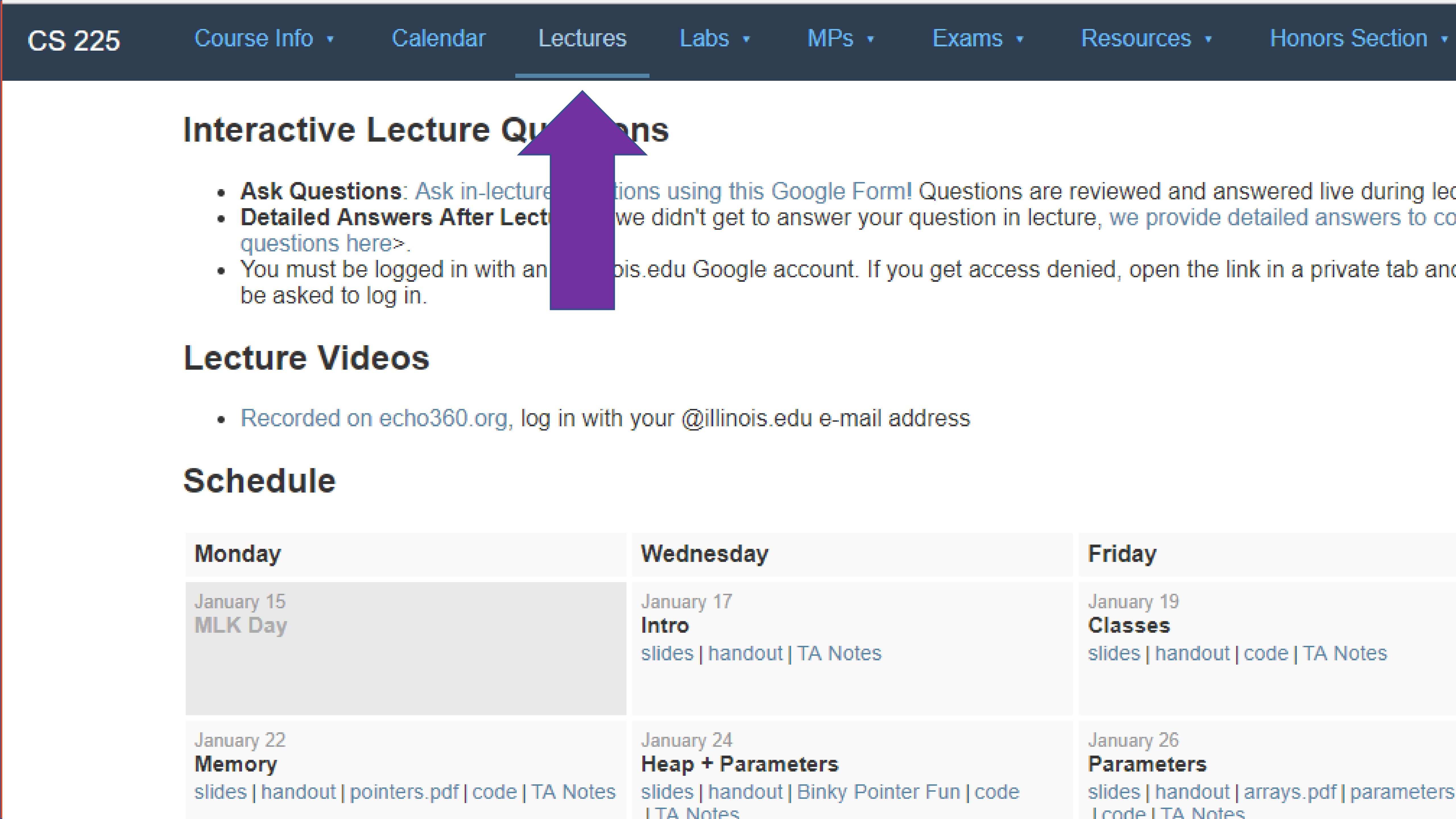
CS 225

Data Structures

Feb. 14 – Trees

Wade Fagen-Ulmschneider



Interactive Lecture Qu ons

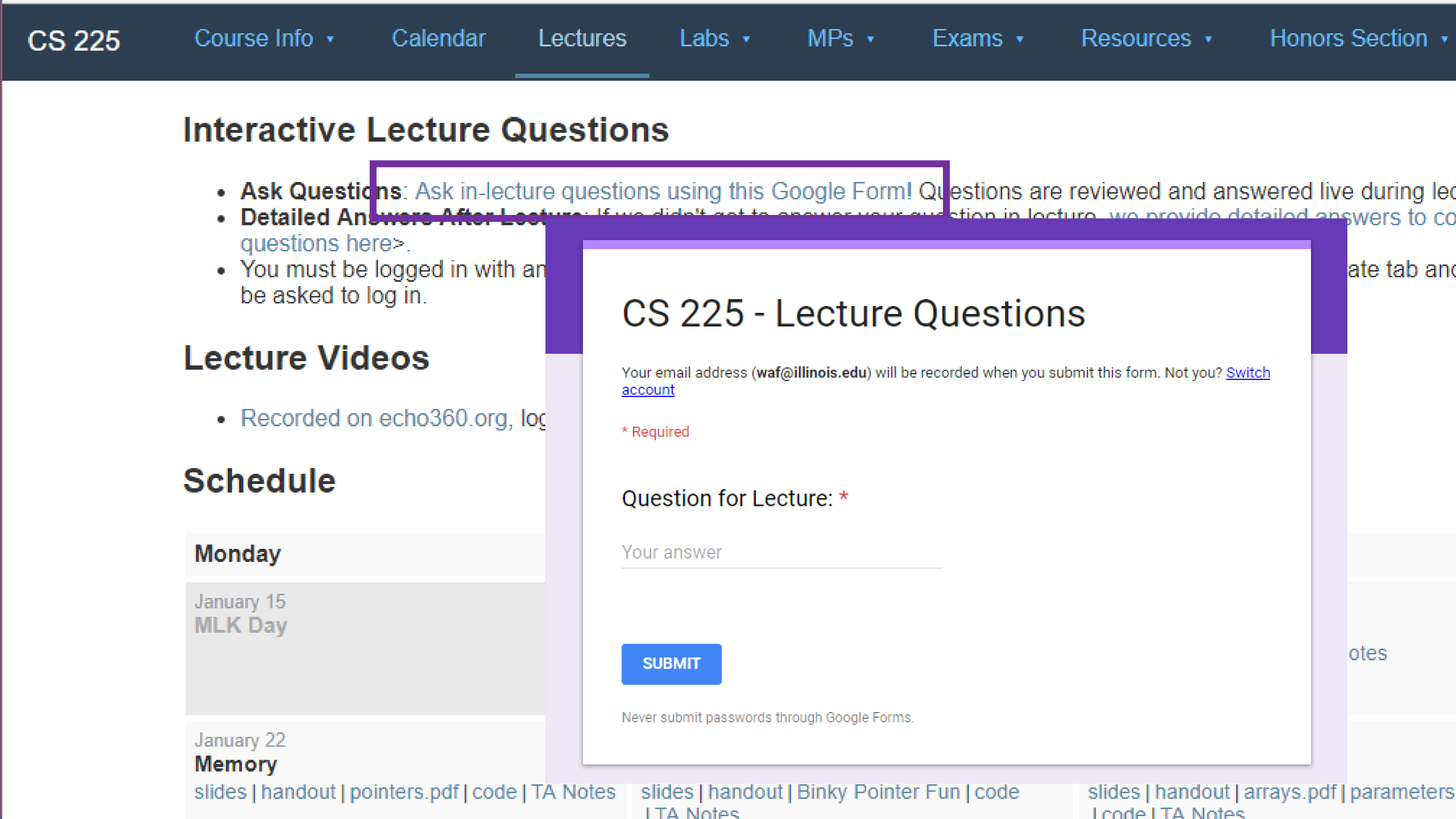
- **Ask Questions**: Ask in-lecture tions using this Google Form! Questions are reviewed and answered live during lec
- **Detailed Answers After Lect** we didn't get to answer your question in lecture, we provide detailed answers to co questions here>.
- You must be logged in with an bis.edu Google account. If you get access denied, open the link in a private tab and be asked to log in.

Lecture Videos

- Recorded on echo360.org, log in with your @illinois.edu e-mail address

Schedule

Monday	Wednesday	Friday
January 15 MLK Day	January 17 **Intro** slides | handout | TA Notes	January 19 **Classes** slides | handout | code | TA Notes
January 22 **Memory** slides | handout | pointers.pdf | code | TA Notes	January 24 **Heap + Parameters** slides | handout | Binky Pointer Fun | code | TA Notes	January 26 **Parameters** slides | handout | arrays.pdf | parameters | code | TA Notes



Interactive Lecture Questions

- **Ask Questions**: Ask in-lecture questions using this Google Form! Questions are reviewed and answered live during lec
- **Detailed Answers After Lecture**: If we didn't get to answer your question in lecture, we provide detailed answers to co questions here>.
- You must be logged in with an ate tab and be asked to log in.

