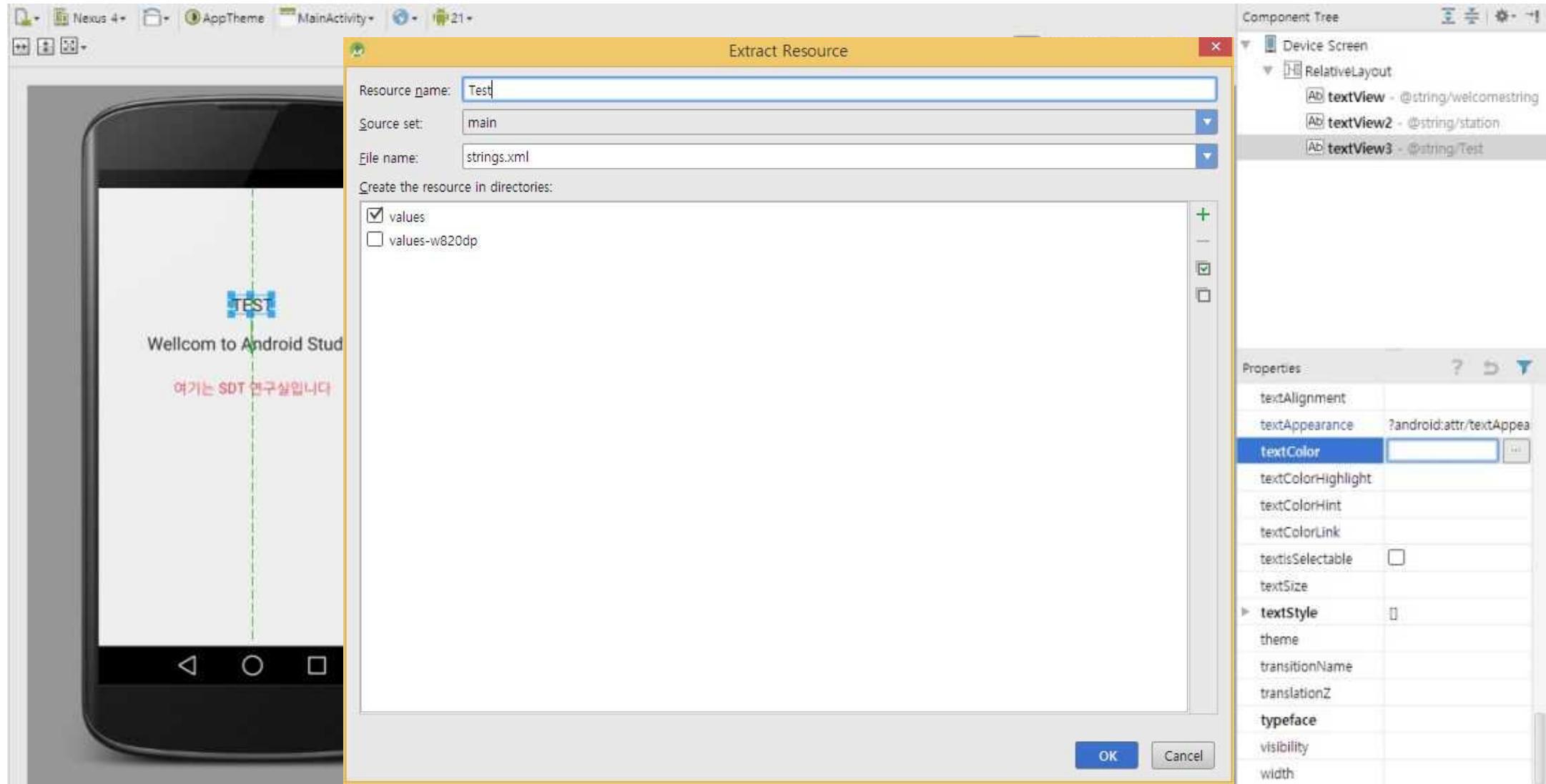


글자 색과 크기 등등 변경하여 표시



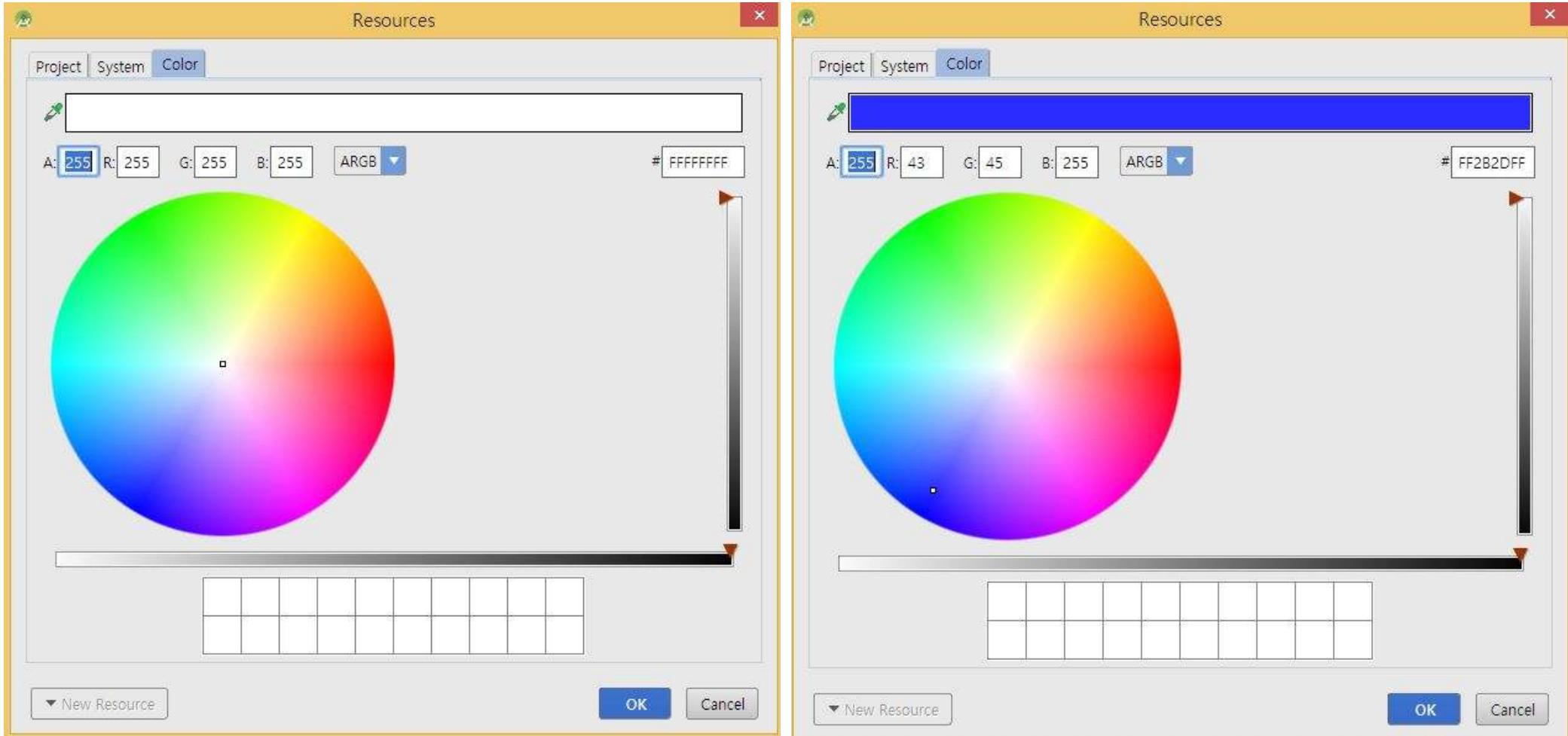
Android Studio Tour: GUI Builder (2/4)

□ Android Training



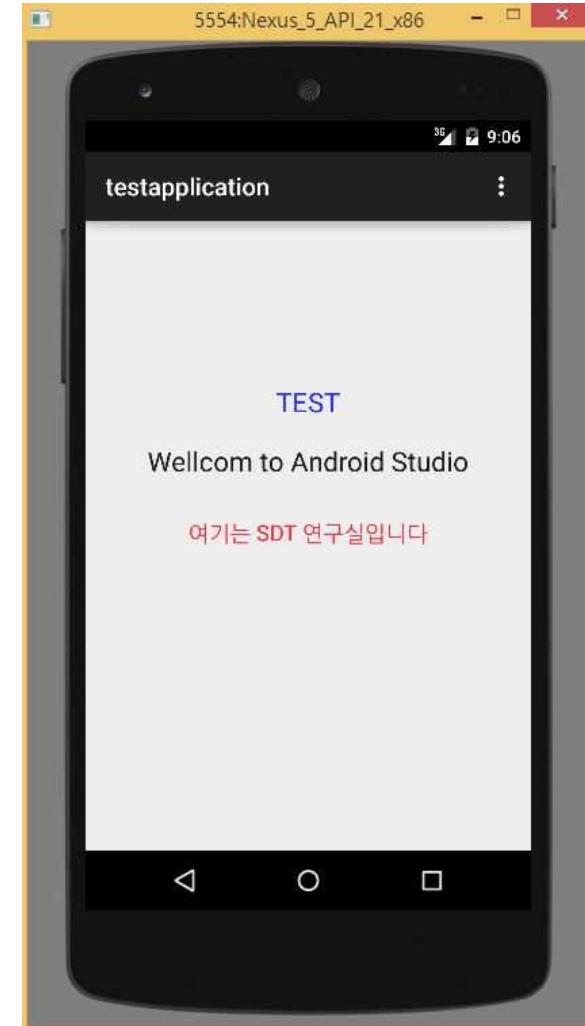
Android Studio Tour: GUI Builder (3/4)

□ Android Training



Android Studio Tour: GUI Builder (4/4)

□ Android Training



Android Studio Tour: Key Commands

Table 1. Programming key commands

Action	Android Studio Key Command
Command look-up (autocomplete command name)	CTRL + SHIFT + A
Project quick fix	ALT + ENTER
Reformat code	CTRL + ALT + L (Win) OPTION + CMD + L (Mac)
Show docs for selected API	CTRL + Q (Win) F1 (Mac)
Show parameters for selected method	CTRL + P
Generate method	ALT + Insert (Win) CMD + N (Mac)
Jump to source	F4 (Win) CMD + down-arrow (Mac)
Delete line	CTRL + Y (Win) CMD + Backspace (Mac)
Search by symbol name	CTRL + ALT + SHIFT + N (Win) OPTION + CMD + O (Mac)

Android Studio Tour: Key Commands (Cont'd)

Table 2. Project and editor key commands

Action	Android Studio Key Command
Build	CTRL + F9 (Win) CMD + F9 (Mac)
Build and run	SHIFT + F10 (Win) CTRL + R (Mac)
Toggle project visibility	ALT + 1 (Win) CMD + 1 (Mac)
Navigate open tabs	ALT + left-arrow; ALT + right-arrow (Win) CTRL + left-arrow; CTRL + right-arrow (Mac)

참조 사이트

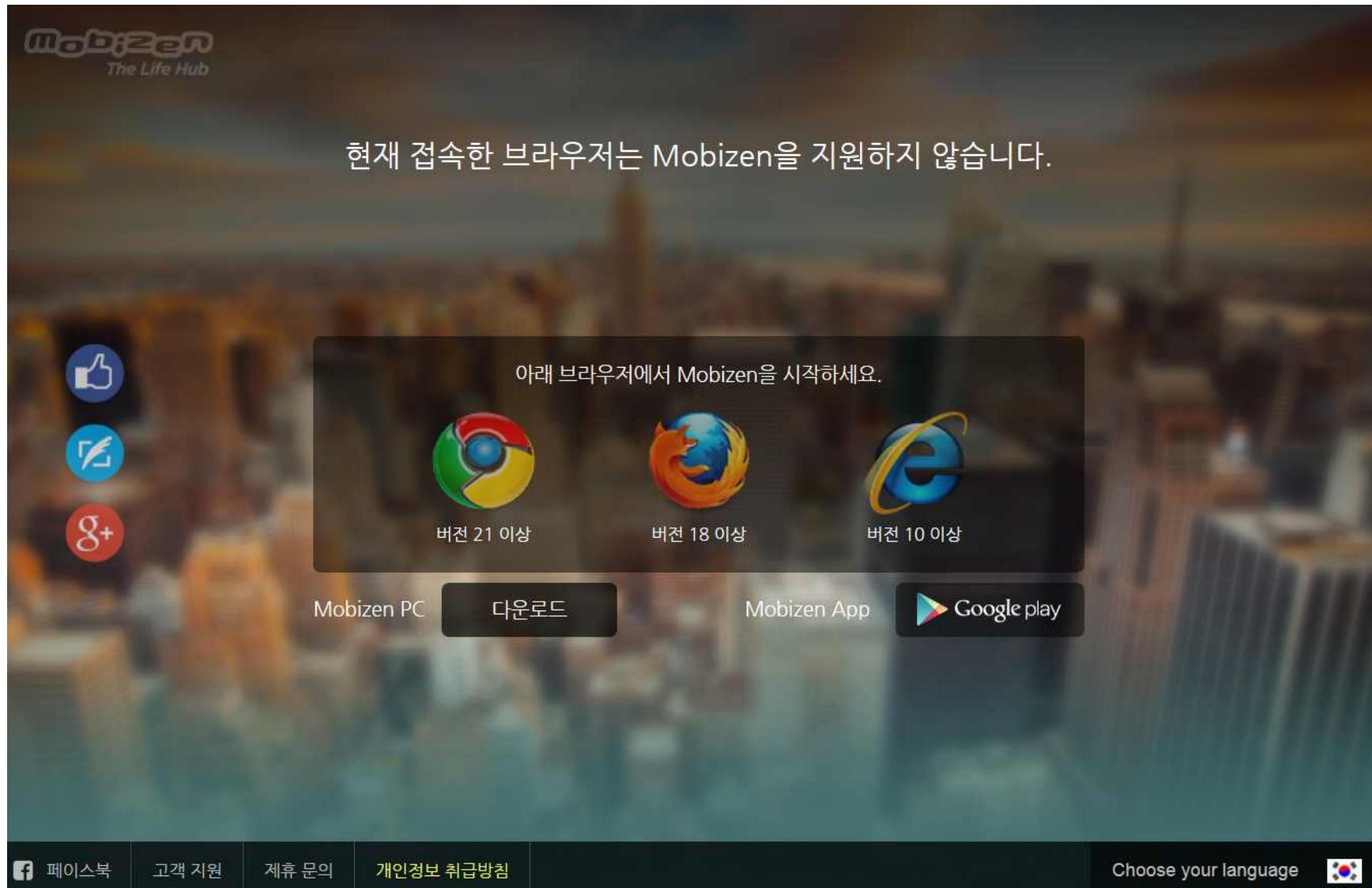
<https://developer.android.com/sdk/installing/studio-tips.html>



Mobizen: Search for Mobizen

The screenshot shows the NAVER search interface. The search bar at the top contains the query "mobizen". Below the search bar is a navigation menu with links to various categories like 통합검색, 블로그, 이미지, etc. Further down are filters for 정렬, 기간, 영역, and 옵션유지 (with options '꺼짐' and '켜짐'). A sidebar on the left lists related search terms under "연관검색어". The main content area displays search results for "사이트" (websites). The first result is "Mobizen" with the URL www.mobizen.pe.kr, described as "모바일 컨텐츠 정보, 리뷰 수록.". The second result is "모비즌" with the URL www.mobizen.com, described as "신청자 작성 고객지원", with a note about "라이프 허브 모비즌, 퍼스널 허브, 홈 클라우드,". The third result is "Mobizen" with the URL www.mobizen.fr, described as "렌터카 전문업체, 이용방법, 동영상, 요금 등 안내, 프랑스어 사이트.". At the bottom right of the search results, there is a link "사이트 더보기 >".

Mobizen: Unsupported Web Browser



Mobizen: Search for Chrome

The screenshot shows the NAVER search interface. The search bar at the top contains the query "chrome". Below the search bar is a navigation menu with links to various search categories: 통합검색 (Unified Search), 어학사전 (Dictionary), 이미지 (Images), 블로그 (Blogs), 지식iN (Knowledge iN), 지식백과 (Encyclopedia), 실시간검색 (Real-time Search), 뉴스 (News), and 더보기 (More). Underneath the menu are filters for 정렬 (Sort), 기간 (Date), 영역 (Region), 옵션유지 (Keep Options), 꺼짐 (Off) (selected), 켜짐 (On), and 상세검색 (Advanced Search). A section titled "연관검색어" (Related Searches) lists terms like "한층 개선된 구글크롬 다운로드", "한층 업그레이드 된 chrome", "한층 개선된 chrome", "chrome 다운로드", "한층 개선된 chrome의 최신버전...", "핸드폰 해킹", and "chrome 삭제", with "더보기" (More) and "신고" (Report) buttons. The main content area is titled "사이트" (Site) and features a card for "구글 크롬" (Google Chrome) with the URL www.google.com/chrome. The card includes links for 속도 (Speed), 간편함 (Convenience), 보안 (Security), 맞춤설정 (Customization), 개인정보 보호 (Data Protection), and a descriptive text about the browser's features.

Mobizen: Downloading Chrome



브라우저

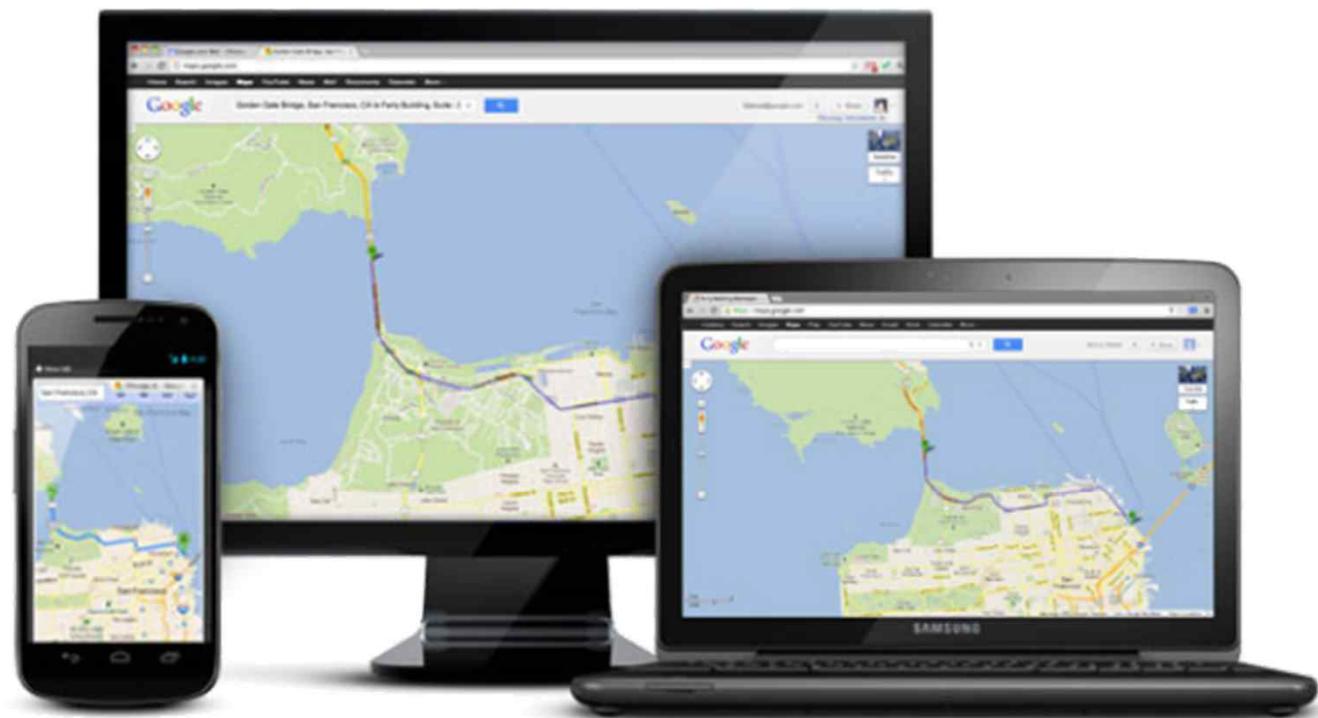
CHROMECAST

웹 스토어

Chrome 브라우저 탐색하기

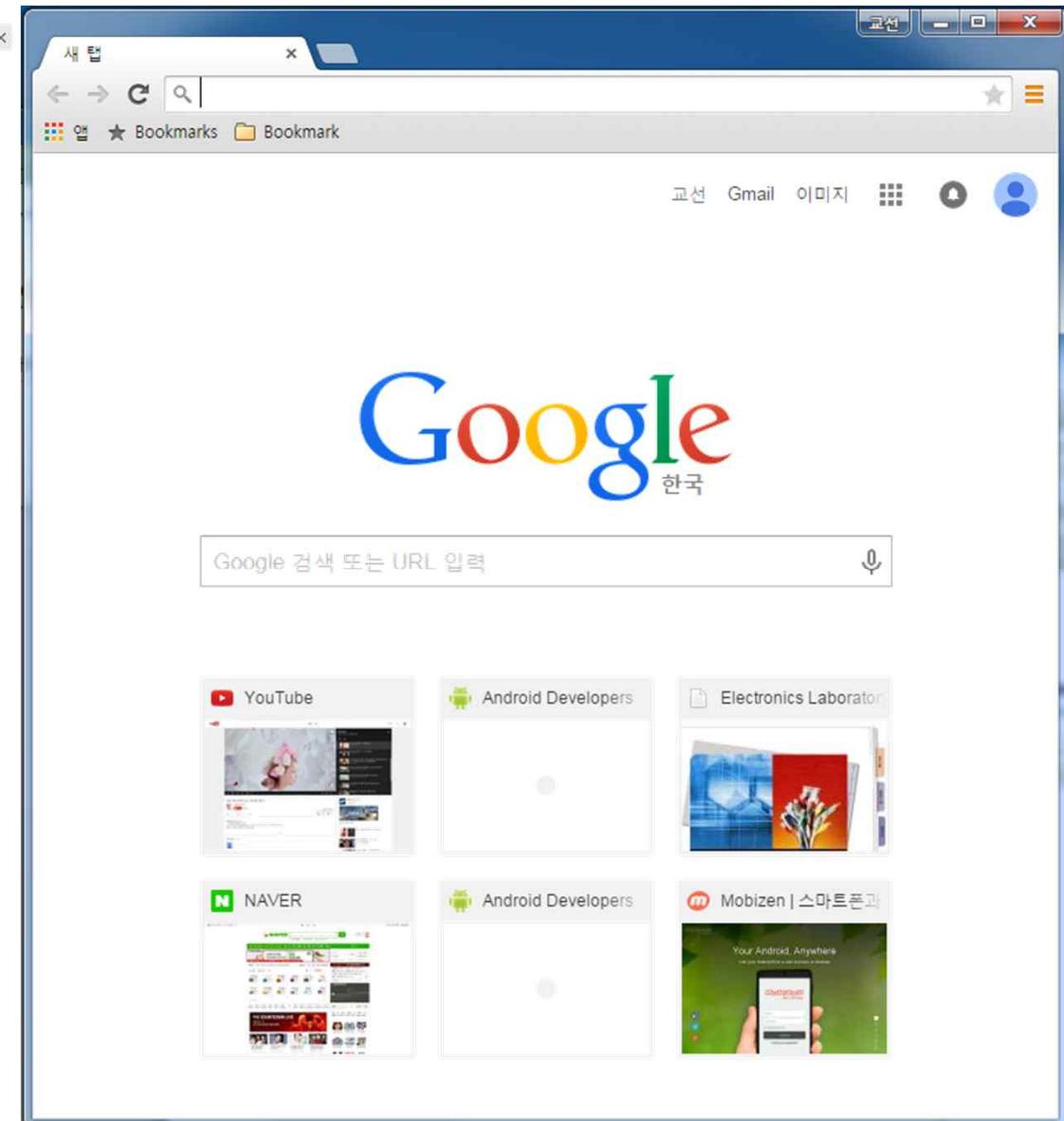
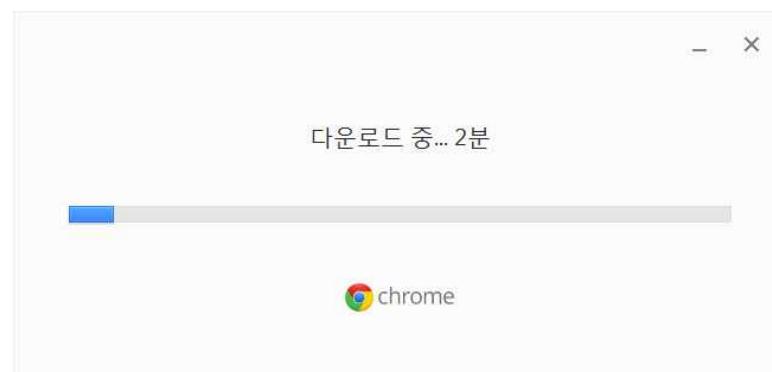
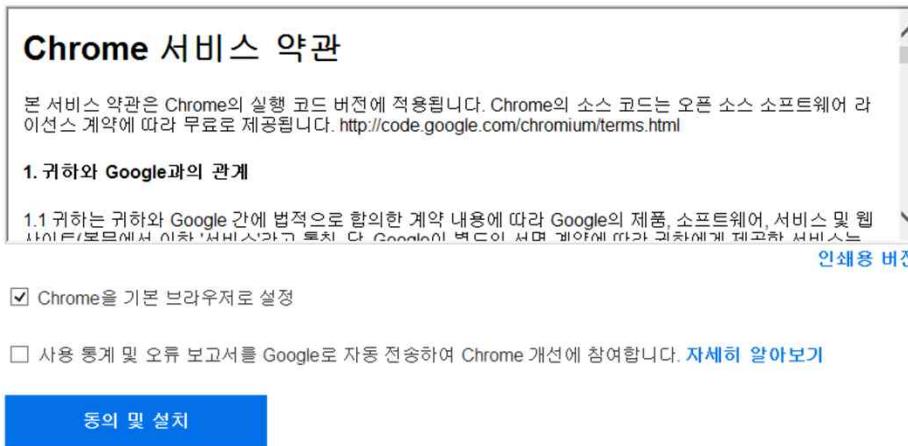
Chrome은 최신 웹에 최적화된 신속하고 간단하며 안전한 웹 브라우저입니다.

[Chrome 다운로드](#)



Mobizen: Downloading Chrome

차원이 다른 웹 서핑을 즐겨보세요.



Mobizen: Downloading Mobizen

The left screenshot shows a Google search results page for "mobizen". The search bar at the top contains "mobizen". Below it, there are several search filters: 웹문서 (Web), 동영상 (Video), 이미지 (Image), 지도 (Map), 앱 (App), 더보기 (More), and 검색 도구 (Search Tools). The main search results section shows a result for "파일전송은 플라잉파일 - flying-file.com" with a link to www.flying-file.com/. Below this, there are sections for "관련검색" (Related searches) and "바로가기" (Direct links). The right screenshot shows the official Mobizen website homepage at <https://mobizen.com>. The page features a large banner with the text "어디서나 화면을 연결하세요" (Connect your screen anywhere) and "모비즌, 당신의 스마트 라이프를 무한대로 확장합니다." (Mobizen, expand your smart life无限). It also features a large image of a hand holding a smartphone displaying the Mobizen app interface.

Android App Programming

August 4, 2015

Prof. Kyosun Kim
Incheon National University



Outline

1. Android Studio Installation
2. Tutorial I
3. Tutorial II
4. Bluetooth Chat
5. Infra-Red Remote Control V1
6. Infra-Red Remote Control V2

□ 참조: The C Programming Language

- ◆ <https://www.youtube.com/playlist?list=PLmngpJVWmdCwQueHIid7BSLJj0at7yEx9>

□ 참조: Atmega128 Micro-Controller

- ◆ https://www.youtube.com/watch?v=kvnc2L5pA78&list=PLmngpJVWmdCyMUuNNDRMIVcfghYj_YIq9

2. Tutorial I

February 20, 2015

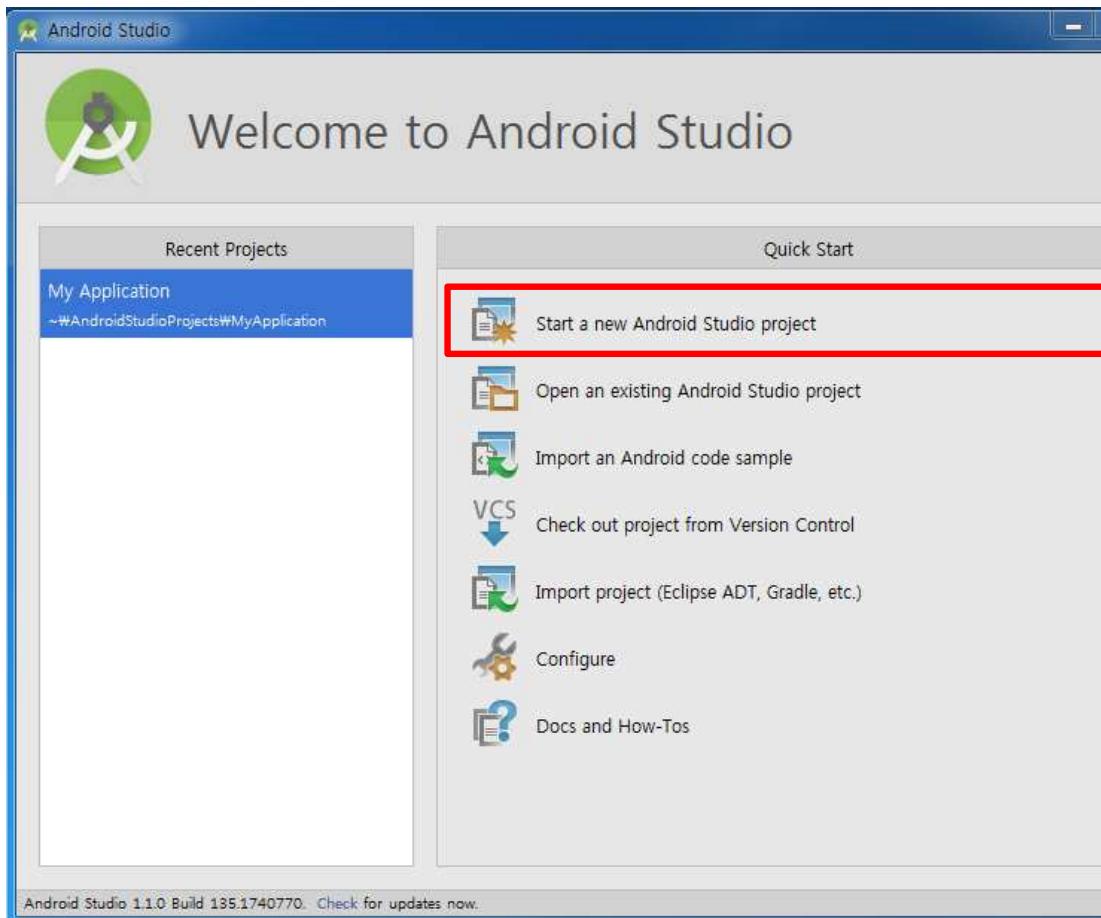
Prof. Kyosun Kim
Incheon National University

Outline

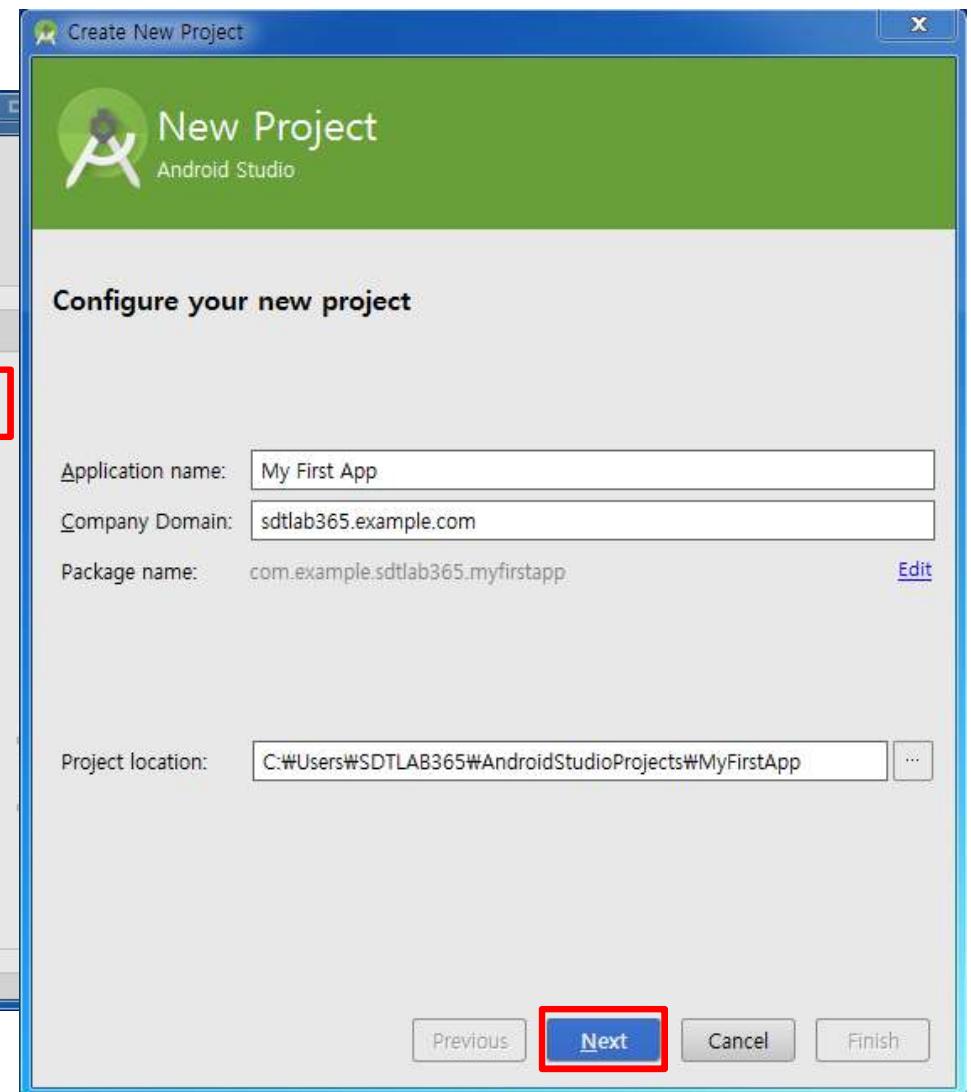
1. Creating an Android Project
2. Android Studio
3. Code (Java + XML)
4. Resources (XML)
5. Gradle Scripts (XML)
6. Running on a Device
7. Building a Simple User Interface