Lecture Videos

- Recorded on echo360.org, log

Schedule

Monday		
January 15 MLK Day		otes
January 22 **Memory** slides | handout | pointers.pdf | code | TA Notes	slides | handout | Binky Pointer Fun | code | TA Notes	slides | handout | arrays.pdf | parameters | code | TA Notes

CS 225 - Lecture Questions

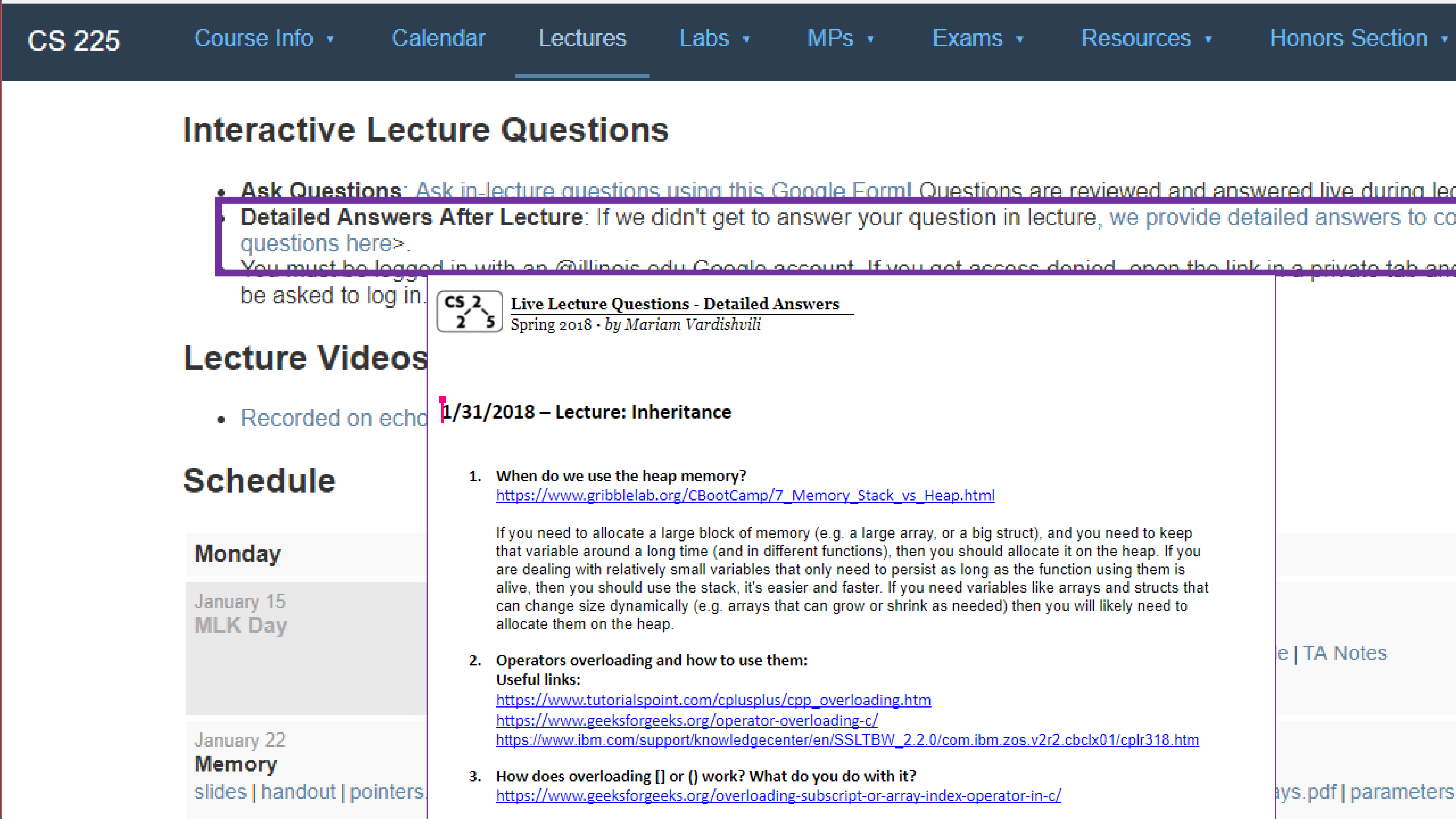
Your email address (**waf@illinois.edu**) will be recorded when you submit this form. Not you? Switch account

* Required

Question for Lecture: *

Your answer

Never submit passwords through Google Forms.



CS 225 | Course Info ▾ | Calendar | Lectures | Labs ▾ | MPs ▾ | Exams ▾ | Resources ▾ | Honors Section ▾

Interactive Lecture Questions

- **Ask Questions**: Ask in-lecture questions using this Google Form! Questions are reviewed and answered live during lec
- **Detailed Answers After Lecture**: If we didn't get to answer your question in lecture, we provide detailed answers to co questions here>.
- You must be logged in with an @illinois.edu Google account. If you get access denied, open the link in a private tab and be asked to log in.

Lecture Videos

- Recorded on echo

Schedule

Monday

January 15
MLK Day

e | TA Notes

January 22
Memory
slides | handout | pointers. ... ys.pdf | parameters

Live Lecture Questions - Detailed Answers
Spring 2018 · *by Mariam Vardishvili*

1/31/2018 – Lecture: Inheritance

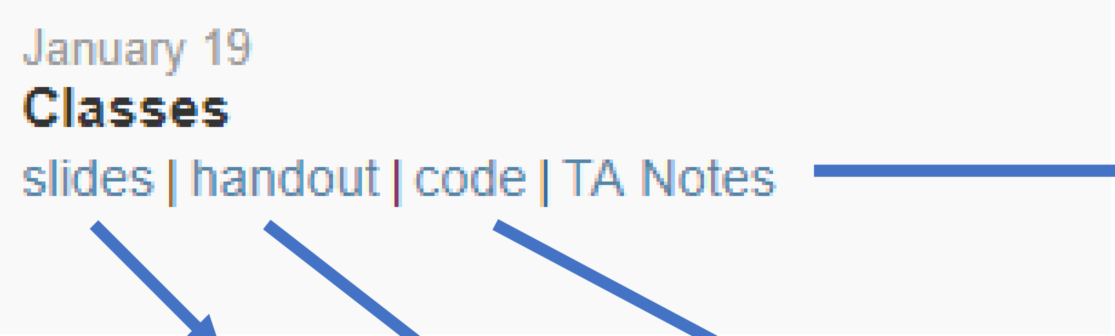
1. **When do we use the heap memory?**
   https://www.gribblelab.org/CBootCamp/7_Memory_Stack_vs_Heap.html

   If you need to allocate a large block of memory (e.g. a large array, or a big struct), and you need to keep that variable around a long time (and in different functions), then you should allocate it on the heap. If you are dealing with relatively small variables that only need to persist as long as the function using them is alive, then you should use the stack, it's easier and faster. If you need variables like arrays and structs that can change size dynamically (e.g. arrays that can grow or shrink as needed) then you will likely need to allocate them on the heap.

2. **Operators overloading and how to use them:**
   Useful links:
   https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm
   https://www.geeksforgeeks.org/operator-overloading-c/
   https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.cbclx01/cplr318.htm

3. **How does overloading [] or () work? What do you do with it?**
   https://www.geeksforgeeks.org/overloading-subscript-or-array-index-operator-in-c/

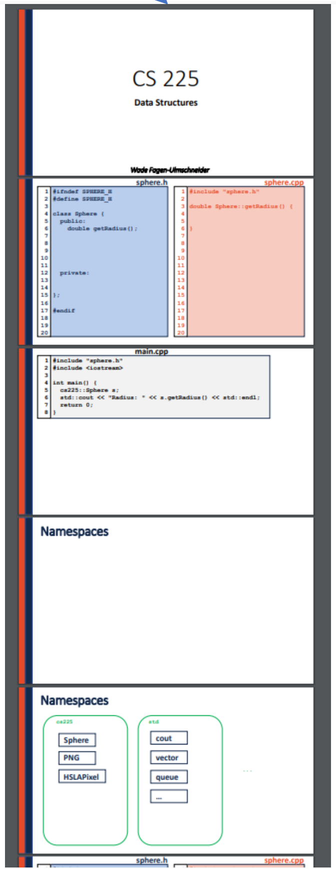
Lecture Resources



January 19

Classes

slides | handout | code | TA Notes



CS 225

Data Structures

Wade Fagen-Ulmschneider

sphere.h

```
#ifndef SPHERE_H
#define SPHERE_H

class Sphere {
  public:
    double getRadius();




  private:


};

#endif
```

sphere.cpp

```
#include "sphere.h"

double Sphere::getRadius() {


}
```

main.cpp

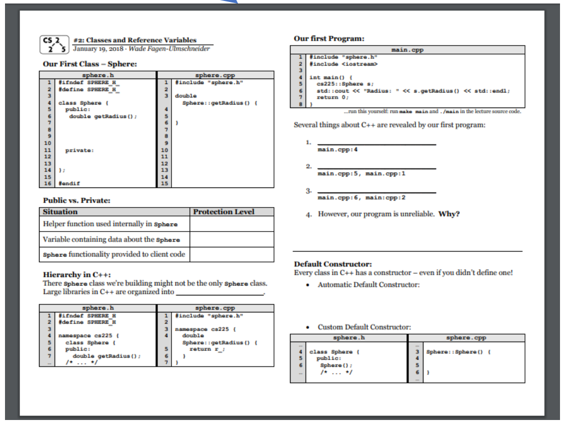
```
#include "sphere.h"
#include <iostream>

int main() {
  cs225::Sphere s;
  std::cout << "Radius: " << s.getRadius() << std::endl;
  return 0;
}
```

Namespaces

Namespaces

sphere.h

sphere.cpp



#2: Classes and Reference Variables

January 19, 2018 · *Wade Fagen-Ulmschneider*

Our First Class – Sphere:

sphere.h

```
#ifndef SPHERE_H_
#define SPHERE_H_

class Sphere {
  public:
    double getRadius();




  private:


};

#endif
```

sphere.cpp

```
#include "sphere.h"

double
  Sphere::getRadius() {


}
```

Public vs. Private:

Situation	Protection Level
Helper function used internally in `Sphere`	
Variable containing data about the `Sphere`	
`Sphere` functionality provided to client code	

Hierarchy in C++:

There `Sphere` class we're building might not be the only `Sphere` class. Large libraries in C++ are organized into ________________.

sphere.h

```
#ifndef SPHERE_H
#define SPHERE_H

namespace cs225 {
  class Sphere {
  public:
    double getRadius();
  /* ... */
```

sphere.cpp

```
#include "sphere.h"

namespace cs225 {
  double
  Sphere::getRadius() {
    return r_;
  }
}
```

Our first Program:

main.cpp

```
1  #include "sphere.h"
2  #include <iostream>
3
4  int main() {
5    cs225::Sphere s;
6    std::cout << "Radius: " << s.getRadius() << std::endl;
7    return 0;
8  }
```

...run this yourself: run `make main` and `./main` in the lecture source code.