1. Creating an Android Project (1/3)

Start a new Android Studio project

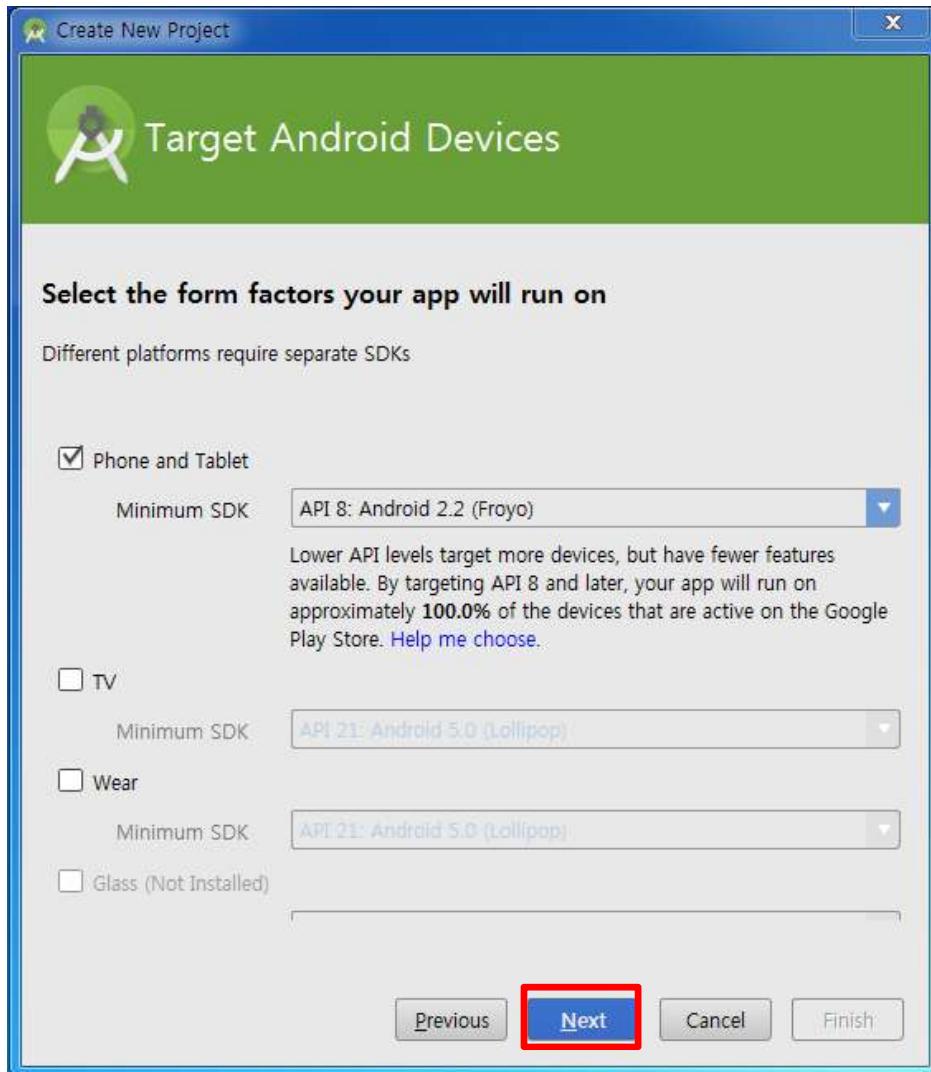


Configure your new project

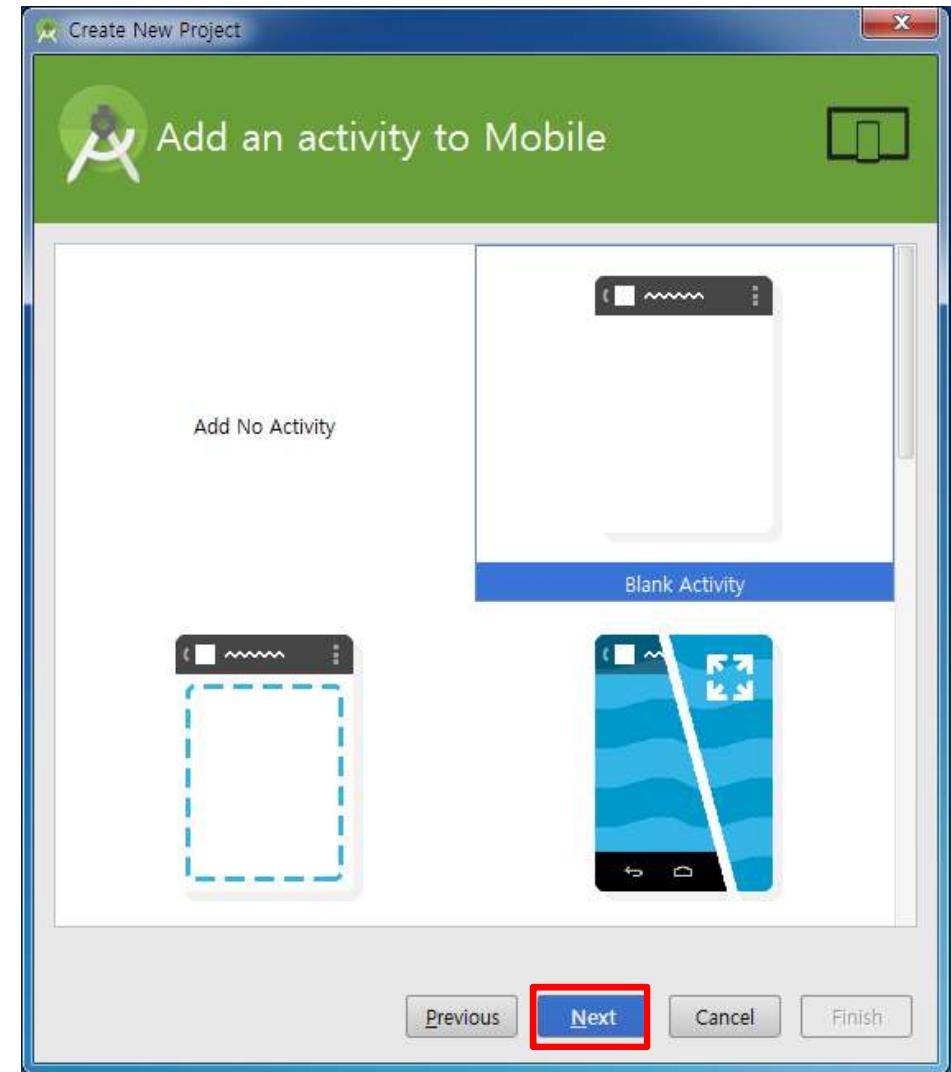


1. Creating an Android Project (2/3)

□ Target Android Devices

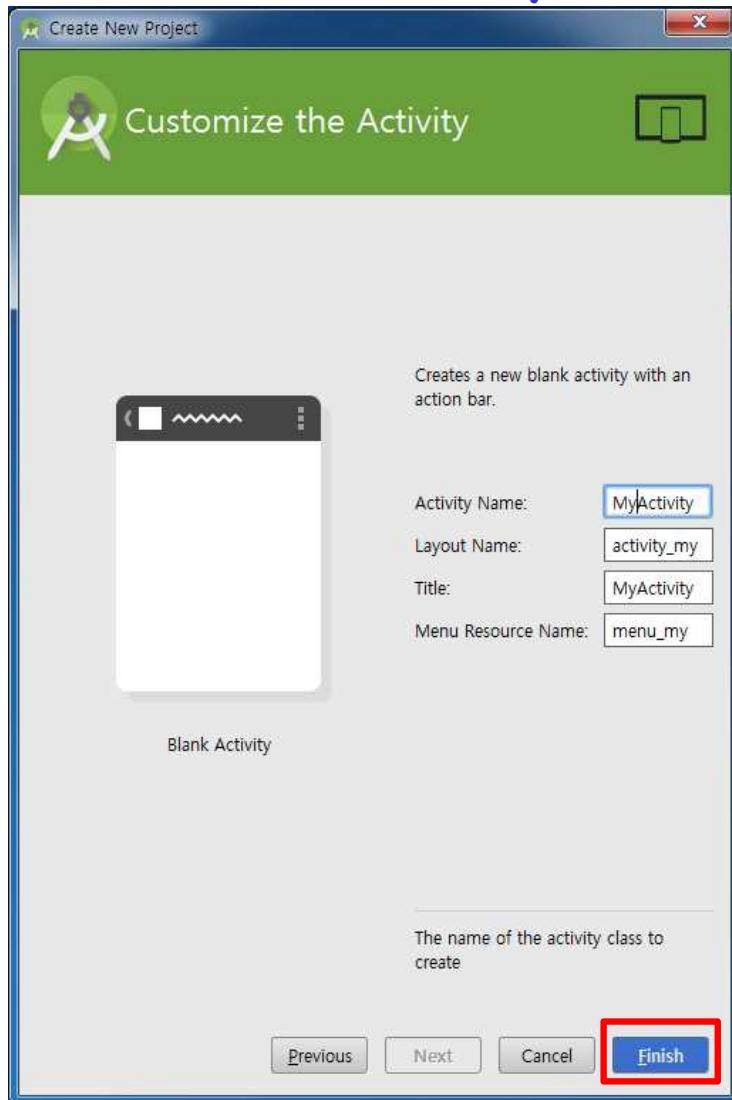


□ Add an activity to Mobile



1. Creating an Android Project (3/3)

□ Customize the Activity



□ Building project...



2. Android Studio

The screenshot shows the Android Studio interface for a project named "My Fisrt App". The left sidebar displays the project structure with files like AndroidManifest.xml, MyActivity.java, and activity_my.xml selected. The main area features a layout editor for "activity_my.xml" showing a smartphone preview with the text "Hello world!". The properties panel on the right lists attributes for the selected view, such as layout_width, layout_height, style, background, and gravity, all set to "match_parent". The bottom navigation bar includes tabs for Terminal, Android, Messages, TODO, Event Log, Gradle Console, and Memory Monitor.

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

MyFisrtApp app src main res layout activity_my.xml

Project 1: Structure 2: Favorites

Build Variants

Terminal Android Messages TODO

Event Log Gradle Console Memory Monitor

Gradle build finished in 9 sec (24 minutes ago)

My Activity.java activity_my.xml

Palette Nexus 4 AppTheme MyActivity 21 Component Tree Device Screen RelativeLayout TextView - @string/hello_world

Properties

layout:width	match_parent
layout:height	match_parent
style	accessibilityLiveRegion
background	alpha
clickable	backgroundTint
contentDescription	backgroundTintMode
elevation	clickable
focusable	contentDescription
focusableInTouchMode	elevation
gravity	focusable
id	focusableInTouchMode
ignoreGravity	gravity

3. Code (Java + XML)

The screenshot shows the Android Studio interface with three tabs open:

- MyActivity.java**: Contains the Java code for the activity.
- AndroidManifest.xml**: Contains the manifest file configuration.
- activity_my.xml**: Contains the XML layout for the activity.

Annotations highlight specific parts of the code:

- AndroidManifest.xml**:
 - Line 1: `<manifest ...>`
 - Line 2: `package="com.example.sdtlab365.myfisrtapp";`
 - Line 10: `<application ...>`
 - Line 11: `android:allowBackup="true"`
 - Line 12: `android:icon="@mipmap/ic_launcher"`
 - Line 13: `android:label="My First App"`
 - Line 14: `android:theme="@style/AppTheme"` (highlighted with a pink box)
 - Line 15: `<activity ...>`
 - Line 16: `android:name=".MyActivity"` (highlighted with a red box)
 - Line 17: `android:label="My First App"`
 - Line 18: `<intent-filter ...>`
 - Line 19: `<action android:name="android.intent.action.MAIN" />`
 - Line 20: `<category android:name="android.intent.category.LAUNCHER" />`
 - Line 21: `</intent-filter>`
 - Line 22: `</activity>`
 - Line 23: `</application>`
 - Line 24: `android:paddingRight="@dimen/activity_horizontal_margin"`
 - Line 25: `android:paddingTop="@dimen/activity_vertical_margin"`
 - Line 26: `</manifest>`- activity_my.xml**:
 - Line 1: `<RelativeLayout ...>`
 - Line 2: `tools:context=".MyActivity"` (highlighted with a red box)
 - Line 3: `<TextView ...>`
 - Line 4: `android:text="Hello world!"`
 - Line 5: `android:layout_width="wrap_content"`
 - Line 6: `android:layout_height="wrap_content" />`
 - Line 7: `android:text="@string/hello_world"` (highlighted with a pink box)
 - Line 8: `</RelativeLayout>`

The screenshot shows the Java code for `MyActivity` in the `MyActivity.java` file. Red boxes highlight specific parts of the code:

```
package com.example.sdtlab365.myfisrtapp;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MyActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu: this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_my, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

4. Resources (XML)

```
MyActivity.java x ApplicationTest.java x dimens.xml x string.xml  
<resources>  
    <!-- Default screen margins, per the Android Design guidelines. -->  
    <dimen name="activity_horizontal_margin">16dp</dimen>  
    <dimen name="activity_vertical_margin">16dp</dimen>  
</resources>
```

```
MyActivity.java x strings.xml x styles.xml x  
Edit translations for all locales in the translations editor.  
<resources>  
    <string name="app_name">My First App</string>  
  
    <string name="hello_world">Hello world!</string>  
    <string name="action_settings">Settings</string>  
</resources>
```

```
MyActivity.java x strings.xml x styles.xml x  
<resources>  
    <!-- Base application theme. -->  
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">  
        <!-- Customize your theme here. -->  
    </style>  
</resources>
```



```
menu_my.xml x  
<menu xmlns:android="http://schemas.android.com/apk/res/android"  
      xmlns:app="http://schemas.android.com/apk/res-auto"  
      xmlns:tools="http://schemas.android.com/tools" tools:context=".MyActivity">  
    <item android:id="@+id/action_settings"  
          android:title="Settings"  
          android:orderInCategory="10"  
          app:showAsAction="never" />  
</menu>
```

5. Gradle Scripts

project

```
// Top-level build file where you can add configuration options common to all sub-projects under this build.  
  
buildscript {  
    repositories {  
        jcenter()  
    }  
    dependencies {  
        classpath 'com.android.tools.build:gradle:1.1.0'  
  
        // NOTE: Do not place your application dependencies here; they belong  
        // in the individual module build.gradle files  
    }  
}  
  
allprojects {  
    repositories {  
        jcenter()  
    }  
}
```

module

```
apply plugin: 'com.android.application'  
  
android {  
    compileSdkVersion 21  
    buildToolsVersion "21.1.2"  
  
    defaultConfig {  
        applicationId "com.example.sdtlab365.myfisrtapp"  
        minSdkVersion 8  
        targetSdkVersion 21  
        versionCode 1  
        versionName "1.0"  
    }  
    buildTypes {  
        release {  
            minifyEnabled false  
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  
        }  
    }  
}  
  
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:21.0.3'  
}
```

6. Running on a Device

□ Installation of SAMSUNG USB Driver for Mobile Phones

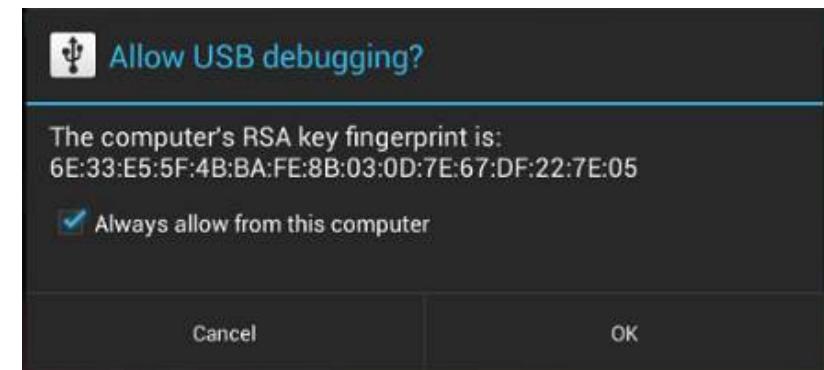
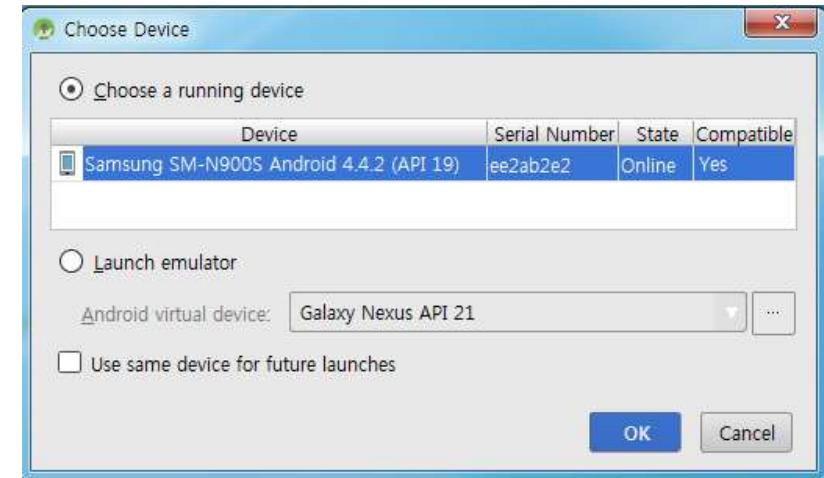
- ◆ http://local.sec.samsung.com/comLocal/support/down/kies_main.do?kind=usb

- ◆ Install the driver even if you installed Kies.

□ Run -> Run... -> App

□ If the device is listed as offline or unauthorized, go to the device display and check for the dialog shown below seeking permission to allow USB debugging.

- ◆ Try to unplug and plug the USB port to fix any problem.



7. Building a Simple User Interface (1/4)

The screenshot shows the Android Studio interface with three open files:

- MyActivity.java**: The Java code for the main activity.
- activity_my.xml**: The XML layout file for the main activity.
- AndroidManifest.xml**: The manifest file defining the application and its activities.

Annotations highlight specific parts of the code:

- A red box surrounds the class definition `public class MyActivity extends ActionBarActivity`.
- A red box surrounds the XML resource reference `R.layout.activity_my` in the `setContentView` call.
- A red arrow points from the `activity_my.xml` label to the `activity_my.xml` file tab.
- A red box surrounds the intent filter in the manifest entry for `MyActivity`.
- A red box surrounds the intent filter in the manifest entry for `DisplayMessageActivity`.
- A pink box surrounds the `sendMessage` method in `MyActivity.java`.
- A red arrow points from the `MyActivity.java` label to the `MyActivity.java` file tab.

```
MyActivity.java
```

```
package com.example.sdtlab365.myfirstapp;

import ...;

public class MyActivity extends ActionBarActivity {
    public final static String EXTRA_MESSAGE = "com.example.sdtlab365.myfirstapp.MESSAGE";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) { ... }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) { ... }
    private void openSettings() { }
    private void openSearch() { }
    public void sendMessage(View view) { ... }
}
```

```
activity_my.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sdtlab365.myfirstapp" >
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My First App"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MyActivity"
            android:label="My First App" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".DisplayMessageActivity"
            android:label="My Message"
            android:parentActivityName=".MyActivity" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.sdtlab365.myfirstapp.MyActivity" />
        </activity>
    </application>
</manifest>
```

```
AndroidManifest.xml
```

7. Building a Simple User Interface (1/4)

MyActivity.java

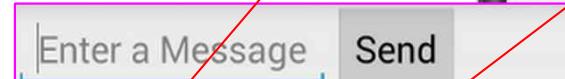
```
setContentView(R.layout.activity_my);
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />
</LinearLayout>
```

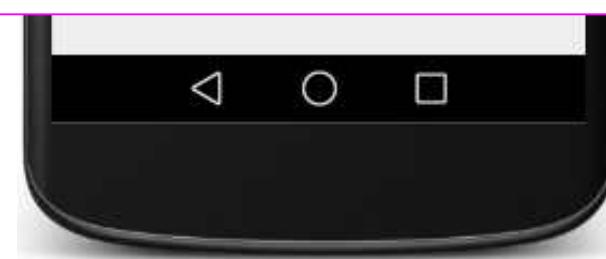
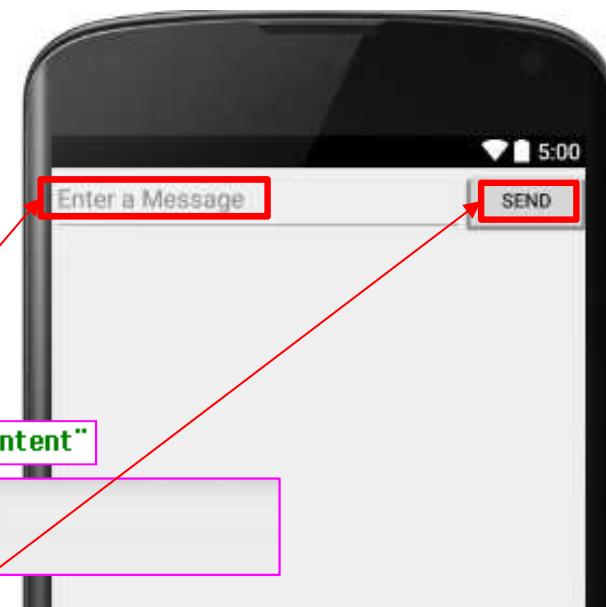
```
<resources>
    <string name="app_name">My First App</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="edit_message">Enter a Message</string>
    <string name="button_send">Send</string>
</resources>
```

// @ xml resource
// + new id

android:layout_width="wrap_content"



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:paddingLeft="16dp" android:paddingRight="16dp"
    android:paddingTop="16dp" android:paddingBottom="16dp">
    <TextView android:text="Hello world!">
        android:layout_width="wrap_content" android:layout_height="wrap_content" />
</RelativeLayout>
```



7. Building a Simple User Interface (2/4)

- **onClick attribute**
 - ◆ **sendMessage method**

The image shows an Android development environment with three tabs open: `activity_my.xml`, `DisplayMessageActivity.java`, and `MyActivity.java`.

activity_my.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="horizontal" >  
    <EditText android:id="@+id/edit_message"  
        android:layout_weight="1"  
        android:layout_width="0dp"  
        android:layout_height="wrap_content"  
        android:hint="Enter a Message" />  
    <Button  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Send"  
        android:onClick="sendMessage" />  
</LinearLayout>
```

DisplayMessageActivity.java:

```
package com.example.sdtlab365.myfirstapp;  
  
import ...  
  