Several things about C++ are revealed by our first program:

1. ____________________
   `main.cpp:4`
2. ____________________
   `main.cpp:5, main.cpp:1`
3. ____________________
   `main.cpp:6, main:cpp:2`
4. However, our program is unreliable. **Why?**

Default Constructor:

Every class in C++ has a constructor – even if you didn't define one!

- Automatic Default Constructor:

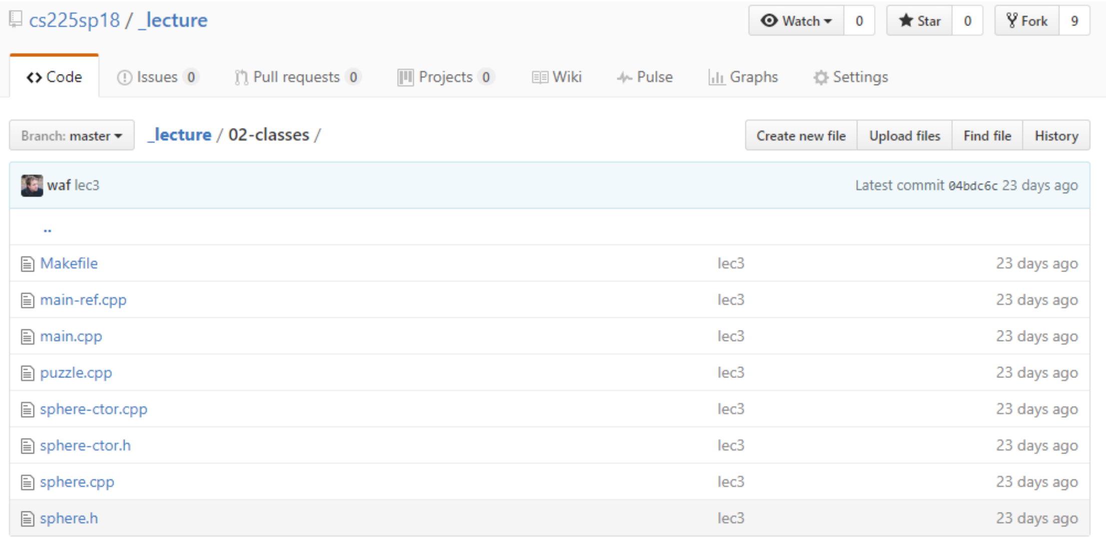
- Custom Default Constructor:

sphere.h

```
...
class Sphere {
  public:
    Sphere();
    /* ... */
...
```

sphere.cpp

```
...
Sphere::Sphere() {


}
...
```



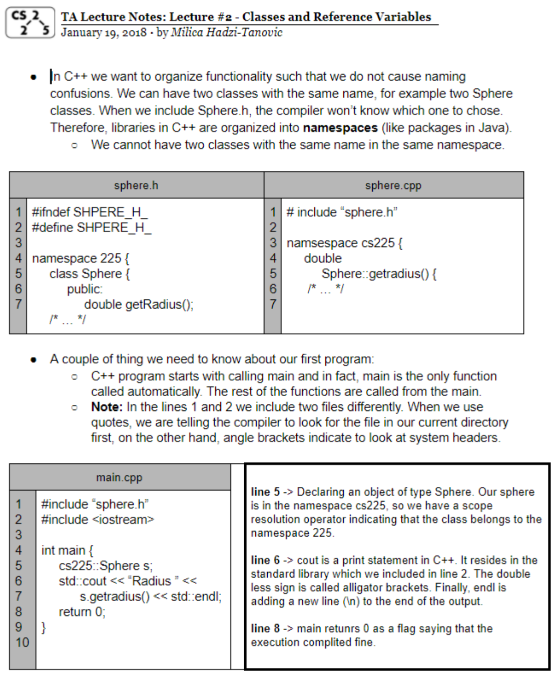
cs225sp18 / _lecture

_lecture / 02-classes /

waf lec3 — Latest commit 04bdc6c 23 days ago

..

Makefile	lec3	23 days ago
main-ref.cpp	lec3	23 days ago
main.cpp	lec3	23 days ago
puzzle.cpp	lec3	23 days ago
sphere-ctor.cpp	lec3	23 days ago
sphere-ctor.h	lec3	23 days ago
sphere.cpp	lec3	23 days ago
sphere.h	lec3	23 days ago



TA Lecture Notes: Lecture #2 - Classes and Reference Variables

January 19, 2018 · by *Milica Hadzi-Tanovic*

- In C++ we want to organize functionality such that we do not cause naming confusions. We can have two classes with the same name, for example two Sphere classes. When we include Sphere.h, the compiler won't know which one to chose. Therefore, libraries in C++ are organized into **namespaces** (like packages in Java).
  - We cannot have two classes with the same name in the same namespace.

sphere.h

```
#ifndef SHPERE_H_
#define SHPERE_H_

namespace 225 {
   class Sphere {
      public:
         double getRadius();
   /* … */
```

sphere.cpp

```
# include “sphere.h”

namsespace cs225 {
   double
      Sphere::getradius() {
    /* … */
```

- A couple of thing we need to know about our first program:
  - C++ program starts with calling main and in fact, main is the only function called automatically. The rest of the functions are called from the main.
  - **Note:** In the lines 1 and 2 we include two files differently. When we use quotes, we are telling the compiler to look for the file in our current directory first, on the other hand, angle brackets indicate to look at system headers.

main.cpp

```
1   #include “sphere.h”
2   #include <iostream>
3
4   int main {
5      cs225::Sphere s;
6      std::cout << “Radius ” <<
7           s.getradius() << std::endl;
8      return 0;
9   }
10
```

line 5 -> Declaring an object of type Sphere. Our sphere is in the namespace cs225, so we have a scope resolution operator indicating that the class belongs to the namespace 225.

line 6 -> cout is a print statement in C++. It resides in the standard library which we included in line 2. The double less sign is called alligator brackets. Finally, endl is adding a new line (\n) to the end of the output.

line 8 -> main retunrs 0 as a flag saying that the execution complited fine.

Queue.h

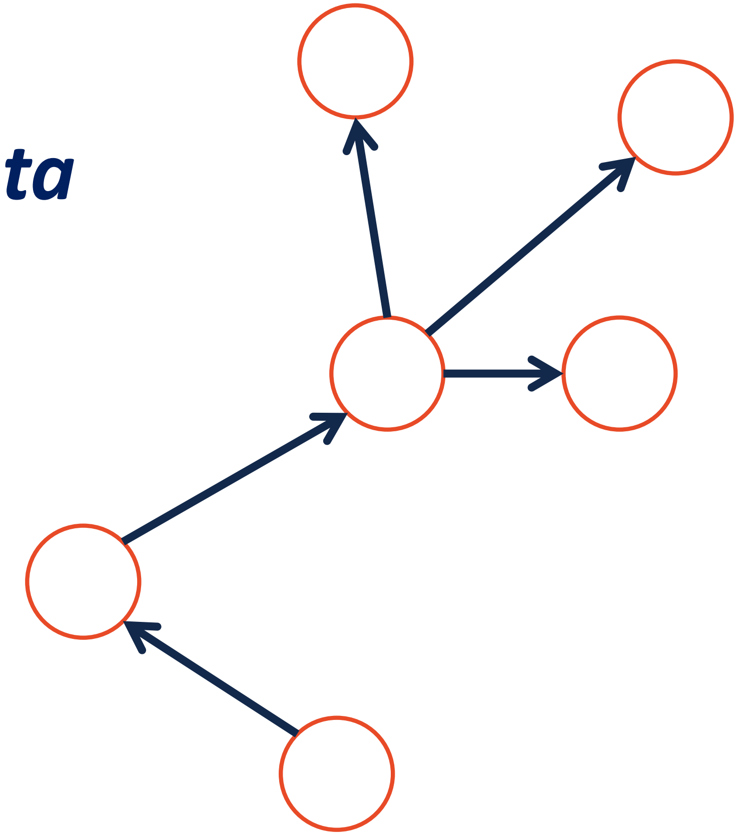
```
template <class QE>
class Queue {
  public:
    class QueueIterator : public std::iterator<std::bidirectional_iterator_tag, QE> {
      public:
        QueueIterator(unsigned index);
        QueueIterator& operator++();
        bool operator==(const QueueIterator &other);
        bool operator!=(const QueueIterator &other);
        QE& operator*();
        QE* operator->();
      private:
        int location_;

    };


  /* ... */

  private:
    QE* arr_; unsigned capacity_, count_, entry_, exit_;
};

```

Big Ideas

How does the **Queue** and the **QueueIterator** interact?

Trees

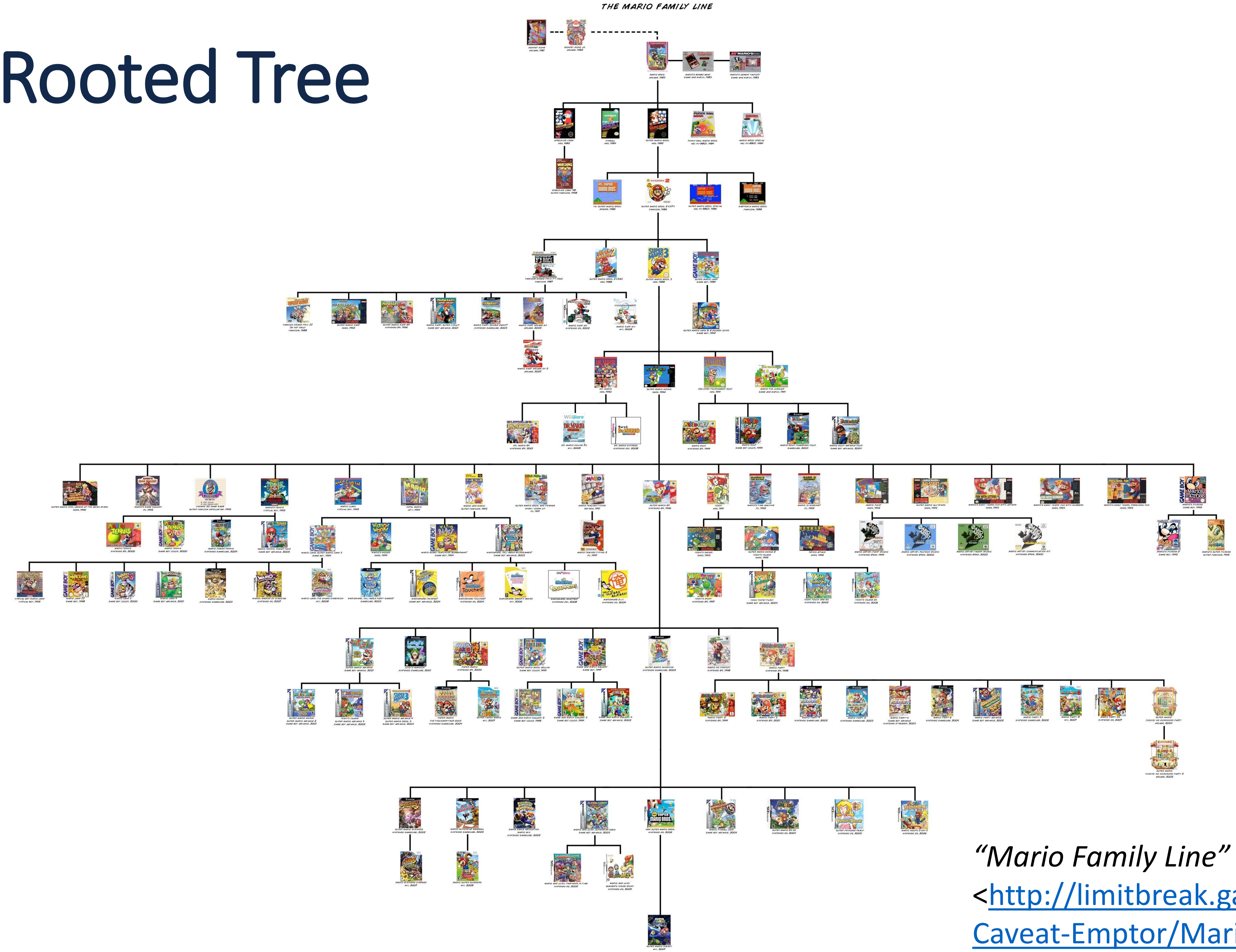
> ***"The most important non-linear data structure in computer science."***
>
> ***- David Knuth, The Art of Programming, Vol. 1***



A tree is:

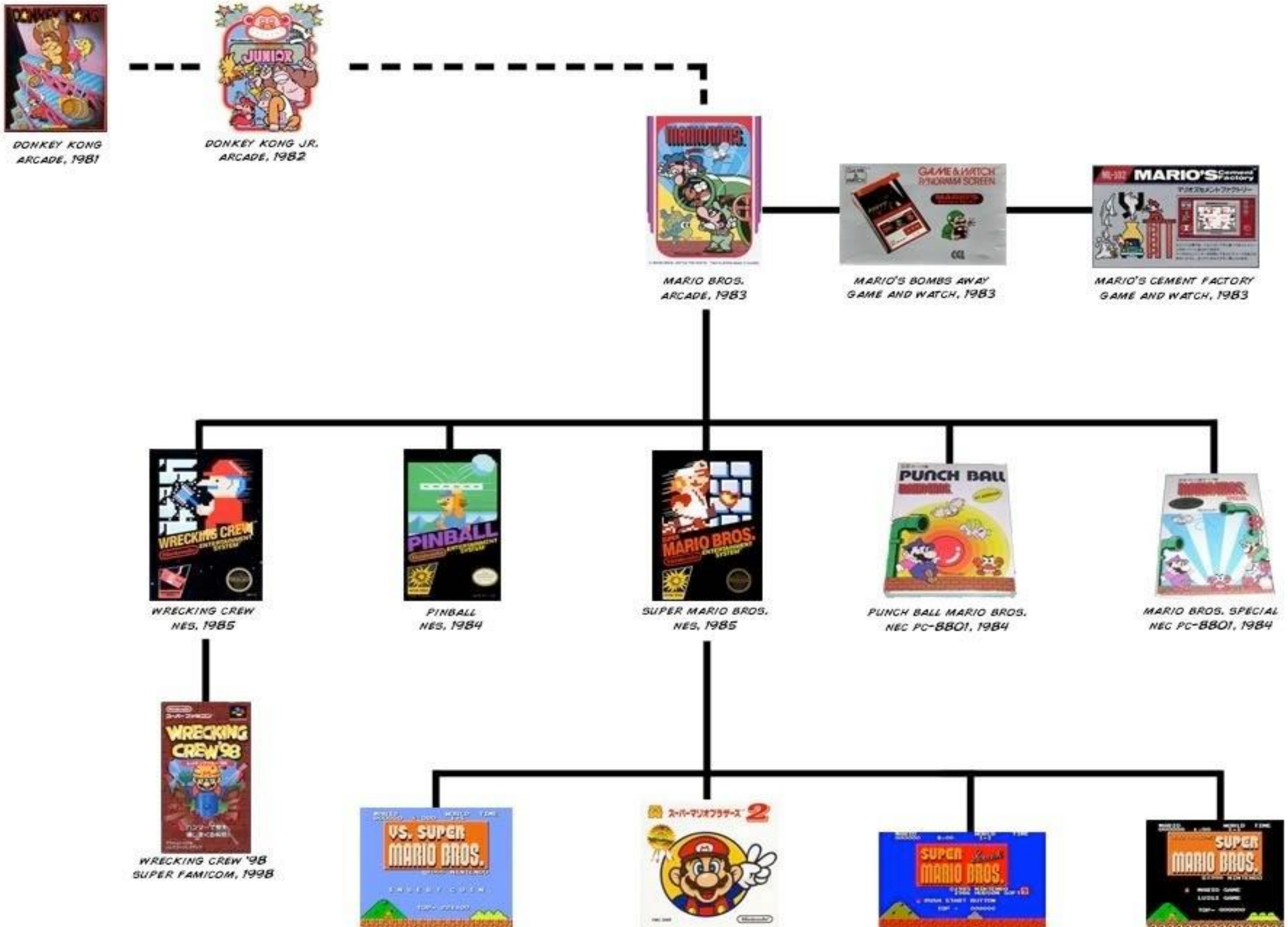
-
-

A Rooted Tree



"Mario Family Line"
<http://limitbreak.gameriot.com/blogs/Caveat-Emptor/Mario-Family-Line>