public class DisplayMessageActivity extends ActionBarActivity {  
    public final static String EXTRA_MESSAGE = "com.example.sdtlab365.myfirstapp.MESSAGE";  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) { ... }  
  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) { ... }  
  
    @Override  
    public boolean onOptionsItemSelected(MenuItem item) { ... }  
  
    public void sendMessage(View view) {  
        Intent intent = new Intent(this, DisplayMessageActivity.class);  
        EditText editText = (EditText) findViewById(R.id.edit_message);  
        String message = editText.getText().toString();  
        intent.putExtra(EXTRA_MESSAGE, message);  
        startActivity(intent);  
    }  
}
```

MyActivity.java:

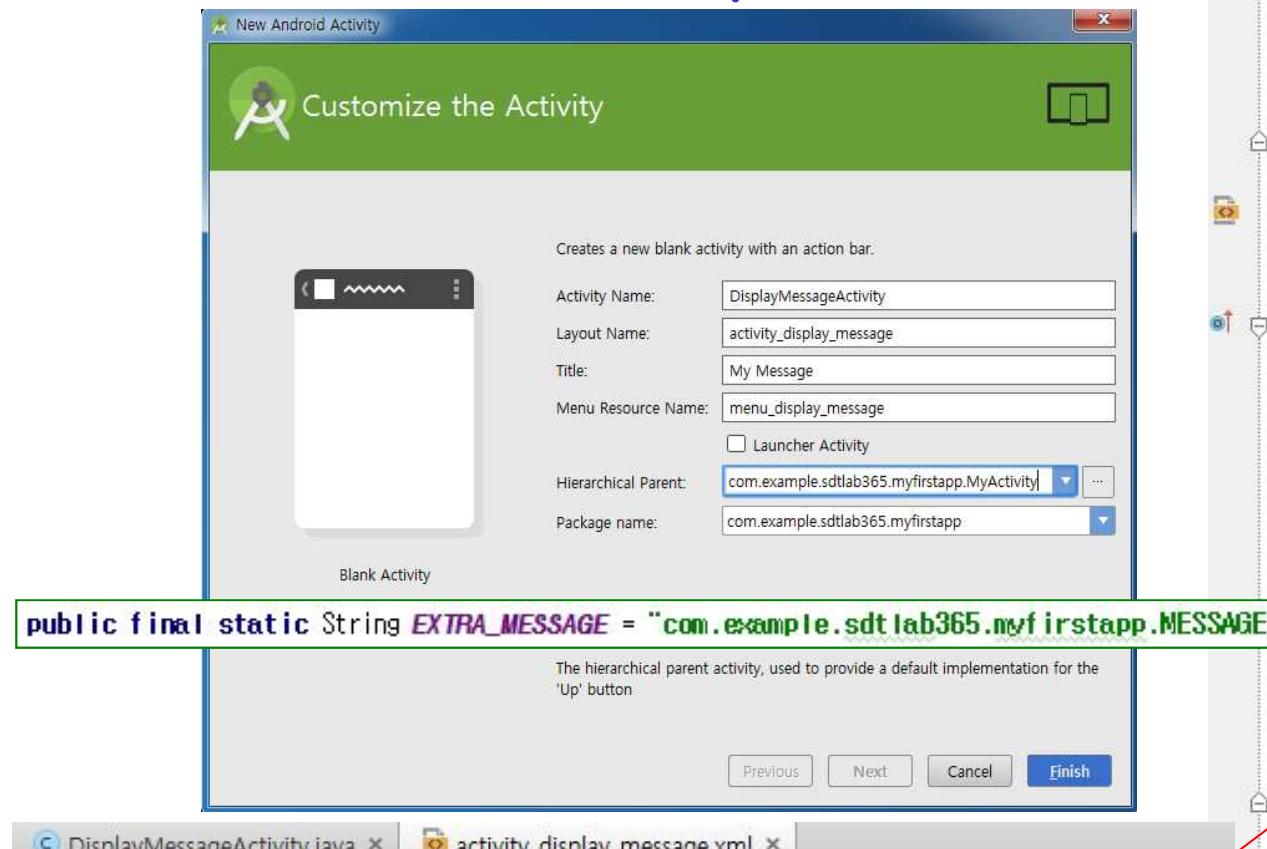
```
package com.example.sdtlab365.myfirstapp;  
  
import ...  
  
public class MyActivity extends ActionBarActivity {  
    public final static String EXTRA_MESSAGE = "com.example.sdtlab365.myfirstapp.MESSAGE";  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) { ... }  
  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) { ... }  
  
    @Override  
    public boolean onOptionsItemSelected(MenuItem item) { ... }  
  
    public void sendMessage(View view) {  
        Intent intent = new Intent(this, DisplayMessageActivity.class);  
        EditText editText = (EditText) findViewById(R.id.edit_message);  
        String message = editText.getText().toString();  
        intent.putExtra(EXTRA_MESSAGE, message);  
        startActivity(intent);  
    }  
}
```

Annotations in red:

- `editText.id` and `editText.onClick` are highlighted in red boxes.
- `EXTRA_MESSAGE` is highlighted in red boxes in both Java files.
- A large red box encloses the `sendMessage` method implementation in `DisplayMessageActivity.java`. Inside this box:
 - `Intent.putExtra` is highlighted with a red box.
 - `EXTRA_MESSAGE` is highlighted with a red box.
 - `startActivity` is highlighted with a red box.
 - The entire block is labeled `// content, class` to its right.
 - The `intent.putExtra` line is labeled `// key-value pair` to its right.

7. Building a Simple User Interface (2/4)

Create a new activity



The screenshot shows the XML layout file 'activity_display_message.xml'. It contains the following code:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"  
    android:layout_height="match_parent" android:paddingLeft="16dp"  
    android:paddingRight="16dp"  
    android:paddingTop="16dp"  
    android:paddingBottom="16dp"  
    tools:context="com.example.sdtlab365.myfirstapp.DisplayMessageActivity">  
    <TextView android:text="Hello world!" android:layout_width="wrap_content"  
        android:layout_height="wrap_content" />
```

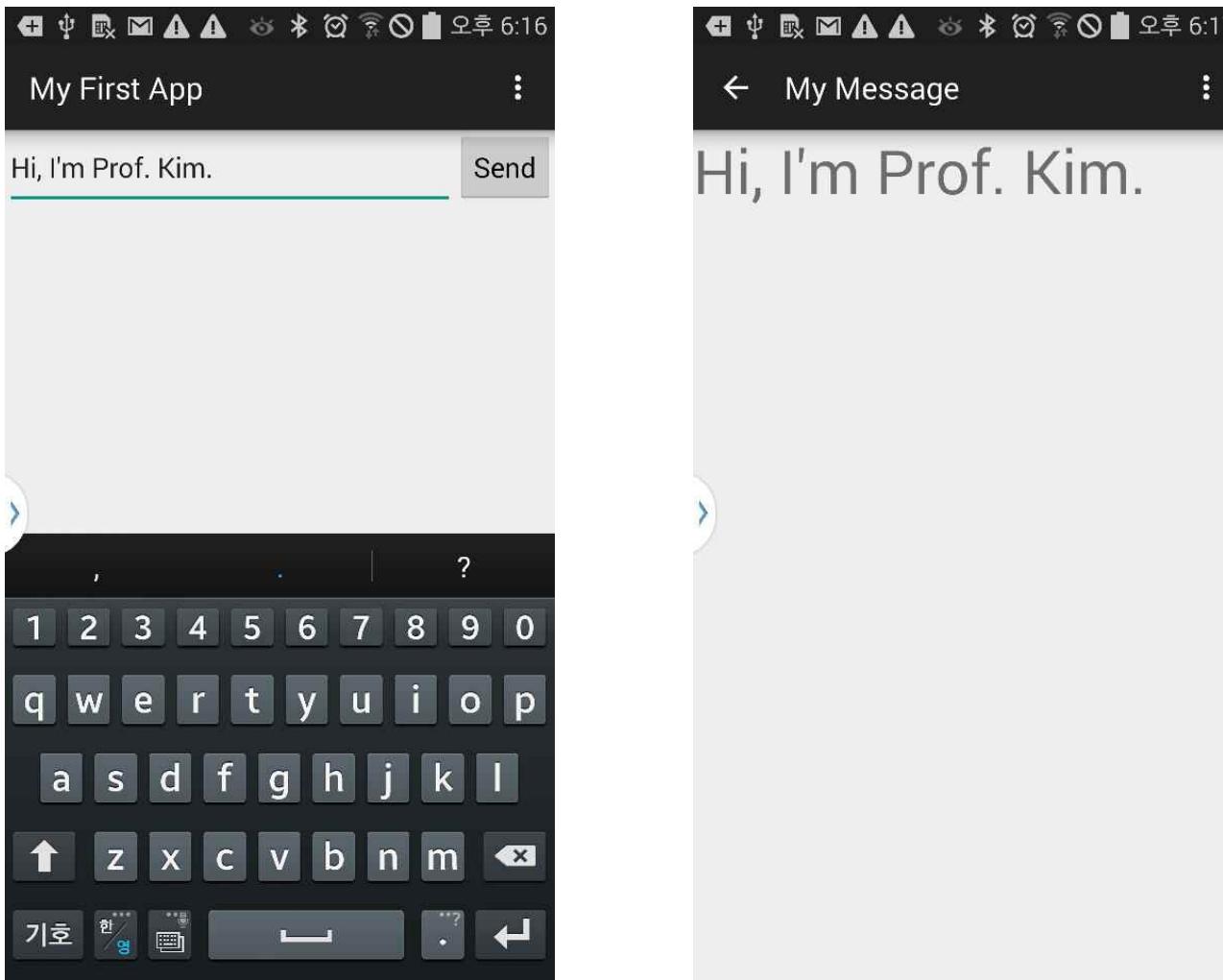
A large red 'X' mark is placed over the entire XML code area.

The screenshot shows the Java code for 'DisplayMessageActivity.java' with several annotations:

- The class definition is annotated: **DisplayMessageActivity** (highlighted in red) extends **ActionBarActivity**.
- The **onCreate** method is annotated: **onCreate** (highlighted in red) takes a **Bundle savedInstanceState** parameter.
- The intent retrieval code is annotated: **// Get the message from the intent**, **Intent intent = getIntent();**, and **String message = intent.getStringExtra(MyActivity.EXTRA_MESSAGE);** (highlighted in green).
- The text view creation code is annotated: **// Create the text view**, **TextView textView = new TextView(this);**, **textView.setTextSize(40);**, and **textView.setText(message);** (highlighted in red).
- The content view setting code is annotated: **setContentView(textView);** (highlighted in red).
- The placeholder fragment code is annotated: **R.layout.activity_display_message** (highlighted in red), followed by the **PlaceholderFragment** class definition.

7. Building a Simple User Interface (4/4)

□ Run -> Run... -> App



Android Tutorial I

May 13, 2016

Prof. Kyosun Kim
Incheon National University

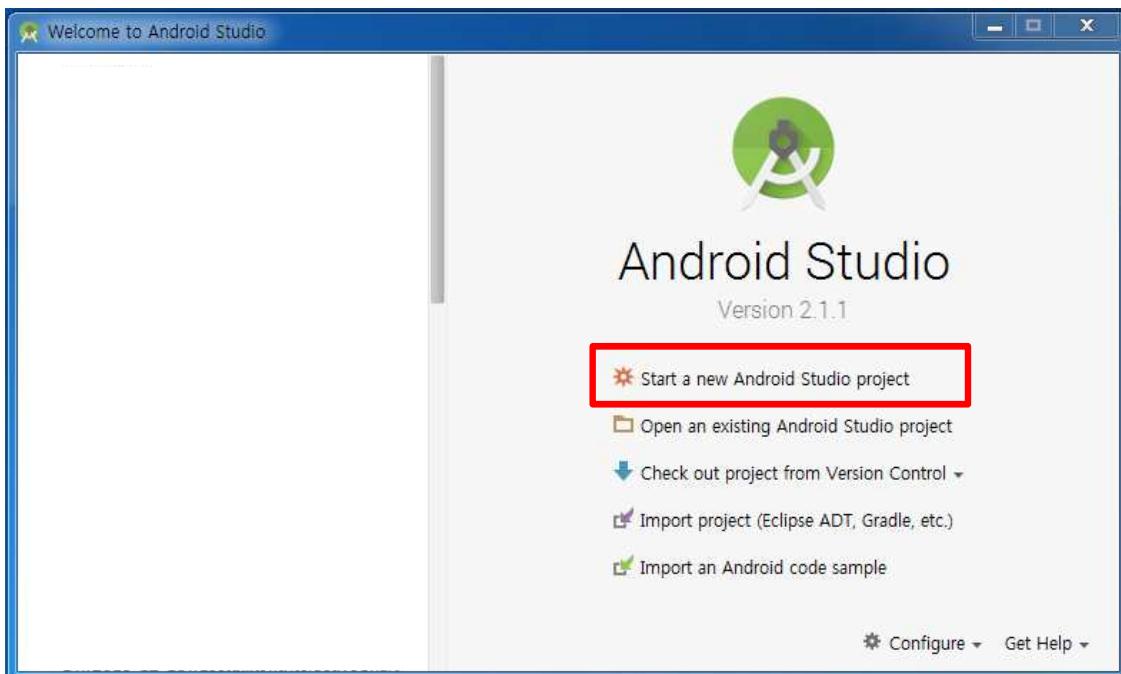


Outline

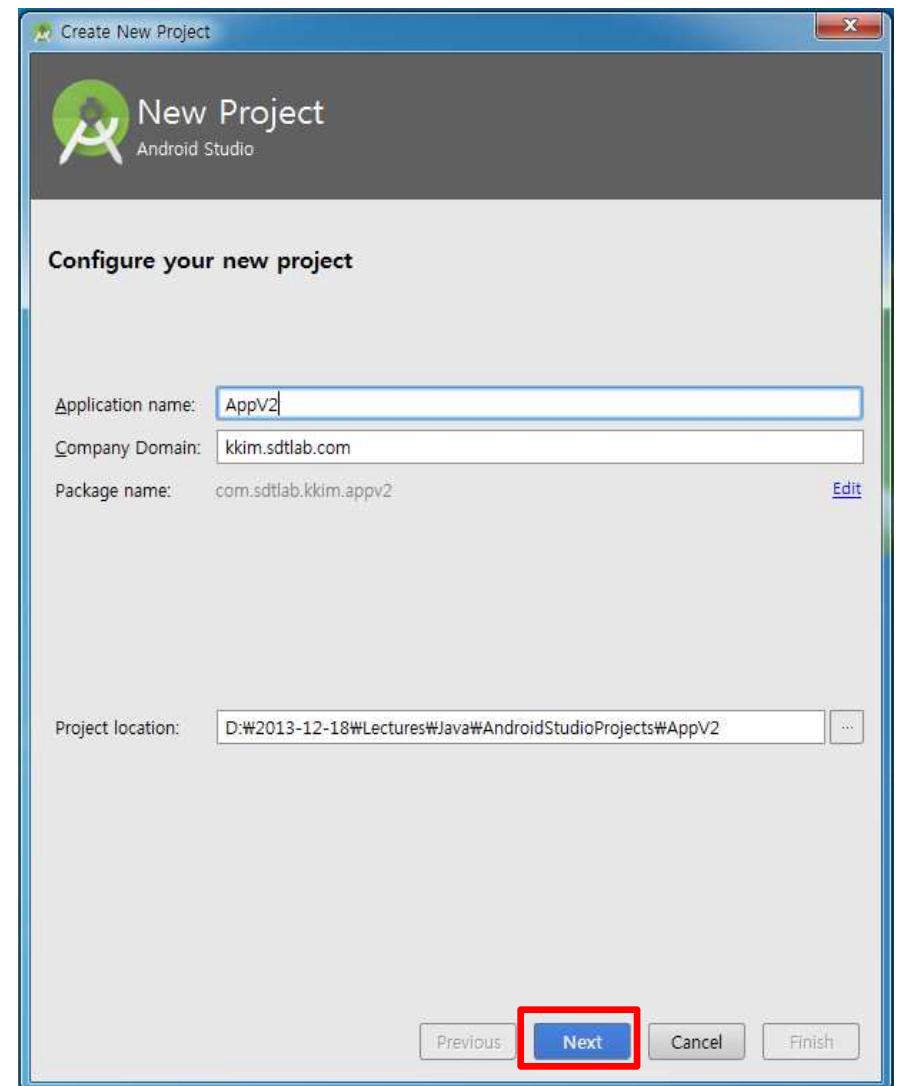
1. Creating an Android Project
2. Android Studio
3. Code (Java + XML)
4. Resources (XML)
5. Gradle Scripts (XML)
6. Running on a Device
7. Building a Simple User Interface