THE MARIO FAMILY LINE



DONKEY KONG
ARCADE, 1981

DONKEY KONG JR.
ARCADE, 1982

MARIO BROS.
ARCADE, 1983

MARIO'S BOMBS AWAY
GAME AND WATCH, 1983

MARIO'S CEMENT FACTORY
GAME AND WATCH, 1983

WRECKING CREW
NES, 1985

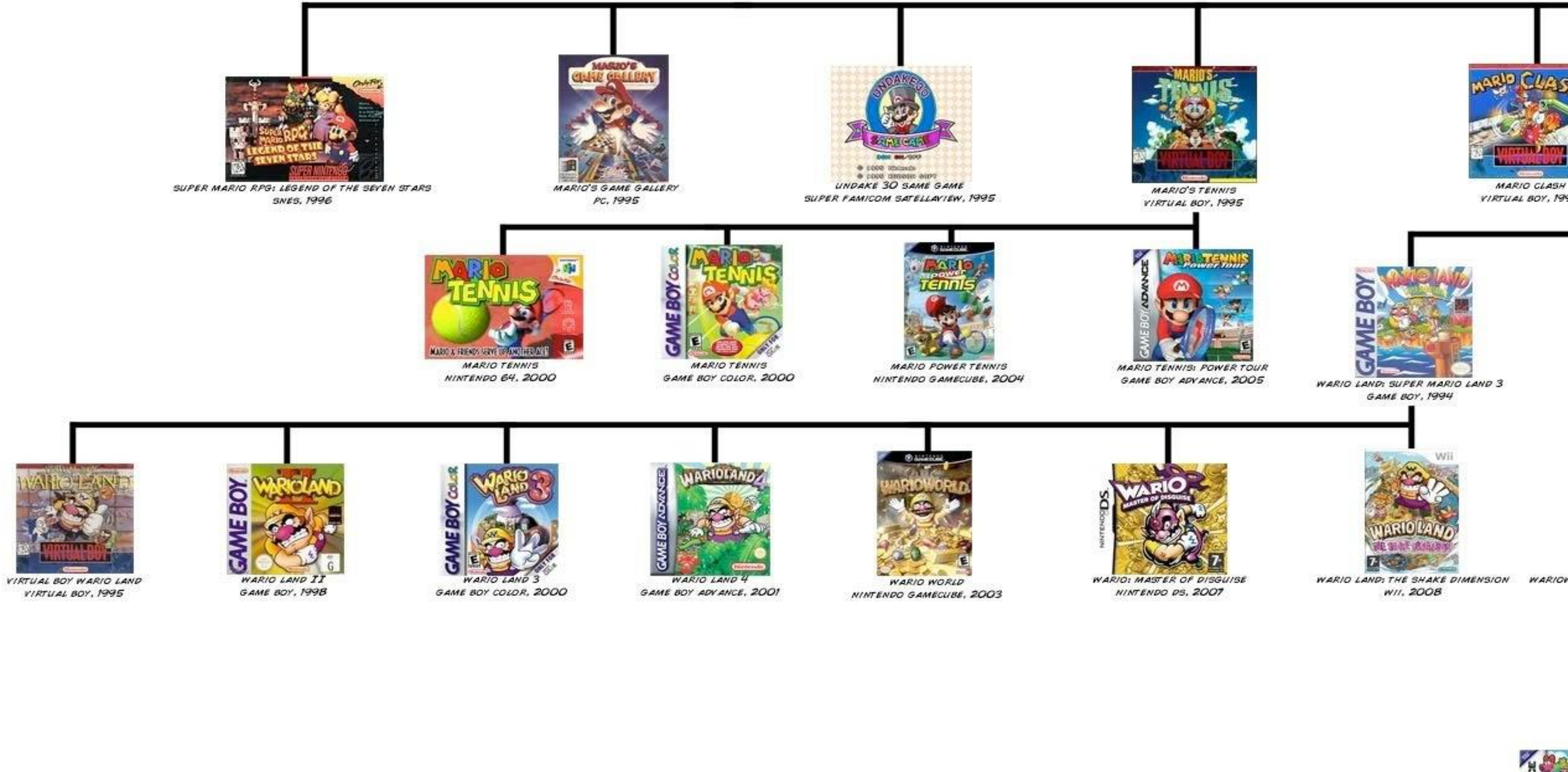
PINBALL
NES, 1984

SUPER MARIO BROS.
NES, 1985

PUNCH BALL MARIO BROS.
NEC PC-8801, 1984

MARIO BROS. SPECIAL
NEC PC-8801, 1984

WRECKING CREW '98
SUPER FAMICOM, 1998



SUPER MARIO RPG: LEGEND OF THE SEVEN STARS
SNES, 1996

MARIO'S GAME GALLERY
PC, 1995

UNDAKE 30 SAME GAME
SUPER FAMICOM SATELLAVIEW, 1995

MARIO'S TENNIS
VIRTUAL BOY, 1995

MARIO CLASH
VIRTUAL BOY, 199

MARIO TENNIS
NINTENDO 64, 2000

MARIO TENNIS
GAME BOY COLOR, 2000

MARIO POWER TENNIS
NINTENDO GAMECUBE, 2004

MARIO TENNIS: POWER TOUR
GAME BOY ADVANCE, 2005

WARIO LAND: SUPER MARIO LAND 3
GAME BOY, 1994

VIRTUAL BOY WARIO LAND
VIRTUAL BOY, 1995

WARIO LAND II
GAME BOY, 1998

WARIO LAND 3
GAME BOY COLOR, 2000

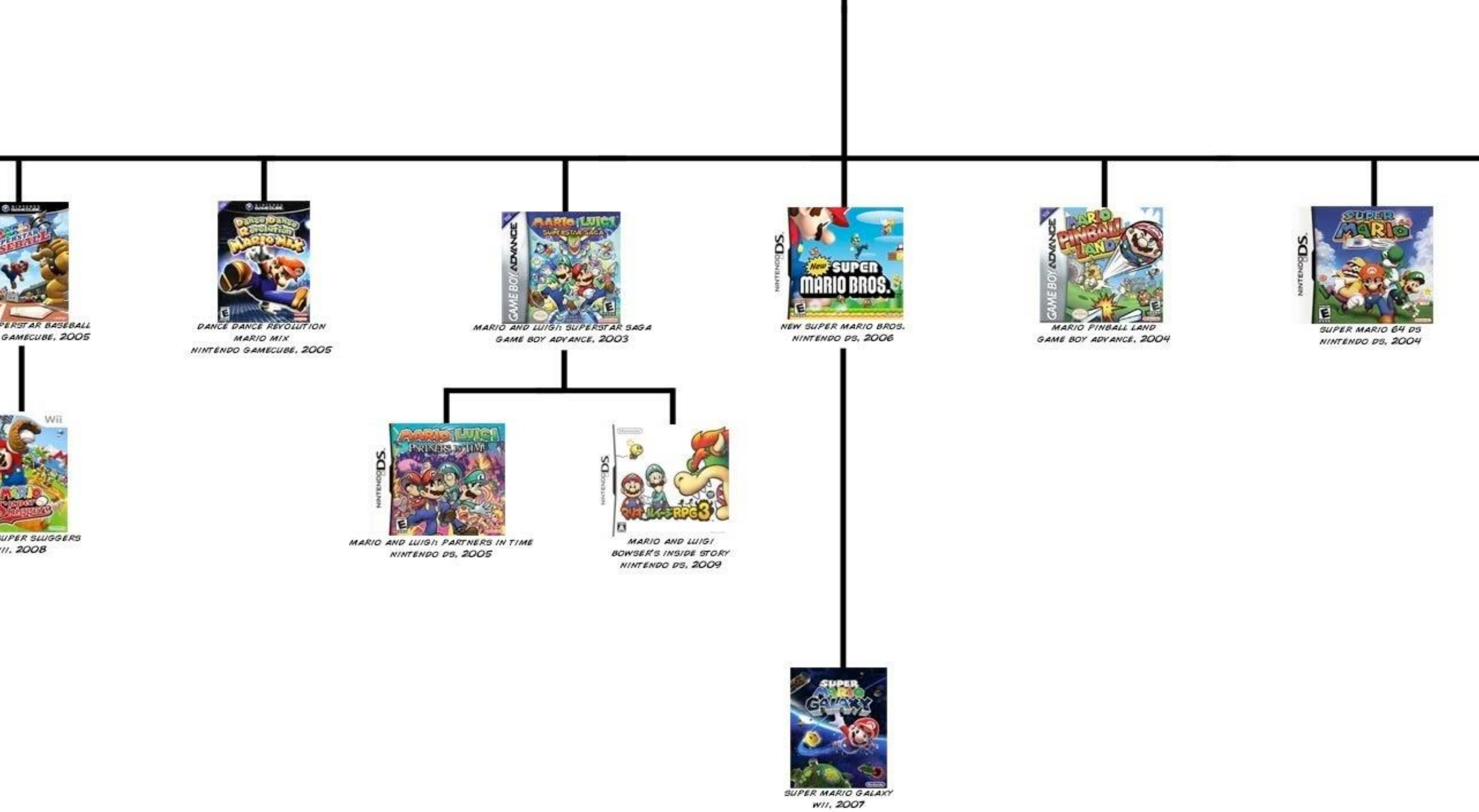
WARIO LAND 4
GAME BOY ADVANCE, 2001

WARIO WORLD
NINTENDO GAMECUBE, 2003

WARIO: MASTER OF DISGUISE
NINTENDO DS, 2007

WARIO LAND: THE SHAKE DIMENSION
WII, 2008

WARIOW



PERSTAR BASEBALL
GAMECUBE, 2005

UPER SLUGGERS
II, 2008

DANCE DANCE REVOLUTION
MARIO MIX
NINTENDO GAMECUBE, 2005

MARIO AND LUIGI: SUPERSTAR SAGA
GAME BOY ADVANCE, 2003

MARIO AND LUIGI: PARTNERS IN TIME
NINTENDO DS, 2005

MARIO AND LUIGI
BOWSER'S INSIDE STORY
NINTENDO DS, 2009

NEW SUPER MARIO BROS.
NINTENDO DS, 2006

SUPER MARIO GALAXY
WII, 2007

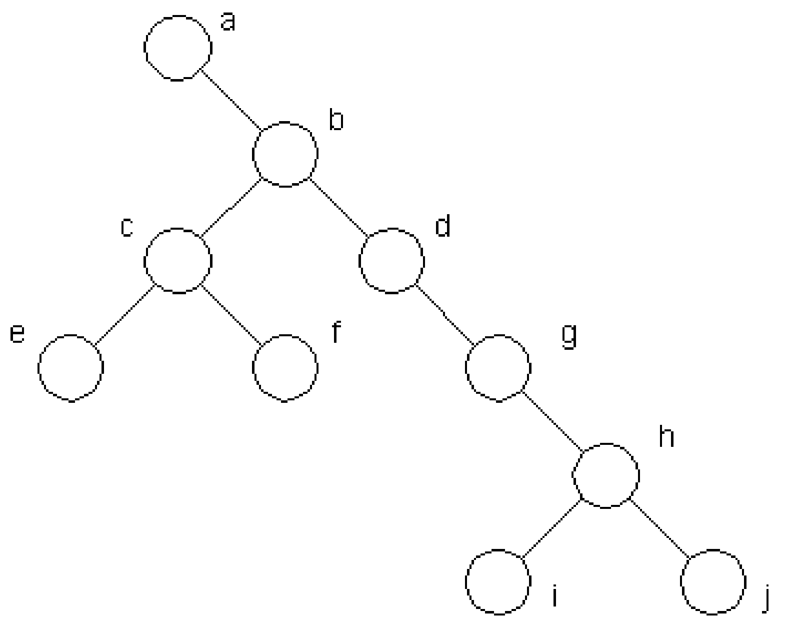
MARIO PINBALL LAND
GAME BOY ADVANCE, 2004

SUPER MARIO 64 DS
NINTENDO DS, 2004

More Specific Trees

We’ll focus on **binary trees**:

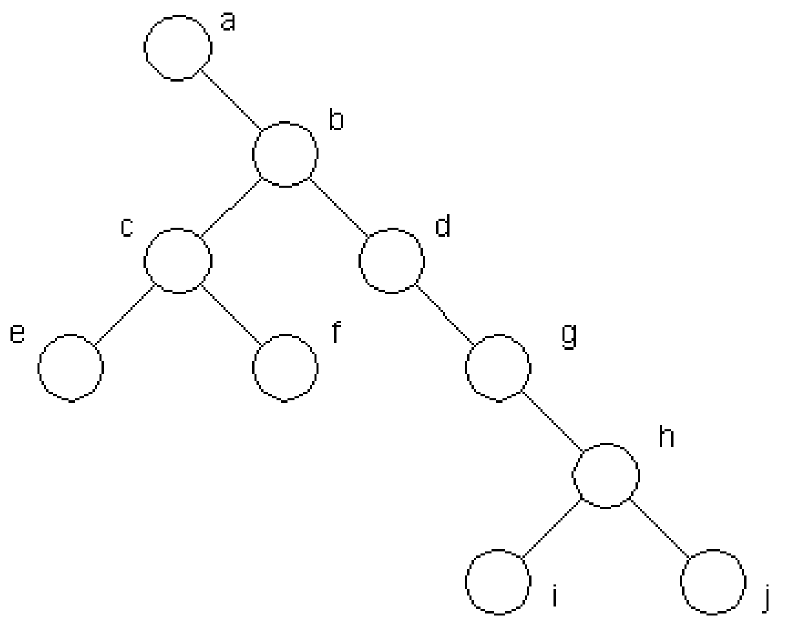
- A binary tree is **rooted** – every node can be reached via a path from the root



More Specific Trees

We’ll focus on **binary trees**:

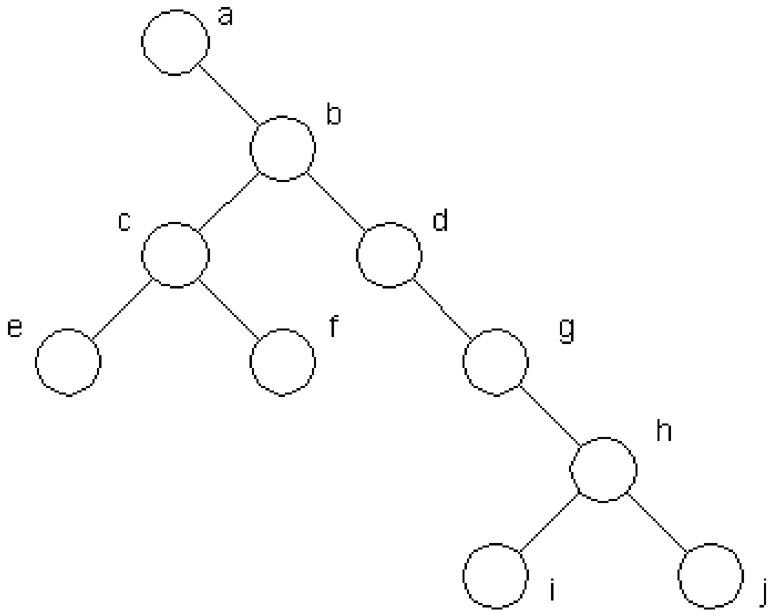
- A binary tree is **acyclic** – there are no cycles within the graph



More Specific Trees

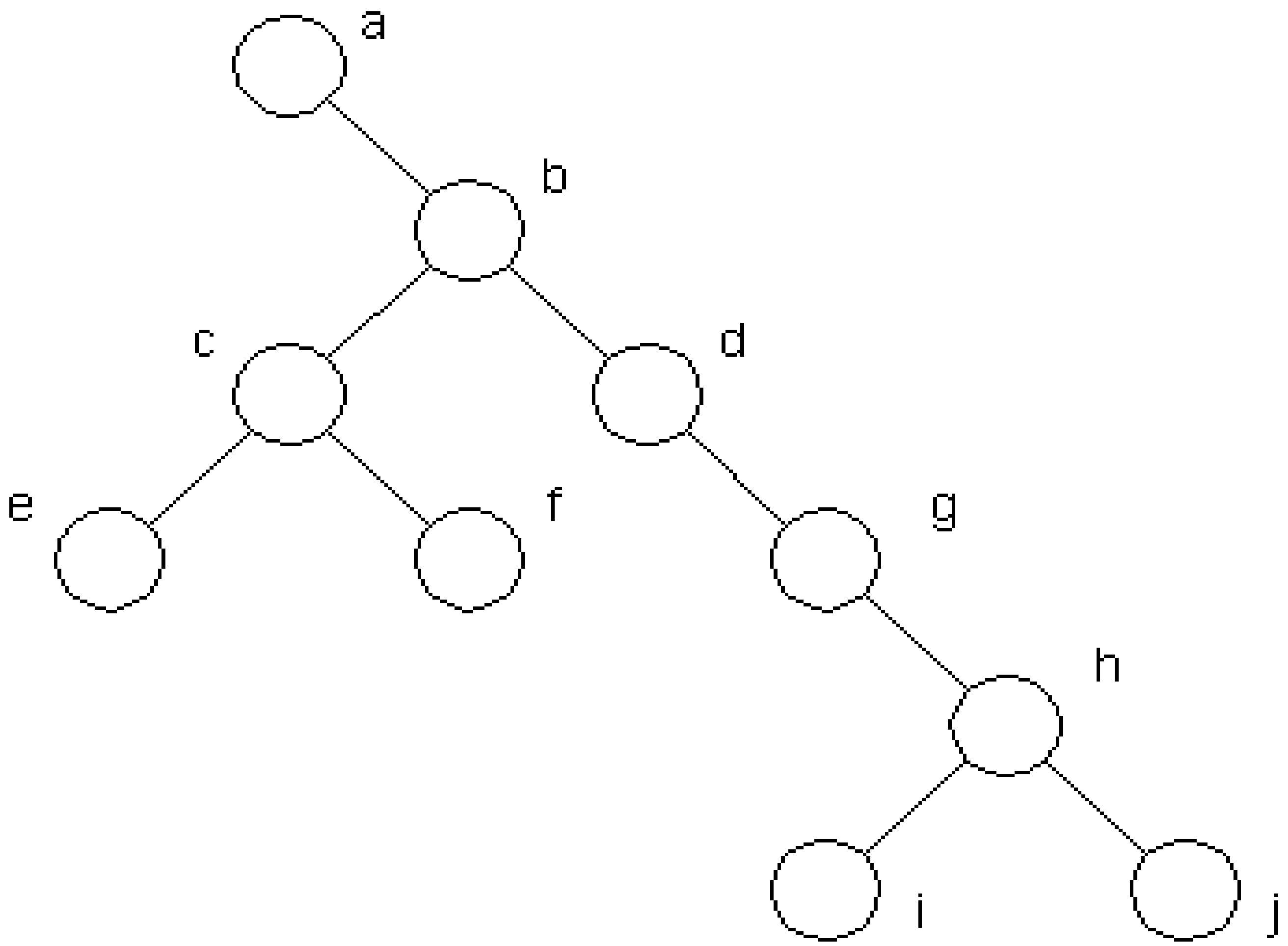
We’ll focus on **binary trees**:

- A binary tree contains **two or fewer children** – where one is the “left child” and one is the “right child”:



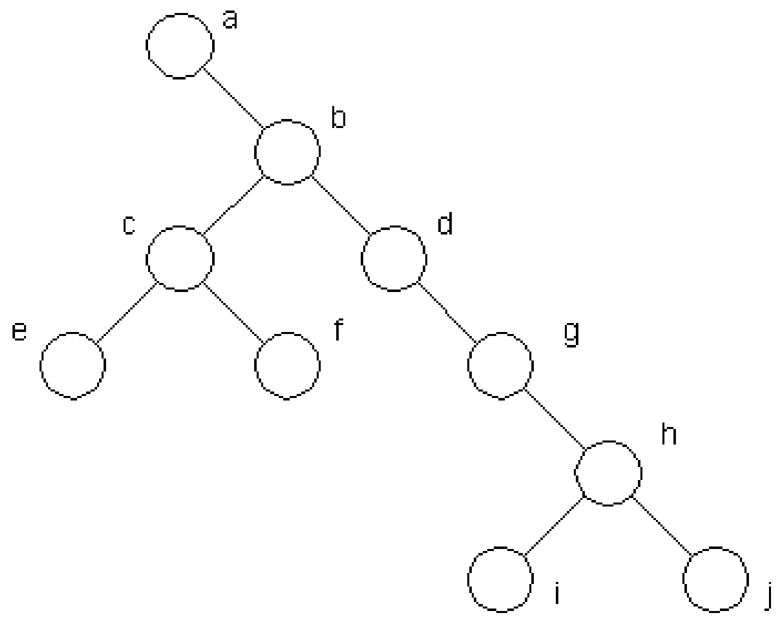
Tree Terminology

- What’s the longest “word” you can make using the vertex labels in the tree (repeats allowed)?