1. Creating an Android Project (1/3)

- Start a new Android Studio project

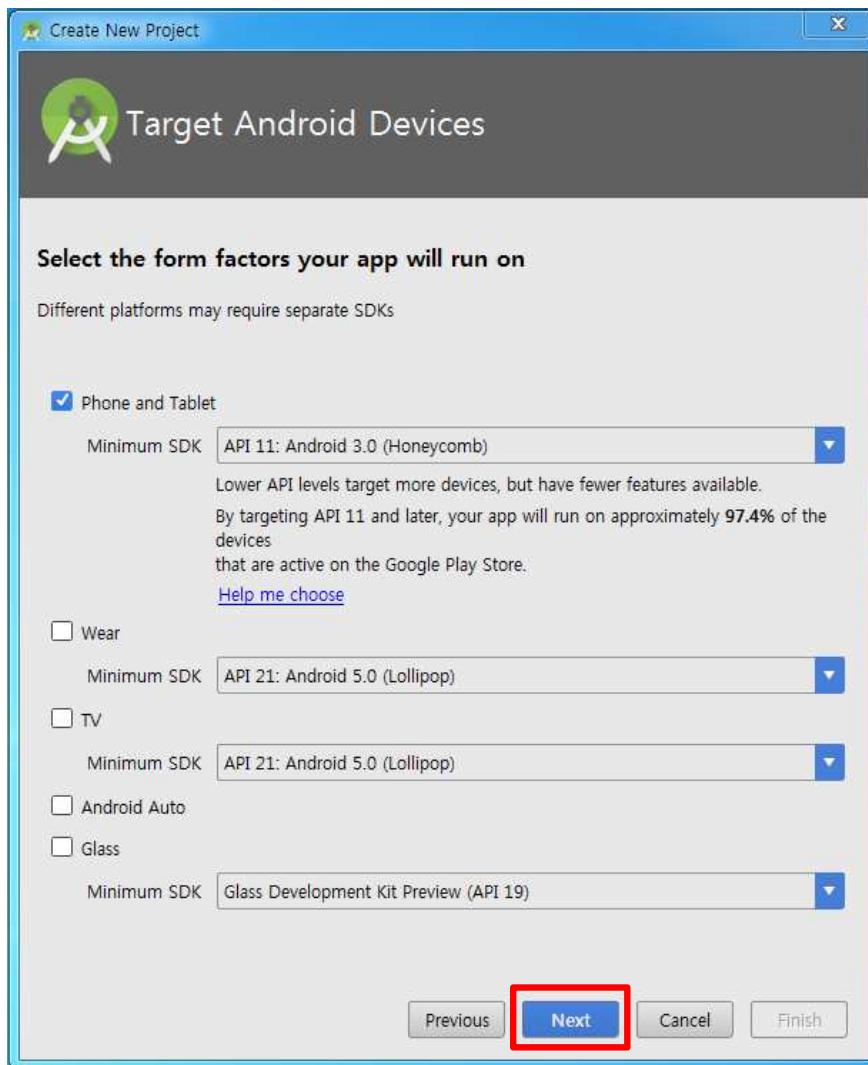


- Configure your new project

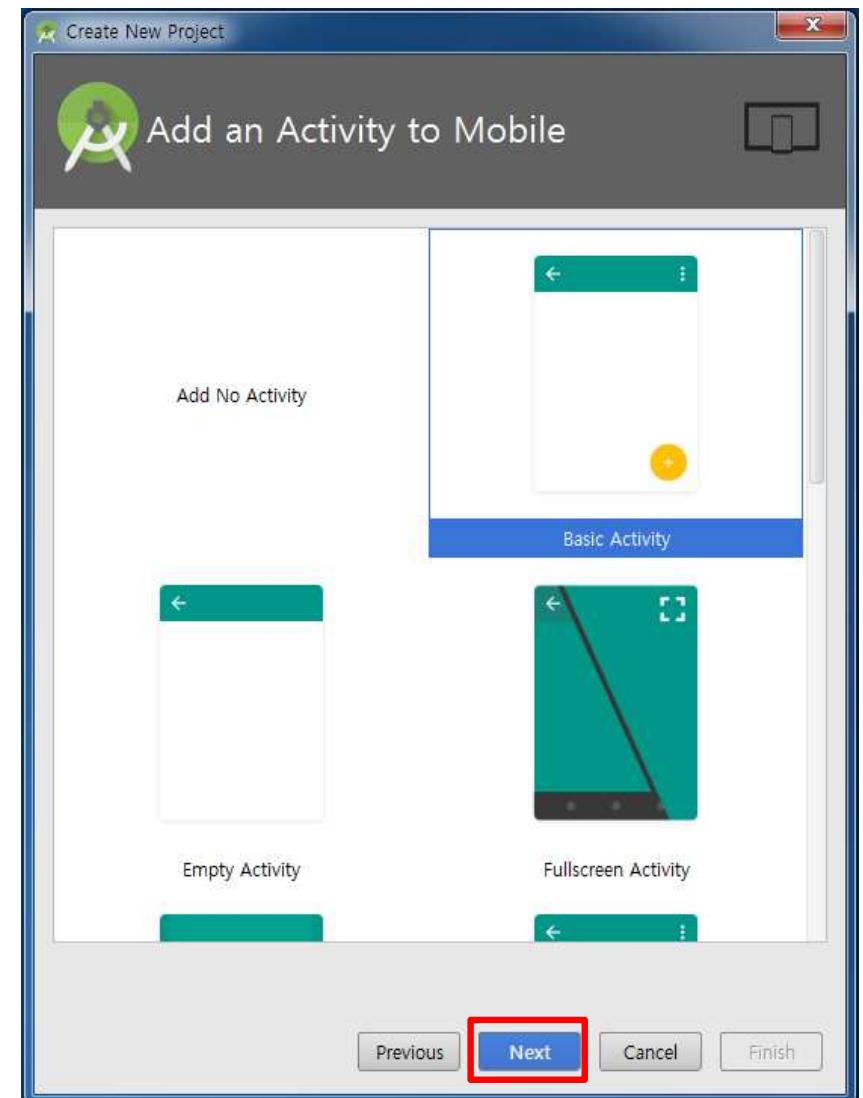


1. Creating an Android Project (2/3)

□ Target Android Devices

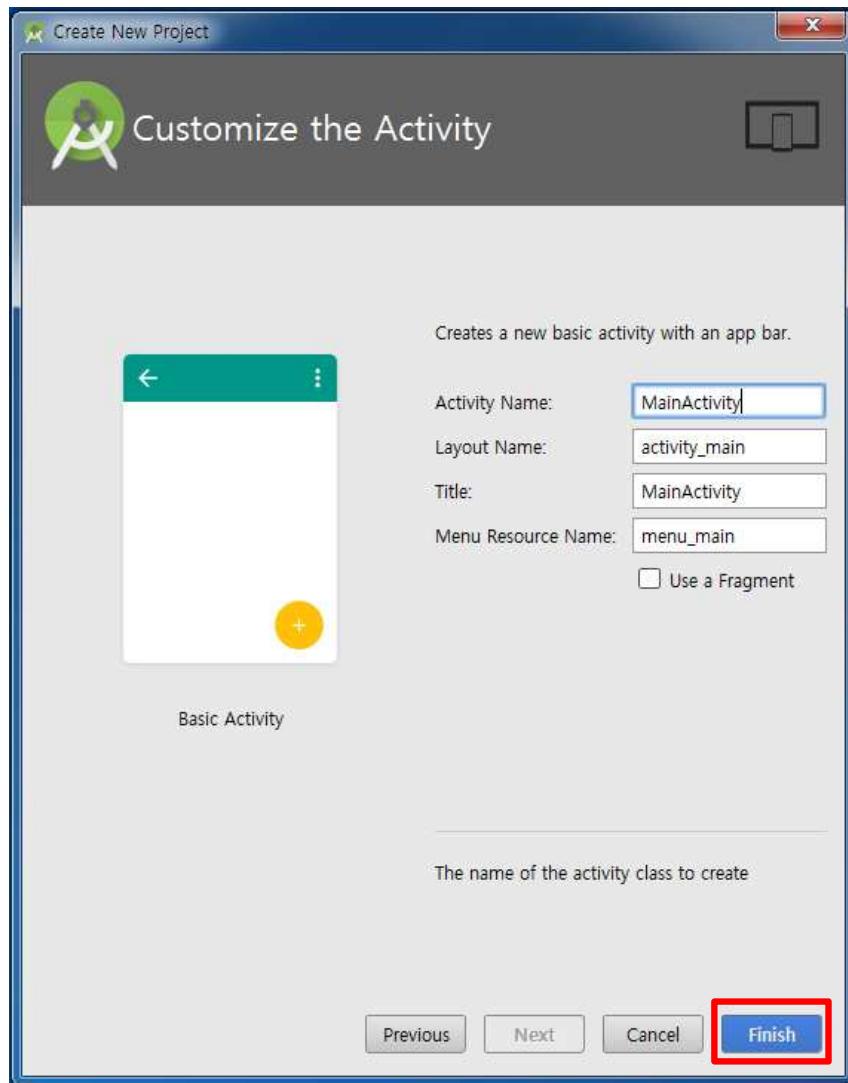


□ Add an activity to Mobile



1. Creating an Android Project (3/3)

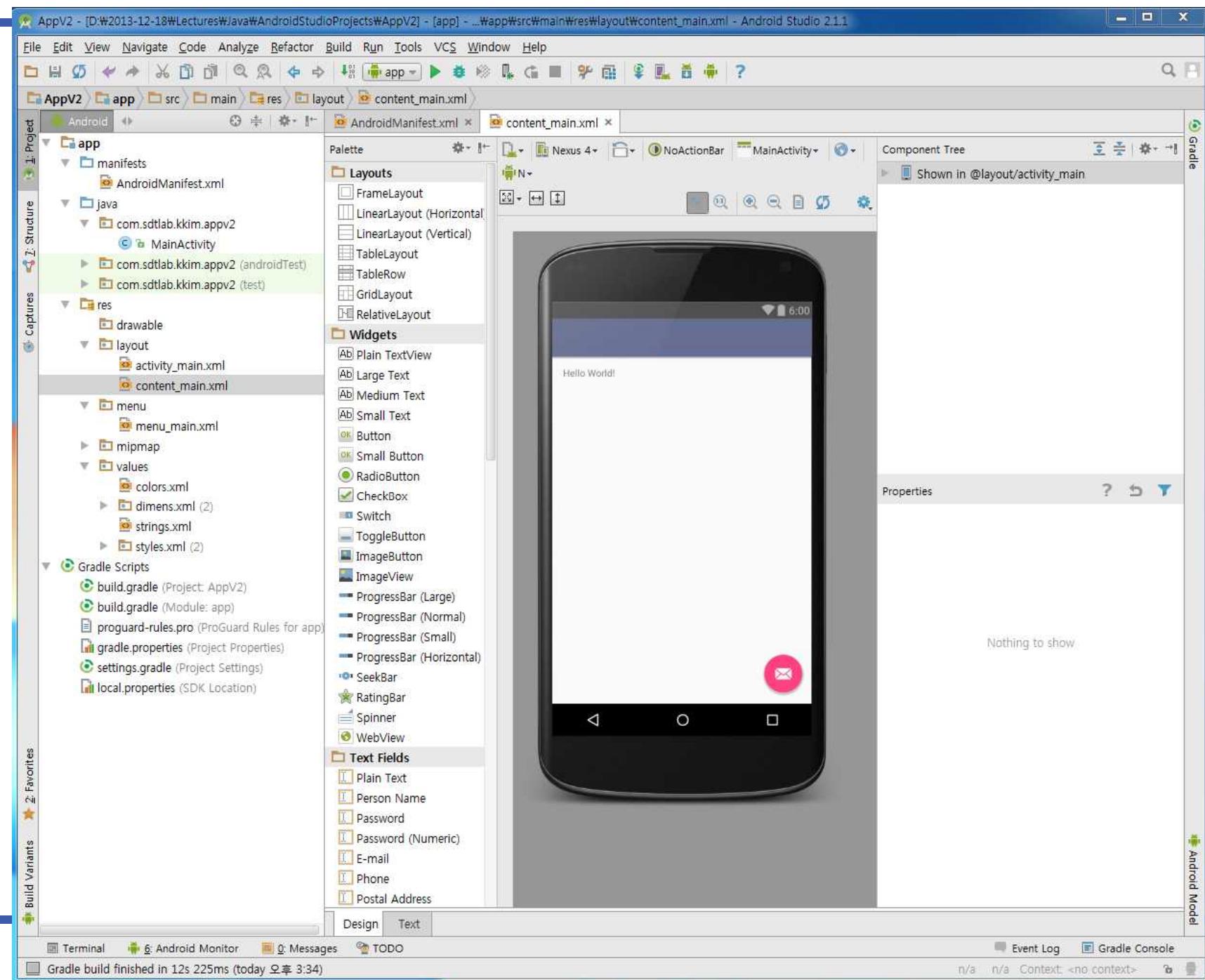
□ Customize the Activity



□ Building project...



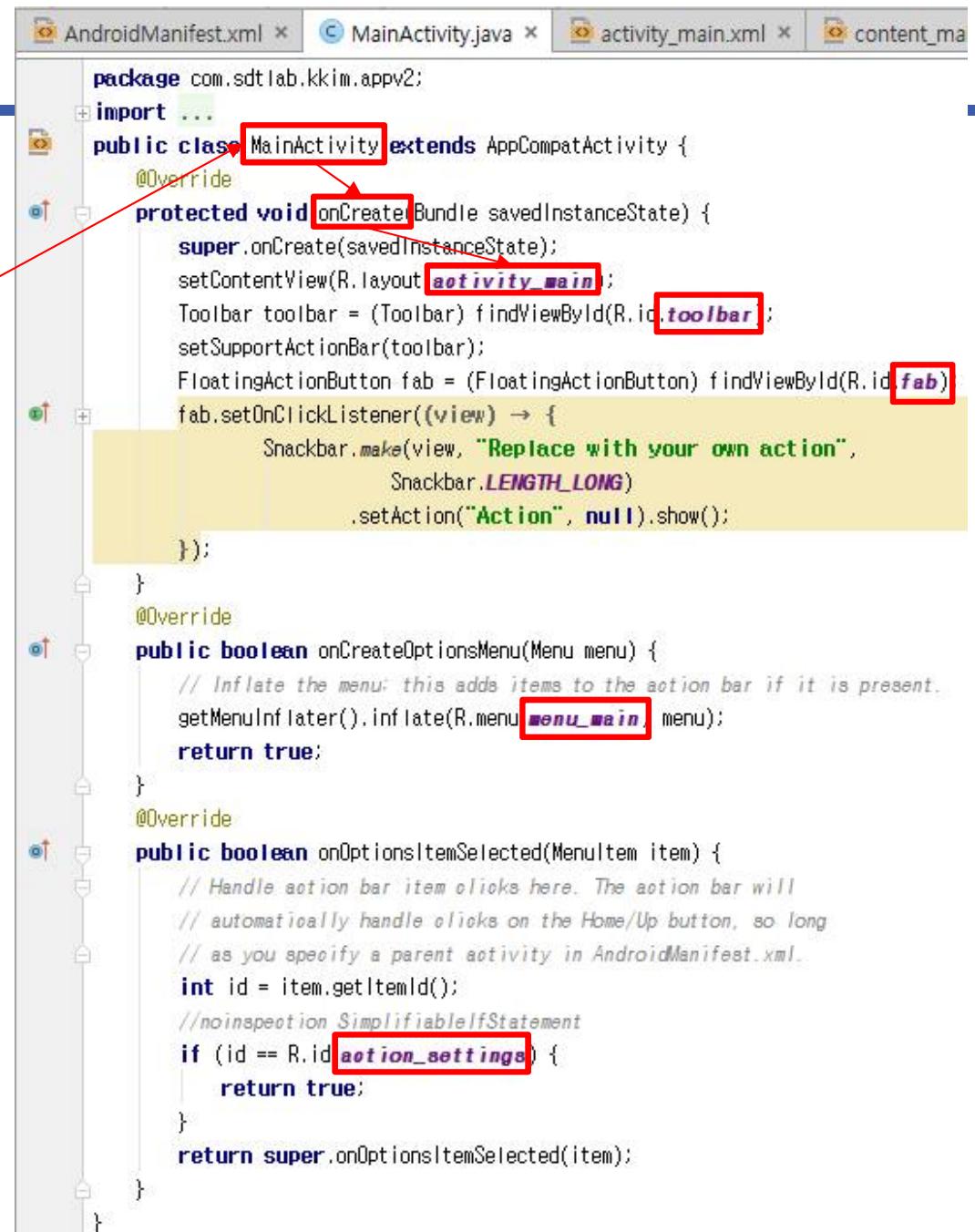
2. Android Studio



3. Code (Java + XML)



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.sdtlab.kkim.appv2">
    <application>
        <activity
            android:name=".MainActivivity"
            android:label="AppV2"
            android:theme="@style/AppTheme">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```



```
package com.sdtlab.kkim.appv2;
import ...
public class MainActivivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(view -> {
            Snackbar.make(view, "Replace with your own action",
                    Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
        });
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu: this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

3. Code (Java + XML)

```
package com.sdtlab.kkjm.appp2;
import ...
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(view) {
            Snackbar.make(view, "Replace with your own action",
                Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
        });
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context="com.sdtlab.kkjm.appp2.MainActivity">
    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">
        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />
    </android.support.design.widget.AppBarLayout>
    <include layout="@layout/content_main" />
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="16dp"
        android:src="@android:drawable/ic_dialog_email" />
</android.support.design.widget.CoordinatorLayout>
```

3. Code (Java + XML)

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context="com.sdtlab.kkim.appv2.MainActivity">
    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">
        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />
    </android.support.design.widget.AppBarLayout>
    <include layout="@layout/content_main" />
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="16dp"
        android:src="@android:drawable/ic_dialog_email" />
</android.support.design.widget.CoordinatorLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingViewBehavior"
    tools:context="com.sdtlab.kkim.appv2.MainActivity"
    tools:showIn="@layout/activity_main">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

3. Code (Java + XML)

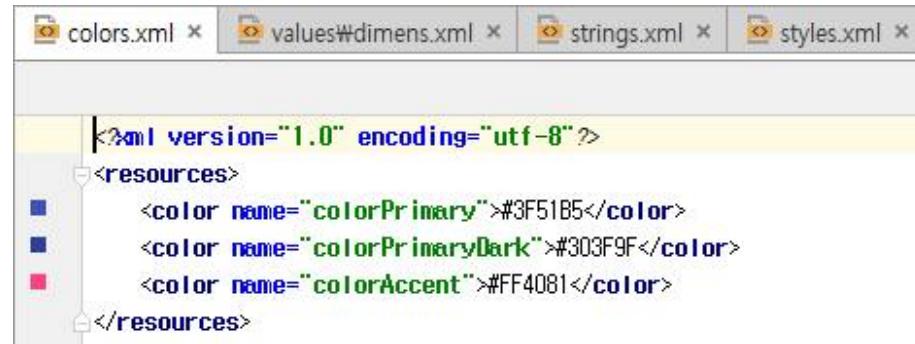
The screenshot shows the Android Studio interface with four tabs at the top: AndroidManifest.xml, MainActivity.java, activity_main.xml, and content_main.xml. The MainActivity.java tab is active, displaying Java code for a Main Activity. Several lines of code are highlighted with pink boxes and arrows pointing to specific identifiers in the code:

```
package com.sdtlab.kkim.appy2;
import ...
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(view) -> {
            Snackbar.make(view, "Replace with your own action",
                Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
        });
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

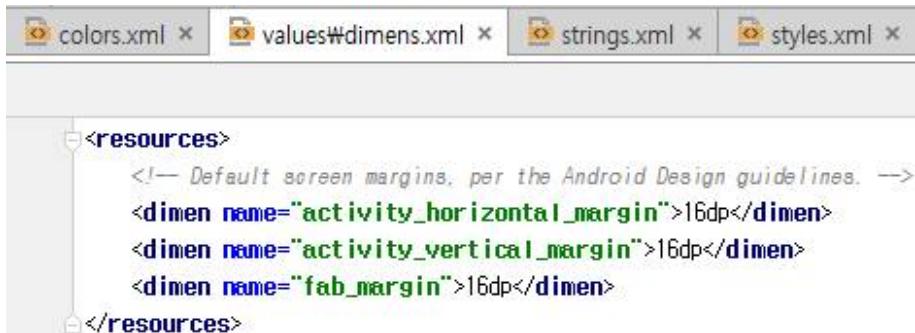
The screenshot shows the menu_main.xml file in the XML editor. A red box highlights the identifier 'action_settings' in the code, which corresponds to the identifier highlighted in the Java code above. Red arrows point from the highlighted code in MainActivity.java to the highlighted identifier in the XML file.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context="com.sdtlab.kkim.appy2.MainActivity">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="Settings"
        app:showAsAction="never" />
</menu>
```

4. Resources (XML)



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

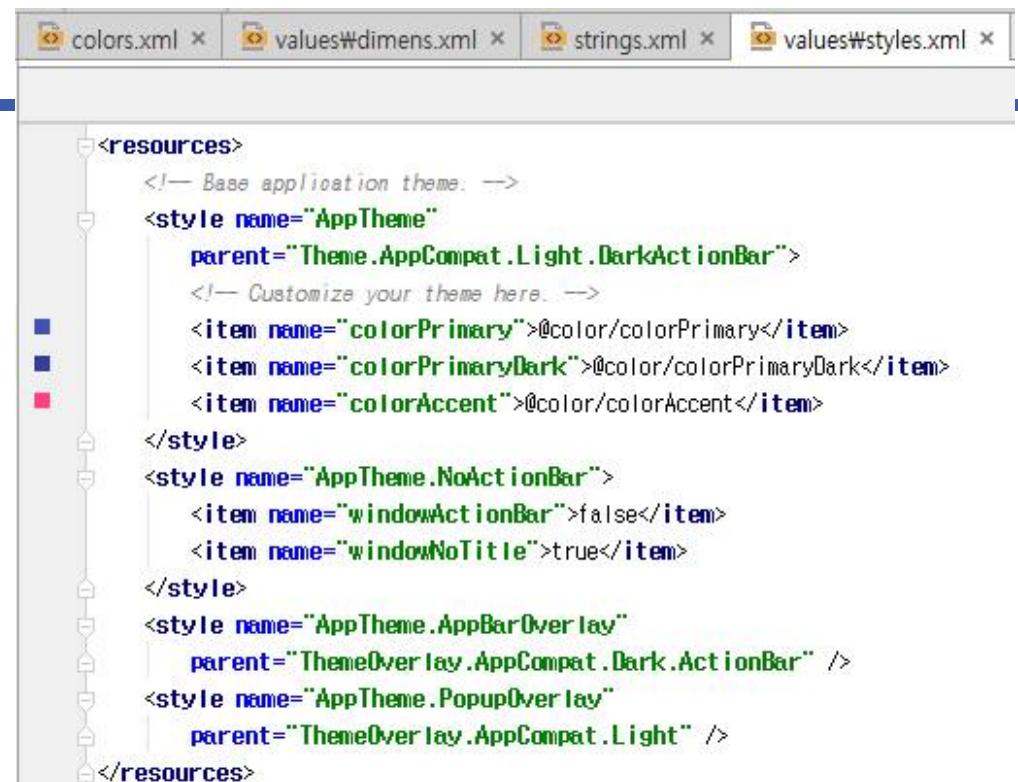


```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="fab_margin">16dp</dimen>
</resources>
```

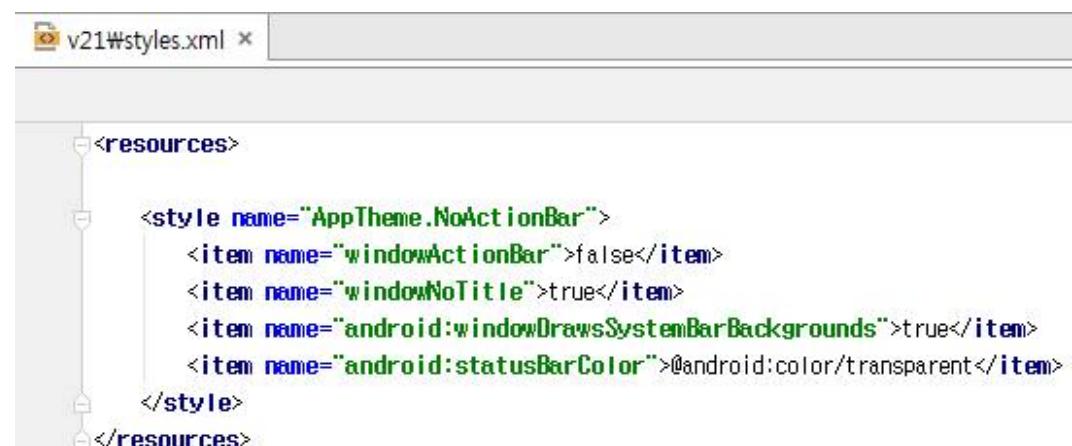


```
resources
Edit translations for all locales in the translations editor.

<resources>
    <string name="app_name">AppV2</string>
    <string name="action_settings">Settings</string>
</resources>
```

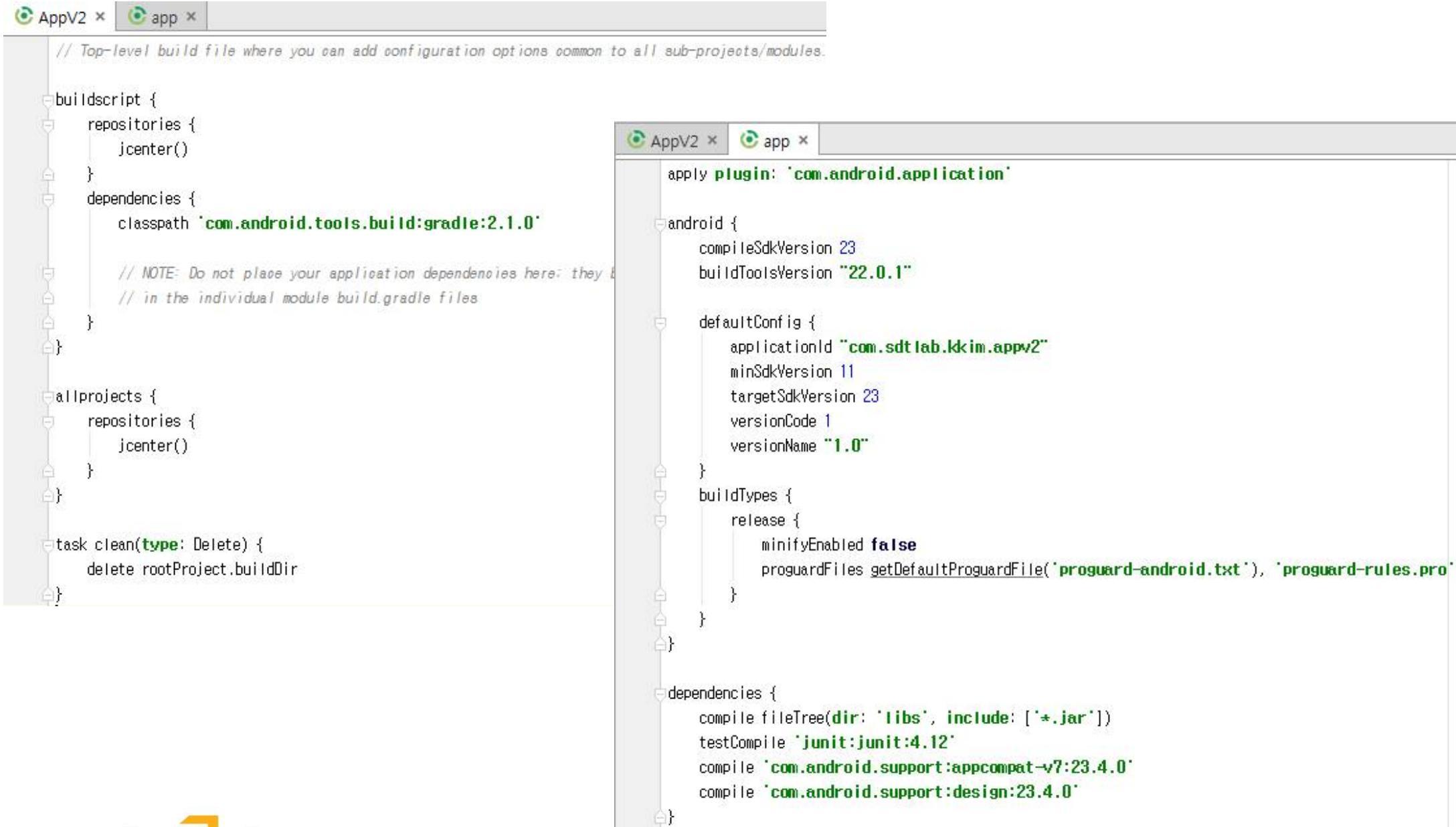


```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme">
        <parent>Theme.AppCompat.Light.DarkActionBar</parent>
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>
    <style name="AppTheme.AppBarOverlay">
        <parent>ThemeOverlay.AppCompat.Dark.ActionBar />
    </style>
    <style name="AppTheme.PopupOverlay">
        <parent>ThemeOverlay.AppCompat.Light />
    </style>
</resources>
```



```
v21/styles.xml
<resources>
    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
        <item name="android:windowDrawsSystemBarBackgrounds">true</item>
        <item name="android:statusBarColor">@android:color/transparent</item>
    </style>
</resources>
```

5. Gradle Scripts



```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.1.0'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion "22.0.1"

    defaultConfig {
        applicationId "com.sdtlab.kkim.appv2"
        minSdkVersion 11
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.4.0'
    compile 'com.android.support:design:23.4.0'
}
```

6. Running on a Device

□ Installation of SAMSUNG USB Driver for Mobile Phones

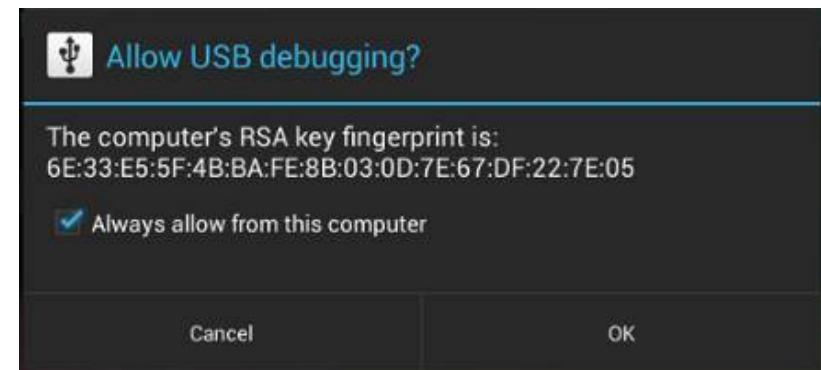
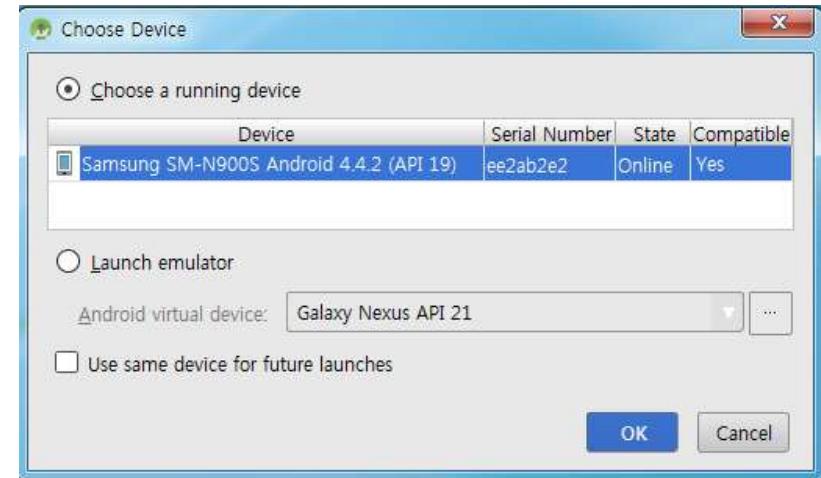
- ◆ http://local.sec.samsung.com/comLocal/support/down/kies_main.do?kind=usb

- ◆ Install the driver even if you installed Kies.

□ Run -> Run... -> App

□ If the device is listed as offline or unauthorized, go to the device display and check for the dialog shown below seeking permission to allow USB debugging.

- ◆ Try to unplug and plug the USB port to fix any problem.

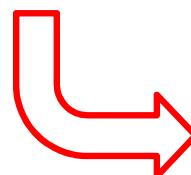


7. Building a Simple User Interface (1/9)

```
content_main.xml x
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".MainActivity"
    tools:showIn="@layout/activity_main">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

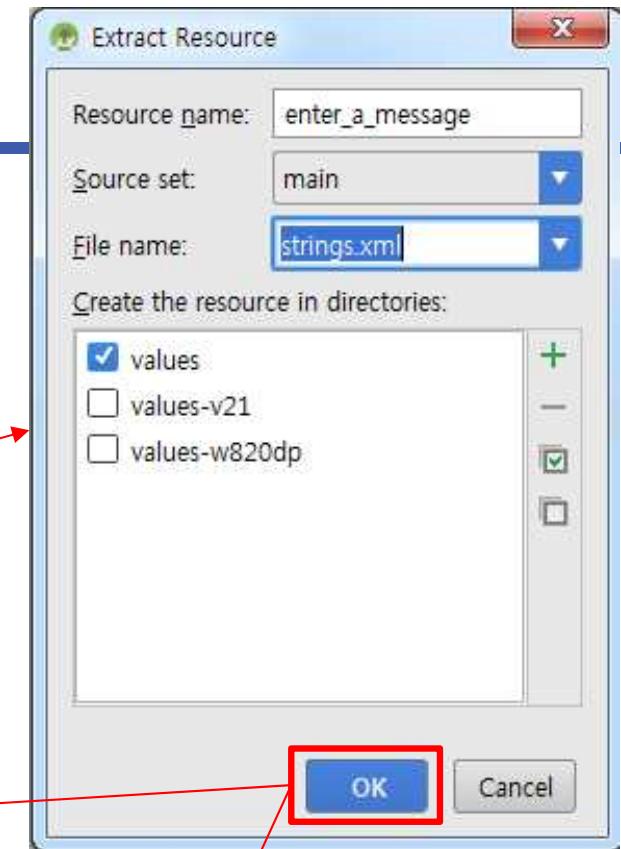


```
content_main.xml x
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".MainActivity"
    tools:showIn="@layout/activity_main">
</LinearLayout>
```

7. Building a Simple User Interface (2/9)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".MainActivity"
    tools:showIn="@layout/activity_main">
    <EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Enter a message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send" />
</LinearLayout>
```



```
<resources>
    <string name="app_name">AppV2</string>
    <string name="action_settings">Settings</string>
    <string name="enter_a_message">Enter a message</string>
    <string name="send">Send</string>
</resources>
```

7. Building a Simple User Interface

The image shows a composite screenshot of an Android development environment. On the left, the XML code editor displays the layout file `content_main.xml`. It contains a `LinearLayout` with a child `EditText` and a child `Button`. The `EditText` has attributes `android:layout_weight="1"`, `android:layout_width="0dp"`, and `android:layout_height="wrap_content"`. The `Button` has attributes `android:layout_width="wrap_content"`, `android:layout_height="wrap_content"`, `android:text="@string/send"`, and `android:onClick="sendMessage"`. A red circle highlights the `onClick` attribute of the button. On the right, the Java code editor shows the `MainActivity.java` file. It includes an `@Override` annotation and a `onOptionsItemSelected` method. Below it, a `sendMessage` method is defined. Red boxes highlight the `sendMessage` method and the `OK` button in a dialog box. The dialog box, titled "Choose Activity to Create the Method", lists "MainActivity (com.sdtlab.kkim.appv2)" and has "app" selected. The `OK` button is highlighted with a red box.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.sdtlab.kkim.appv2.MainActivity"
    tools:showIn="@layout/activity_main">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/enter_a_message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/send"
        android:onClick="sendMessage"/>
</LinearLayout>
```

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }
    return super.onOptionsItemSelected(item);
}

public void sendMessage(View view) { }
```

7. Building a Simple User Interface (4/9)

The screenshot shows two instances of the Android Studio code editor. The left instance displays the `MainActivity.java` file, and the right instance shows the same file with code completion suggestions overlaid.

MainActivity.java Content:

```
package com.sdtlab.kkim.appp2;
import ...
public class MainActivity extends AppCompatActivity {
    public final static String EXTRA_MESSAGE = "com.mycompany.myfirstapp.MESSAGE";
    @Override
    protected void onCreate(Bundle savedInstanceState) { ... }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) { ... }
    @Override
    ? android.content.Intent? Alt+Enter
        onOptionsItemSelected(MenuItem item) { ... }
        public void sendMessage(View view) {
            Intent intent = new Intent(this, DisplayMessageActivity.class);
            EditText editText = (EditText) findViewById(R.id.edit_message);
            String message = editText.getText().toString();
            intent.putExtra(EXTRA_MESSAGE, message);
            startActivity(intent);
        }
}
```

Code Completion Suggestion (Left):

- Import class
 - Create class 'Intent'
 - Create enum 'Intent'
 - Create inner class 'Intent'
 - Create interface 'Intent'
- Insert App Indexing API Code
- Split into declaration and assignment

Code Completion Suggestion (Right):

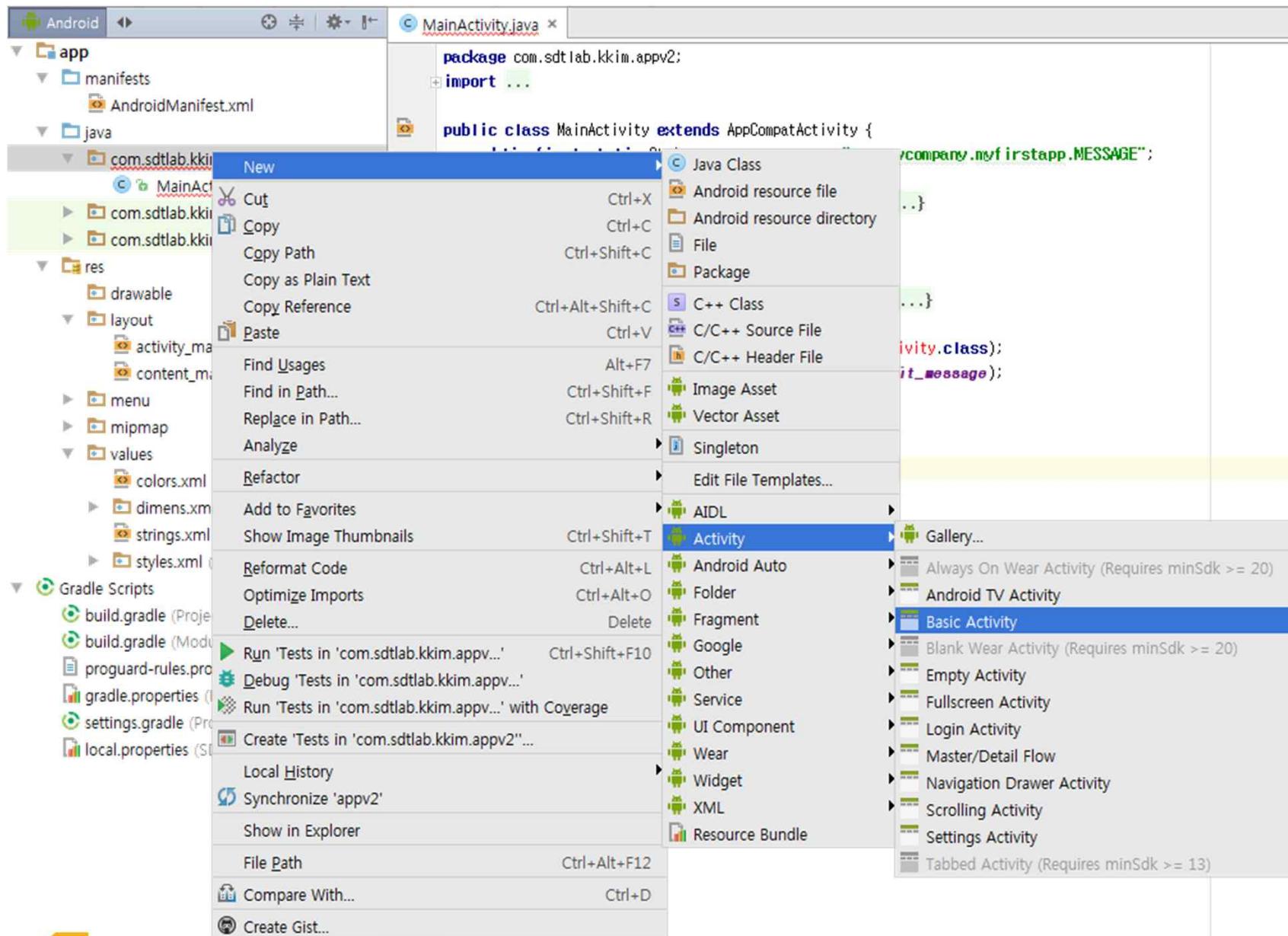
- Import android.content.Intent;
- Import android.os.Bundle;
- Import android.support.design.widget.FloatingActionButton;
- Import android.support.design.widget.Snackbar;
- Import android.support.v7.app.AppCompatActivity;
- Import android.support.v7.widget.Toolbar;
- Import android.view.View;
- Import android.view.Menu;
- Import android.view.MenuItem;
- Import android.widget.EditText;

MainActivity.java Content (Completed):

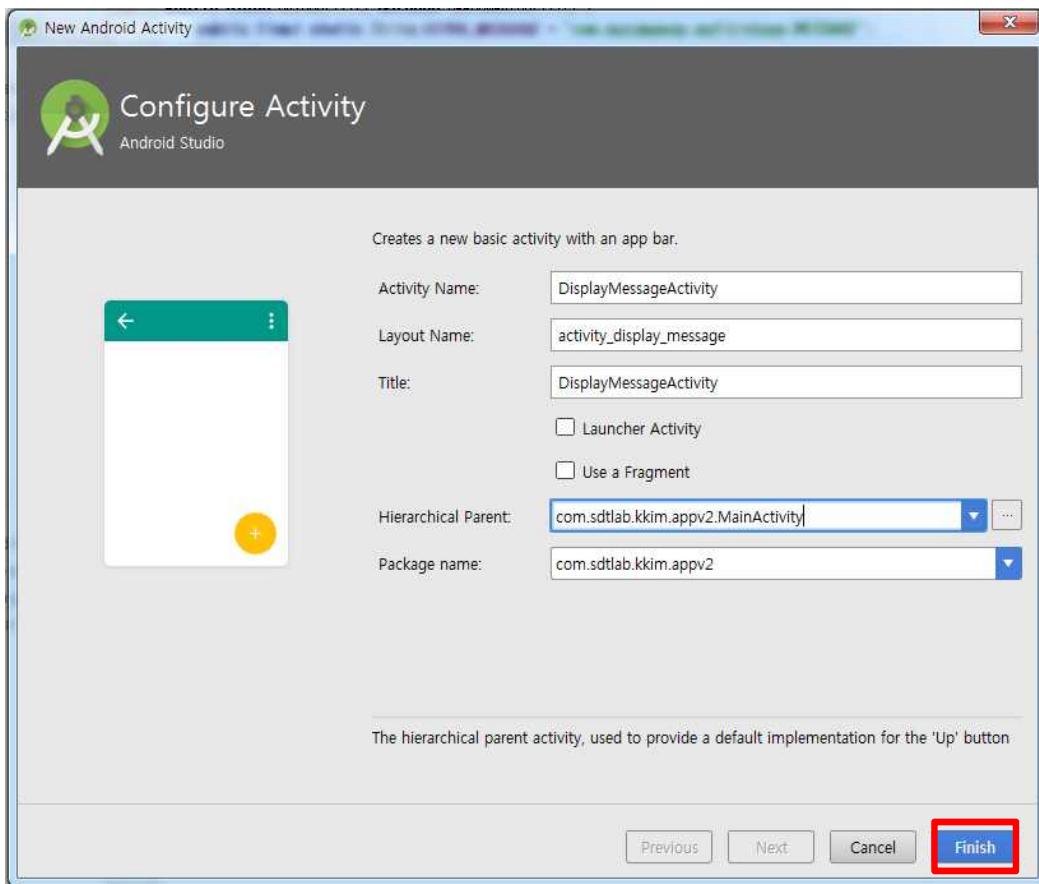
```
package com.sdtlab.kkim.appp2;
import android.content.Intent;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;