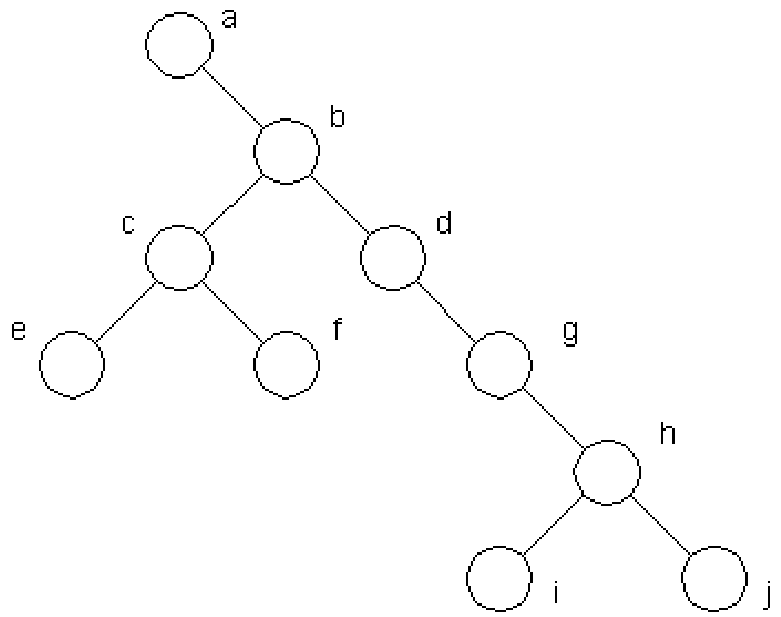
Tree Terminology

- Find an edge that is not on the longest path in the tree. Give that edge a reasonable name.
- One of the vertices is called the root of the tree. Which one?
- Make an "word" containing the names of the vertices that have a parent but no sibling.
- How many parents does each vertex have?
- Which vertex has the fewest children?
- Which vertex has the most ancestors?
- Which vertex has the most descendants?
- List all the vertices is b's left subtree.
- List all the leaves in the tree.



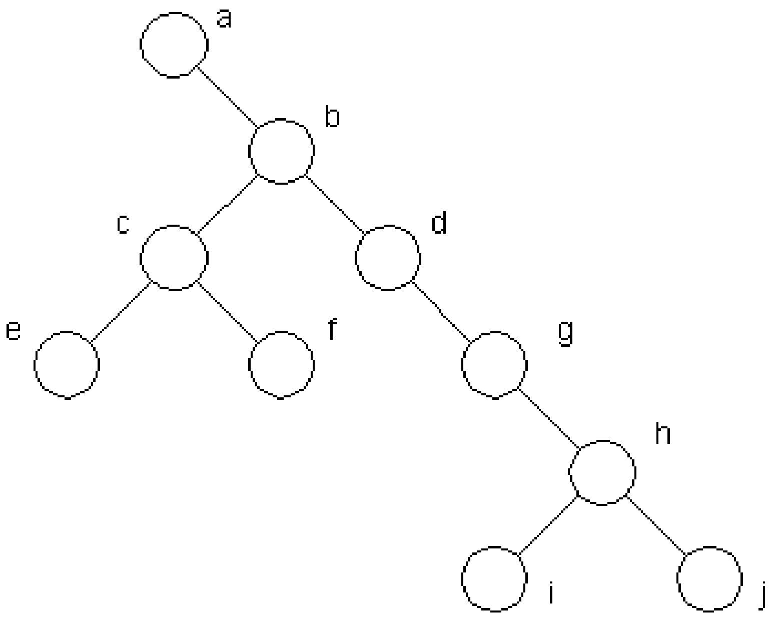
Tree Terminology

- What’s the longest “word” you can make using the vertex labels in the tree (repeats allowed)?
- Find an edge that is not on the longest path in the tree. Give that edge a reasonable name.
- One of the vertices is called the root of the tree. Which one?
- Make an “word” containing the names of the vertices that have a parent but no sibling.
- How many parents does each vertex have?
- Which vertex has the fewest children?
- Which vertex has the most ancestors?
- Which vertex has the most descendants?
- List all the vertices is b’s left subtree.
- List all the leaves in the tree.



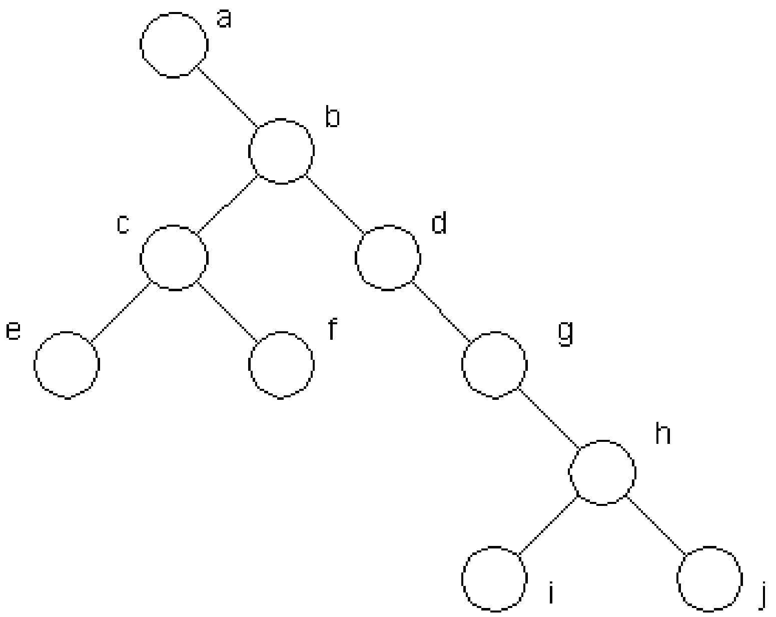
Tree Terminology

- What’s the longest “word” you can make using the vertex labels in the tree (repeats allowed)?
- Find an edge that is not on the longest path in the tree. Give that edge a reasonable name.
- One of the vertices is called the root of the tree. Which one?
- Make an “word” containing the names of the vertices that have a parent but no sibling.
- How many parents does each vertex have?
- Which vertex has the fewest children?
- Which vertex has the most ancestors?
- Which vertex has the most descendants?
- List all the vertices is b’s left subtree.
- List all the leaves in the tree.



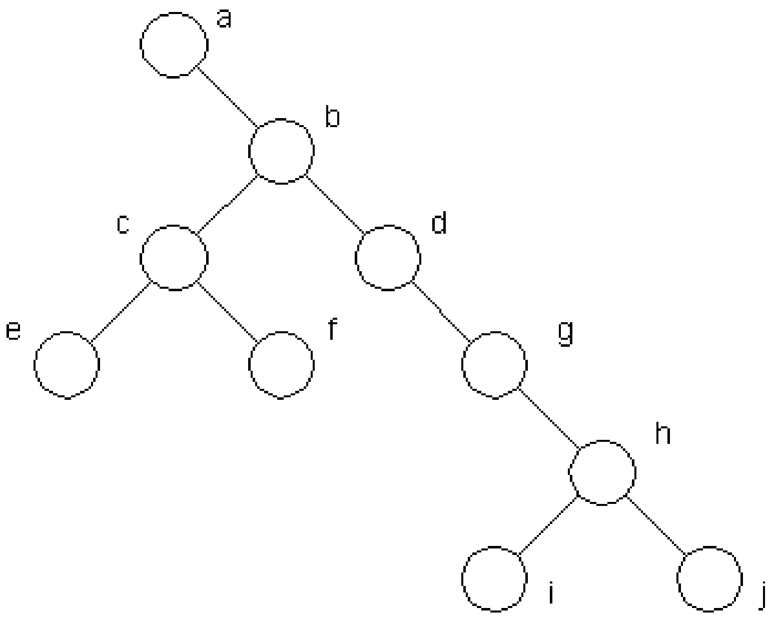
Tree Terminology

- What's the longest "word" you can make using the vertex labels in the tree (repeats allowed)?
- Find an edge that is not on the longest path in the tree. Give that edge a reasonable name.
- One of the vertices is called the root of the tree. Which one?
- Make an "word" containing the names of the vertices that have a parent but no sibling.
- How many parents does each vertex have?
- Which vertex has the fewest children?
- Which vertex has the most ancestors?
- Which vertex has the most descendants?
- List all the vertices is b's left subtree.
- List all the leaves in the tree.



Tree Terminology

- What’s the longest “word” you can make using the vertex labels in the tree (repeats allowed)?
- Find an edge that is not on the longest path in the tree. Give that edge a reasonable name.
- One of the vertices is called the root of the tree. Which one?
- Make an “word” containing the names of the vertices that have a parent but no sibling.
- How many parents does each vertex have?
- Which vertex has the fewest children?
- Which vertex has the most ancestors?
- Which vertex has the most descendants?
- List all the vertices is b’s left subtree.
- List all the leaves in the tree.