public void sendMessage(View view) {
    Intent intent = new Intent(this, DisplayMessageActivity.class);
    EditText editText = (EditText) findViewById(R.id.edit_message);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
}
```

7. Building a Simple User Interface (5/9)



7. Building a Simple User Interface (6/9)



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.sdtlab.kkим.appv2">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="AppV2"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="AppV2"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".DisplayMessageActivity"
            android:label="DisplayMessageActivity"
            android:parentActivityName=".MainActivity"
            android:theme="@style/AppTheme.NoActionBar">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.sdtlab.kkим.appv2.MainActivity" />
        </activity>
    </application>
</manifest>
```

7. Building a Simple User Interface (7/9)

The screenshot shows the Android Studio interface with two tabs open: `DisplayMessageActivity.java` and `activity_display_message.xml`. A red arrow points from the Java code to the XML file.

DisplayMessageActivity.java:

```
package com.sdtlab.kkim.appv2;
import ...
public class DisplayMessageActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_display_message);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(view -> {
            Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
        });
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    }
}
```

activity_display_message.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context="com.sdtlab.kkim.appv2.DisplayMessageActivity">
    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">
        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />
    </android.support.design.widget.AppBarLayout>
    <include layout="@layout/content_display_message" />
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="16dp"
        android:src="@android:drawable/ic_dialog_email" />
</android.support.design.widget.CoordinatorLayout>
```

7. Building a Simple User Interface (8/9)

The screenshot shows two code editors side-by-side. The left editor displays `DisplayMessageActivity.java` and the right editor displays `content_display_message.xml`.

DisplayMessageActivity.java:

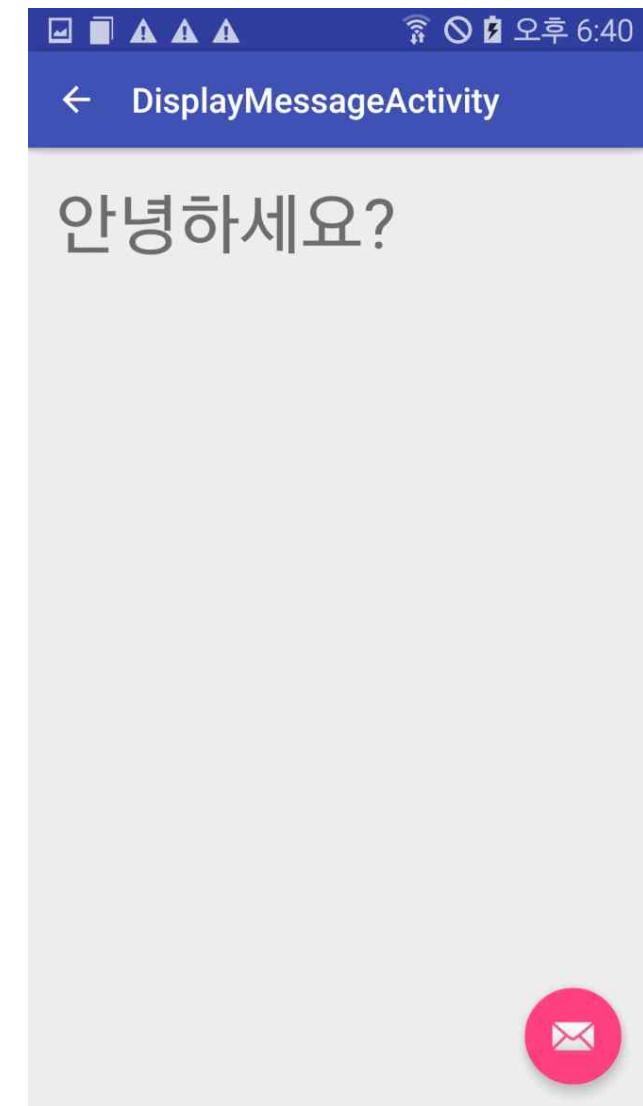
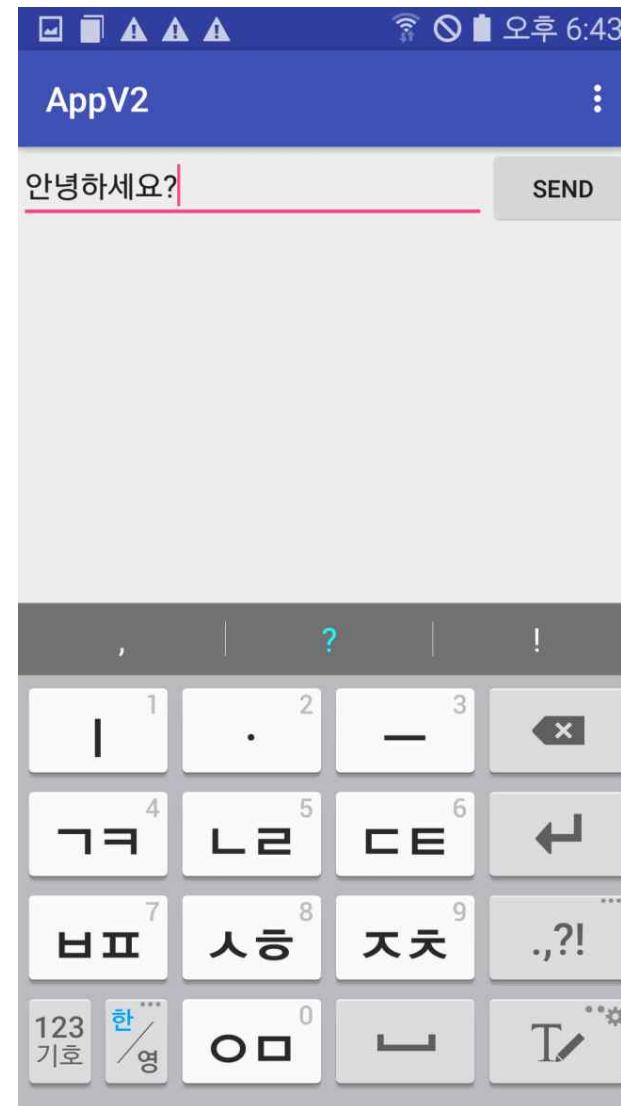
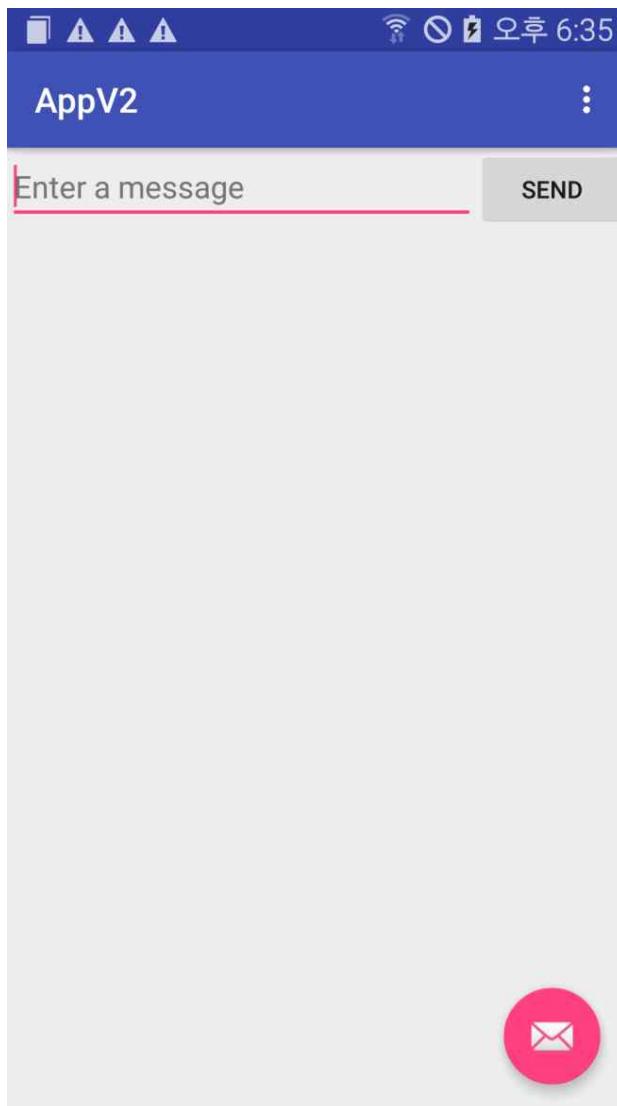
```
import ...  
  
public class DisplayMessageActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_display_message);  
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  
        setSupportActionBar(toolbar);  
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);  
        fab.setOnClickListener((view) -> {  
            Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)  
                .setAction("Action", null).show();  
        });  
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);  
  
        Intent intent = getIntent();  
        String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);  
        TextView textView = new TextView(this);  
        textView.setTextSize(40);  
        textView.setText(message);  
  
        RelativeLayout layout = (RelativeLayout) findViewById(R.id.content);  
        layout.addView(textView);  
    }  
}
```

content_display_message.xml:

```
<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingBottom="16dp"  
    android:paddingLeft="16dp"  
    android:paddingRight="16dp"  
    android:paddingTop="16dp"  
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingViewBehavior"  
    tools:context="com.sdtlab.kkim.app2.DisplayMessageActivity"  
    tools:showIn="@layout/activity_display_message"  
    android:id="@+id/content">  
</RelativeLayout>
```

A red arrow points from the highlighted `content` ID in the Java code to the corresponding `android:id` attribute in the XML layout file.

7. Building a Simple User Interface (9/9)



Android App Programming

August 4, 2015

Prof. Kyosun Kim
Incheon National University



Outline

1. Android Studio Installation
2. Tutorial I
3. Tutorial II
4. Bluetooth Chat
5. Infra-Red Remote Control V1
6. Infra-Red Remote Control V2

□ 참조: The C Programming Language

- ◆ <https://www.youtube.com/playlist?list=PLmngpJVWmdCwQueHIid7BSLJj0at7yEx9>

□ 참조: Atmega128 Micro-Controller

- ◆ https://www.youtube.com/watch?v=kvnc2L5pA78&list=PLmngpJVWmdCyMUuNNDRMIVcfgfYj_YIq9

3. Tutorial II

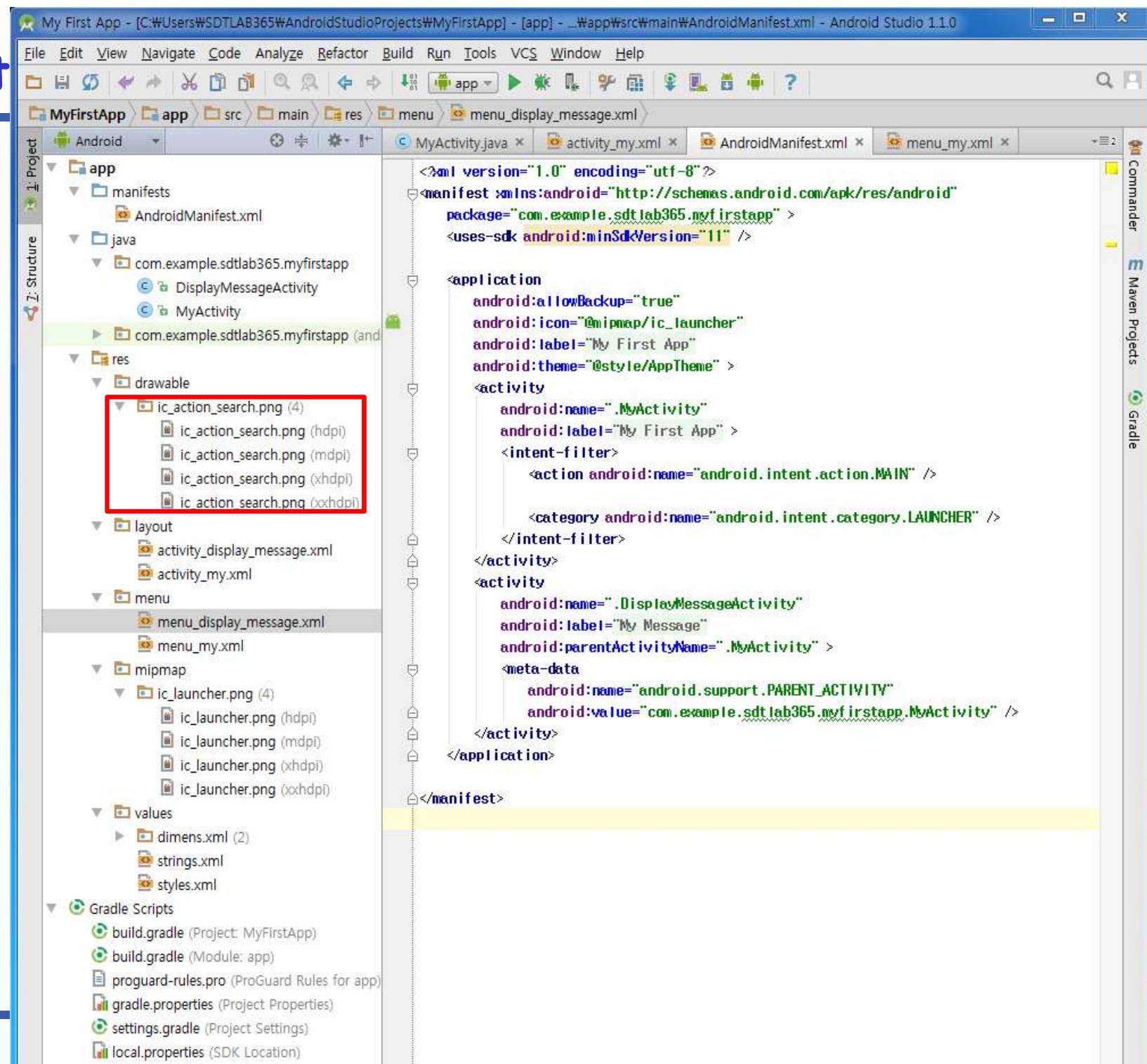
February 21, 2015

Prof. Kyosun Kim
Incheon National University

Outline

1. Android Studio
2. Installation of Icons
3. Adding the Action Bar
4. Styling the Action Bar - Deprecated

1. Android St



2. Installation of Icons

Download icons

- ◆ <http://developer.android.com/design/downloads/index.html#action-bar-icon-pack>
- ◆ unzip Android_Design_Icons_20131106

Right click the Drawable folder > Select New > Image Asset

◆ Asset Type

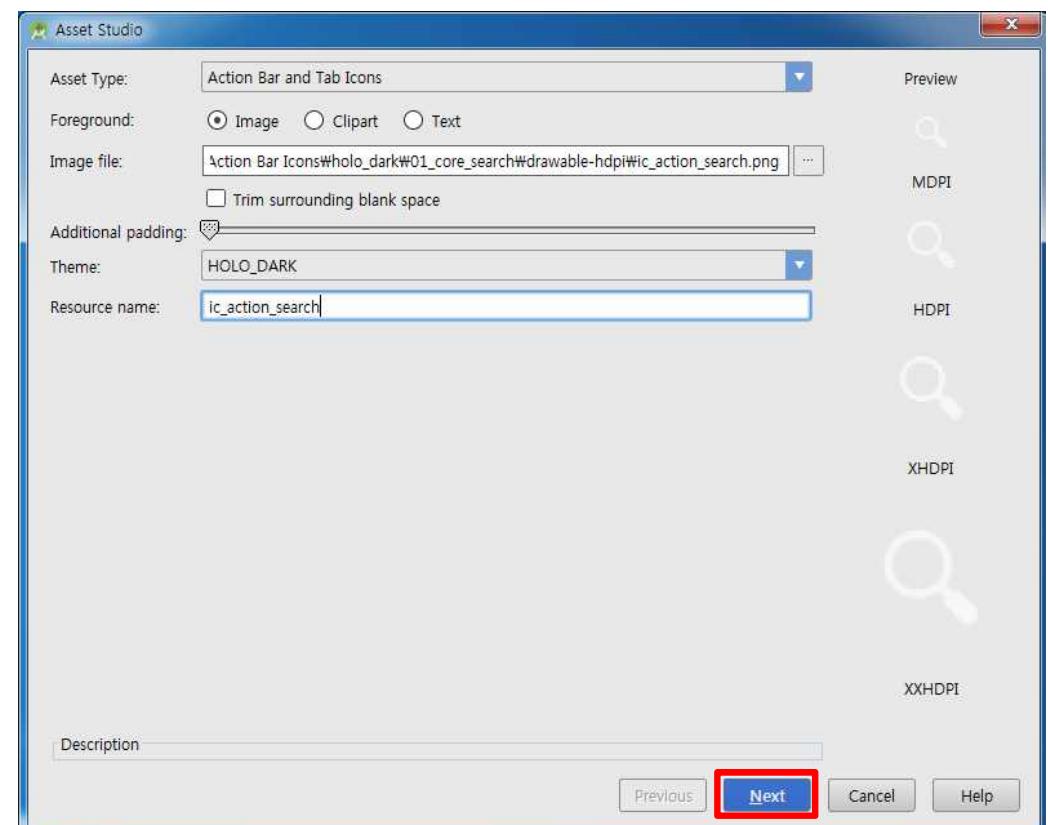
- ✓ Action Bar and Tab Icons

◆ Image file

✓ Android Design -
Icons 20131120\
Action Bar Icons\
holo_dark\
01_core_search\
drawable-hdpi\
ic_action_search.png

◆ Resource name

- ✓ ic_action_search



3. Adding the Action Bar (1/3)

The screenshot shows the AndroidManifest.xml file and a portion of the MainActivity.java code.

AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sdtlab365.myfirstapp" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".DisplayMessageActivity"
            android:label="@string/title_activity_display_message"
            android:parentActivityName=".MainActivity" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.sdtlab365.myfirstapp.MainActivity" />
        </activity>
    </application>
</manifest>
```

MainActivity.java:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle presses on the action bar items
    switch(item.getItemId()) {
        case R.id.action_search:
            openSearch();
            return true;
        case R.id.action_settings:
            openSettings();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

private void openSettings() { }

private void openSearch() { }

public void sendMessage(View view) {
    Intent intent = new Intent(this, DisplayMessageActivity.class);
    EditText editText = (EditText) findViewById(R.id.edit_message);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivity(intent);
}
```

3. Adding the Action Bar (2/3)

The screenshot shows the Android Studio interface with two tabs open: `MyActivity.java` and `menu_my.xml`.

MyActivity.java:

```
package com.example.sdtlab365.myfirstapp;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.content.Intent;
import android.widget.EditText;

public class MyActivity extends ActionBarActivity {
    public final static String EXTRA_MESSAGE = "com.example.sdtlab365.myfirstapp.MESSAGE";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu items for use in the action bar
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_my, menu);
        return super.onCreateOptionsMenu(menu);
    }
}
```

menu_my.xml:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools" tools:context=".MyActivity">
    <item android:id="@+id/action_search"
          android:icon="@drawable/ic_action_search"
          android:title="Search"
          app:showAsAction="ifRoom" />
    <item android:id="@+id/action_settings"
          android:icon="@drawable/ic_action_setting"
          android:title="@string/action_settings"
          android:orderInCategory="100"
          app:showAsAction="never" />
</menu>
```

MyActivity.java (Annotations):

- `MyActivity` (extends `ActionBarActivity`)
- `onCreate` (with `savedInstanceState`)
- `setContentView` (with `activity_my`)
- `onCreateOptionsMenu` (with `menu`)

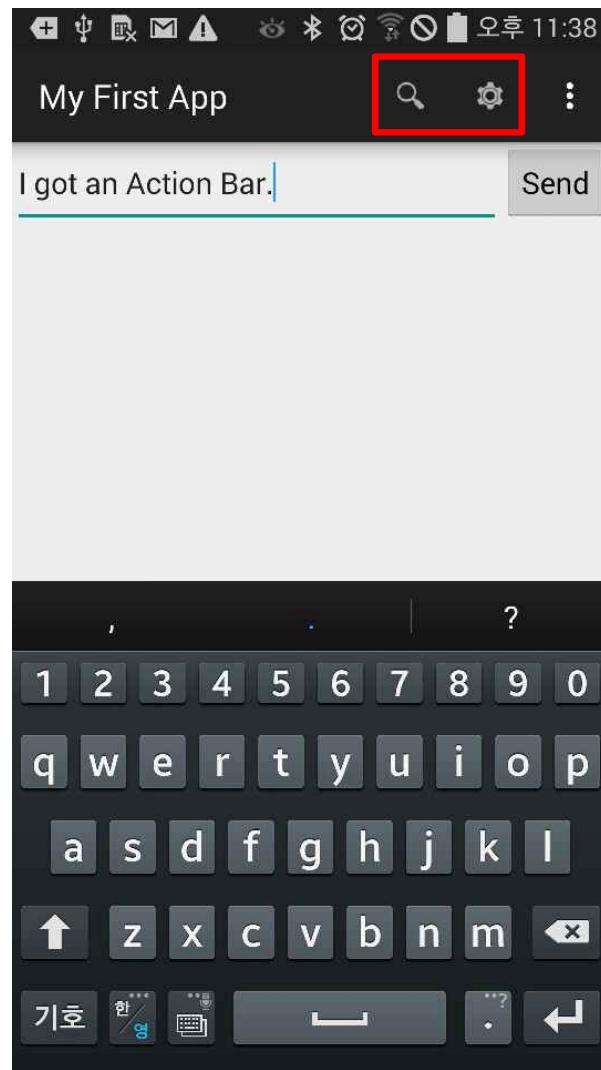
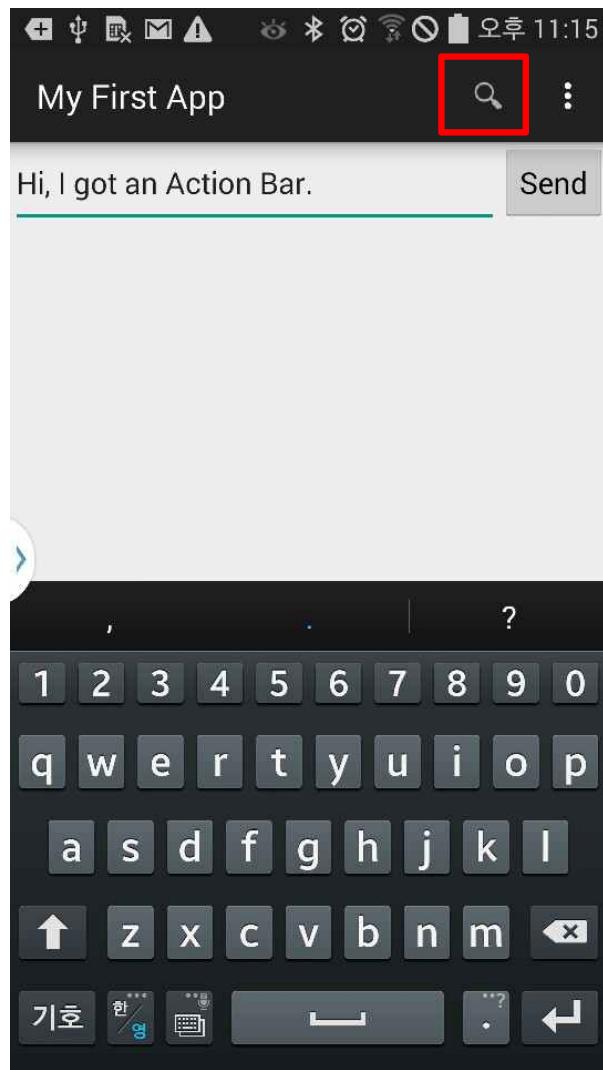
menu_my.xml (Annotations):

- `action_search` (with `icon`, `title`, `showAsAction`)
- `action_settings` (with `icon`, `title`, `orderInCategory`, `showAsAction`)

Bottom Navigation:

— — *Android Tut*

3. Adding the Action Bar (3/3)



4. Styling the Action Bar (1/5)

```
activity_my.xml x AndroidManifest.xml x themes.xml x
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sdtlab365.myfirstapp" >

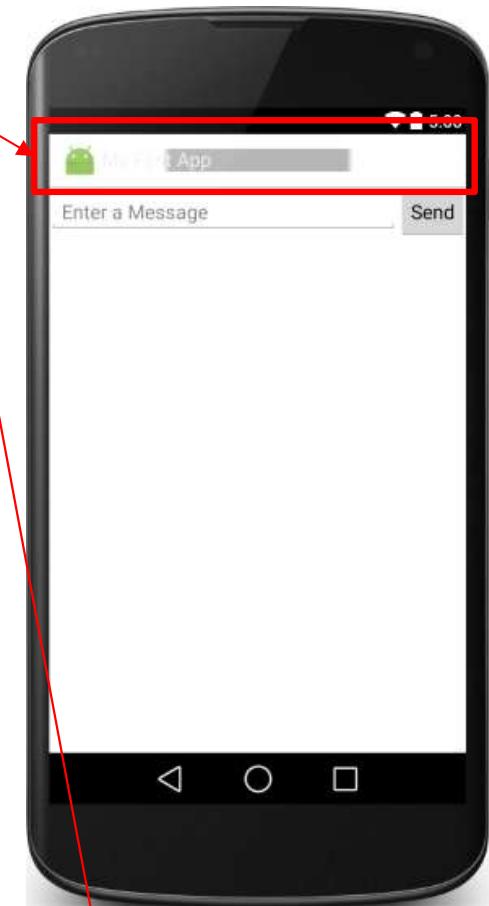
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/CustomActionBarTheme" >
        <activity>
```

```
activity_my.xml x AndroidManifest.xml x themes.xml x
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme">
        parent="@android:style/Theme.Holo.Light.DarkActionBar"
        <item name="android:actionBarStyle">@style/MyActionBar@drawable/actionbar_background

02-24 19:23:18.933 11878-11878/com.example.sdtlab365.myfirstapp E/AndroidRuntime: FATAL EXCEPTION: main  
Process: com.example.sdtlab365.myfirstapp, PID: 11878  
java.lang.RuntimeException: Unable to start activity ComponentInfo{com.example.sdtlab365.myfirstapp/com.example.  
at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2292)


```



4. Styling the Action Bar (2/5)



The figure shows two screenshots of an Android application. The left screenshot displays the XML code for the themes.xml file, which defines styles for the action bar. The right screenshot shows the resulting application interface on a virtual device.

themes.xml (Left Screenshot):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme">
        <parent>@android:style/Theme.Holo</parent>
        <item name="android: actionBarStyle">@style/MyActionBar</item>
        <item name="android: actionBarTabTextStyle">@style/MyActionBarTabText</item>
        <item name="android: actionBarTextColor">@color/actionbar_text</item>
    </style>
    <!-- ActionBar styles -->
    <style name="MyActionBar">
        <parent>@android:style/Widget.Holo.Light.ActionBar</parent>
        <item name="android: titleTextStyle">@style/MyActionBarTitleText</item>
    </style>
    <!-- ActionBar title text -->
    <style name="MyActionBarTitleText">
        <parent>@android: style/TextAppearance.Holo.Widget.ActionBar.Title</parent>
        <item name="android: textColor">@color/actionbar_text</item>
    </style>
    <!-- ActionBar tabs text style -->
    <style name="MyActionBarTabText">
        <parent>@android: style/Widget.Holo.ActionBar.TabText</parent>
        <item name="android: textColor">@color/actionbar_text</item>
    </style>
</resources>
```

colors.xml (Right Screenshot):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="actionbar_text">#fffff</color>
</resources>
```

Red boxes highlight specific elements:

- A red box surrounds the `actionbar_text` color definition in the colors.xml file.
- Red boxes highlight the `actionbar_text` references in the `MyActionBar`, `MyActionBarTitleText`, and `MyActionBarTabText` styles within the themes.xml file.

4. Styling the Action Bar (3/5)

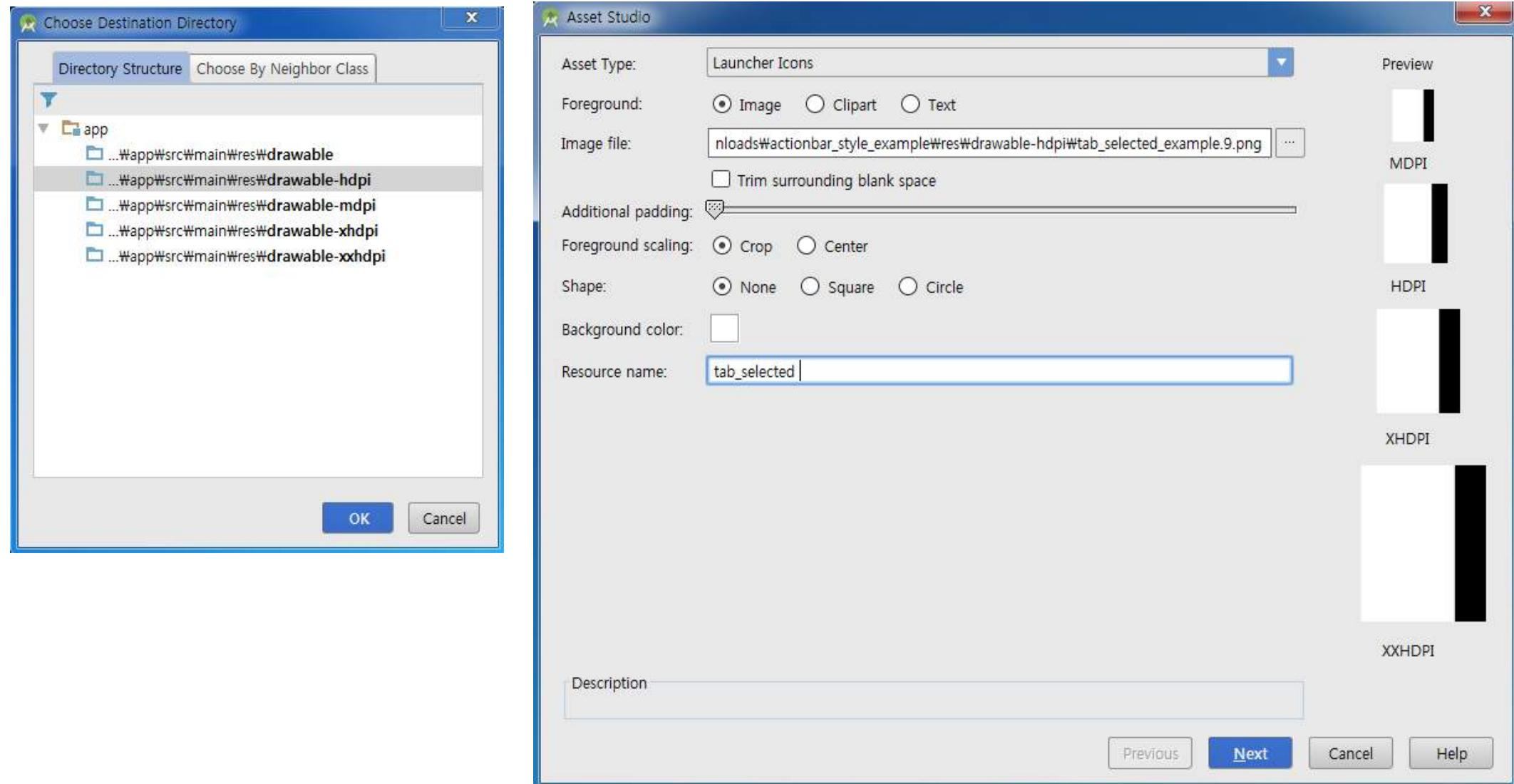
□ Chrome Only

The screenshot shows the "Android Action Bar Style Generator" website. At the top, there's a header with a "Fork me on GitHub" button. Below it, a message says "DEPRECATED: Consider using Toolbar or its support library equivalent." The main content area has a heading "Output resources" and a large "DOWNLOAD ZIP" button highlighted with a red box. To the right, there's a preview of an Android application interface with tabs and a navigation bar.



The screenshot shows the configuration interface of the "Android Action Bar Style Generator". It includes fields for "Style name" (set to "holo"), "Style compatibility" (set to "ActionBarSherlock"), "Base theme" (set to "Dark"), and "Action bar style" (set to "Solid"). On the right, there are several toggle switches for "Action bar texture", "Tab hairline style", "Neutral pressed states", "Action bar color", "Stacked color", "Tab indicator color", "Popup color", "Accent color", "Action mode background color", and "Action mode highlight color". A "Preview" window on the right shows a sample of the generated style with tabs and a navigation bar.

4. Styling the Action Bar (4/5)



4. Styling the Action Bar (5/5)

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Non focused states -->
    <item android:state_focused="false" android:state_selected="false"
          android:state_pressed="false"
          android:drawable="@android:color/transparent" />
    <item android:state_focused="false" android:state_selected="true"
          android:state_pressed="false"
          android:drawable="@drawable/tab_selected" />

    <!-- Focused states -->
    <item android:state_focused="true" android:state_selected="false"
          android:state_pressed="false"
          android:drawable="@drawable/tab_unselected_focused" />
    <item android:state_focused="true" android:state_selected="true"
          android:state_pressed="false"
          android:drawable="@drawable/tab_selected_focused" />

    <!-- Pressed -->
    <!-- Non focused states -->
    <item android:state_focused="false" android:state_selected="false"
          android:state_pressed="true"
          android:drawable="@drawable/tab_unselected_pressed" />
    <item android:state_focused="false" android:state_selected="true"
          android:state_pressed="true"
          android:drawable="@drawable/tab_selected_pressed" />

    <!-- Focused states -->
    <item android:state_focused="true" android:state_selected="false"
          android:state_pressed="true"
          android:drawable="@drawable/tab_unselected_pressed" />
    <item android:state_focused="true" android:state_selected="true"
          android:state_pressed="true"
          android:drawable="@drawable/tab_selected_pressed" />
</selector>
```

Deprecated !

Use Toolbar !

Use Material Design !

- Elevation, Shadow, & Color

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme" parent="@android:style/Theme.Holo">
        <item name="android:actionBarTabStyle">@style/MyActionBarTabs</item>
    </style>

    <!-- ActionBar tabs styles -->
    <style name="MyActionBarTabs" parent="@android:style/Widget.holo.ActionBar.TabView">
        <!-- tab indicator -->
        <item name="android:background">@drawable/actionbar_tab_indicator</item>
    </style>
</resources>
```