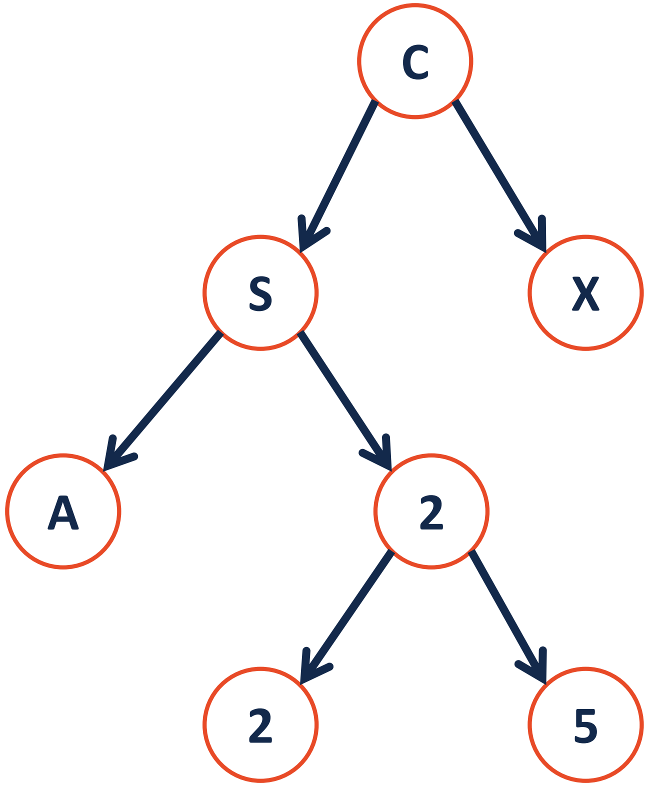
Binary Tree – Defined

A *binary tree* T is either:

•

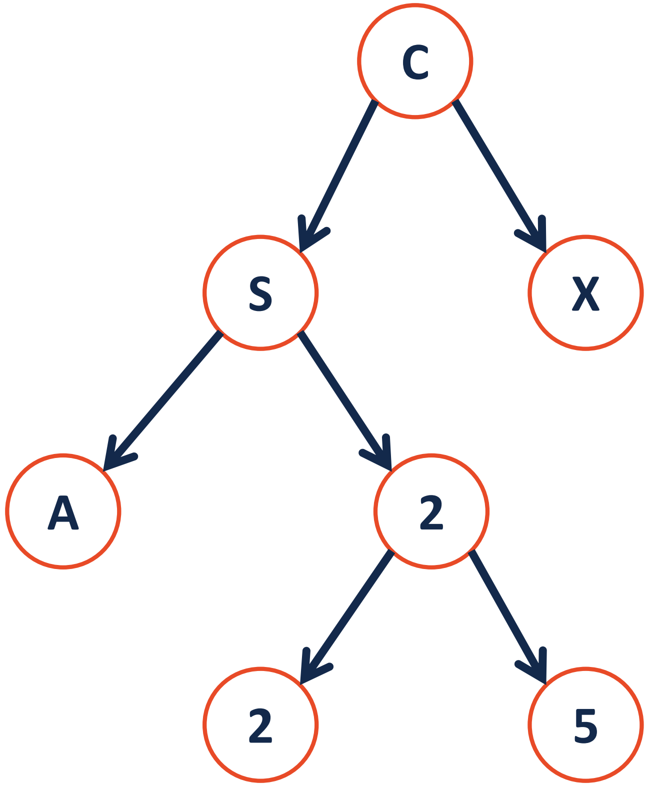
OR

•



Tree Property: height

height(T): length of the longest path from the root to a leaf



Given a binary tree T:

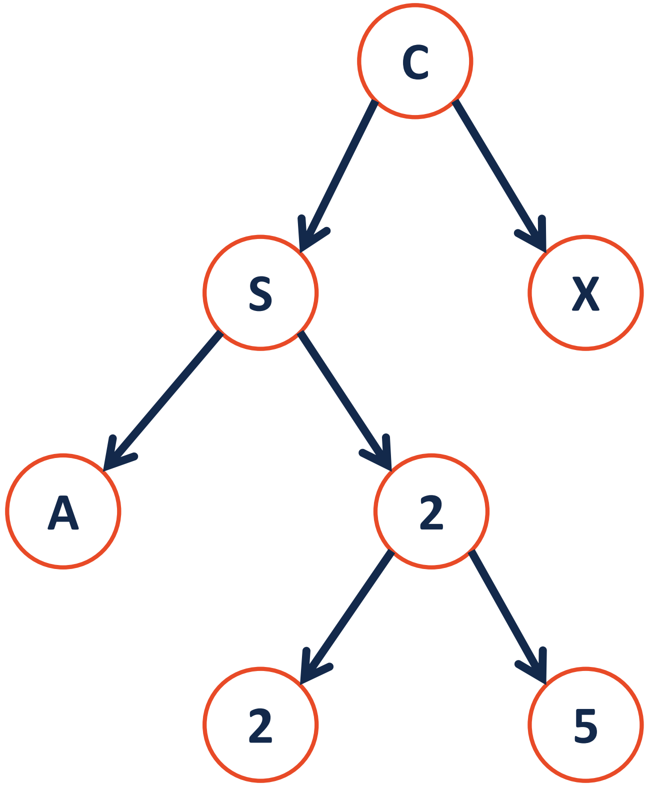
height(T) =

Tree Property: full

A tree ***F*** is **full** if and only if:

1.

2.

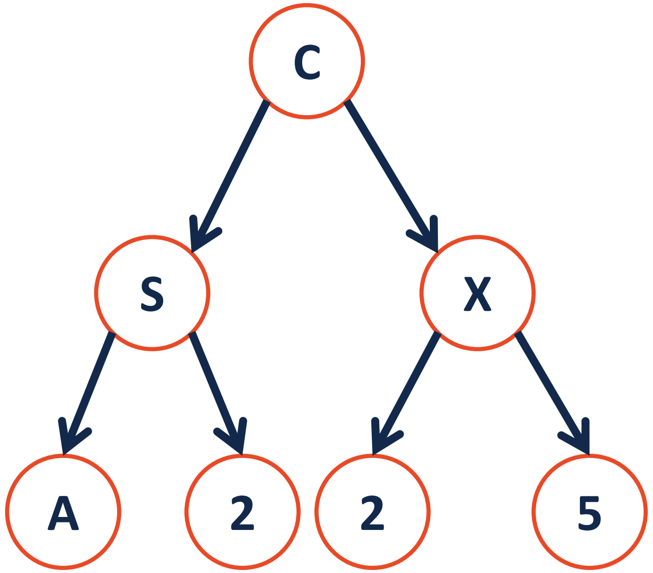


Tree Property: perfect

A **perfect** tree ***P*** is:

1.

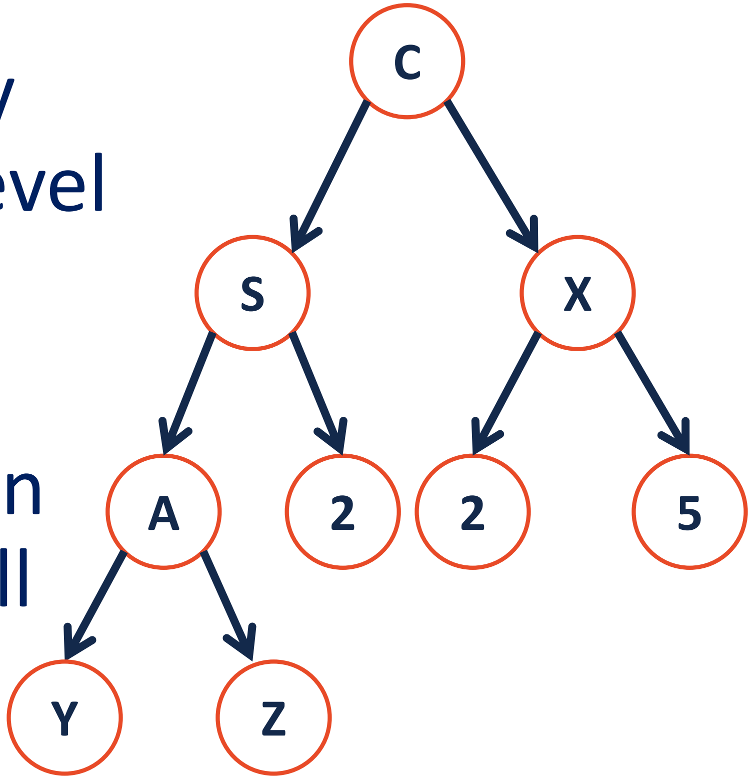
2.



Tree Property: complete

Conceptually: A perfect tree for every level except the last, where the last level if "pushed to the left".

Slightly more formal: For any level k in [0, h-1], k has 2^k nodes. For level h, all nodes are "pushed to the left".



Tree Property: complete

A **complete** tree ***C*** of height **h**, $\mathbf{C_h}$:

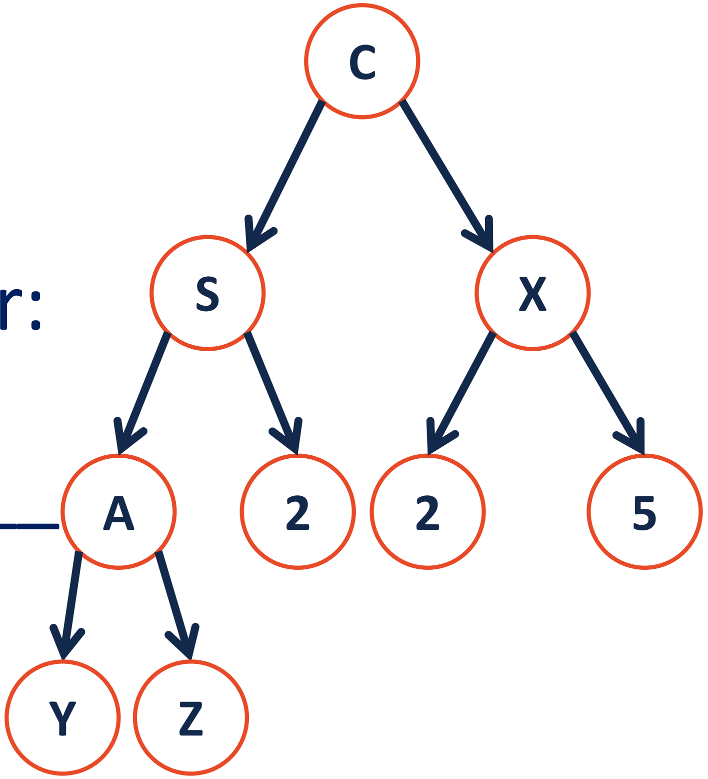
1. $\mathbf{C_{-1}}$ = {}

2. $\mathbf{C_h}$ *(where h>0)* **= {r, $\mathbf{T_L}$, $\mathbf{T_R}$}** and either:

T_L is ____________ and T_R is ___________

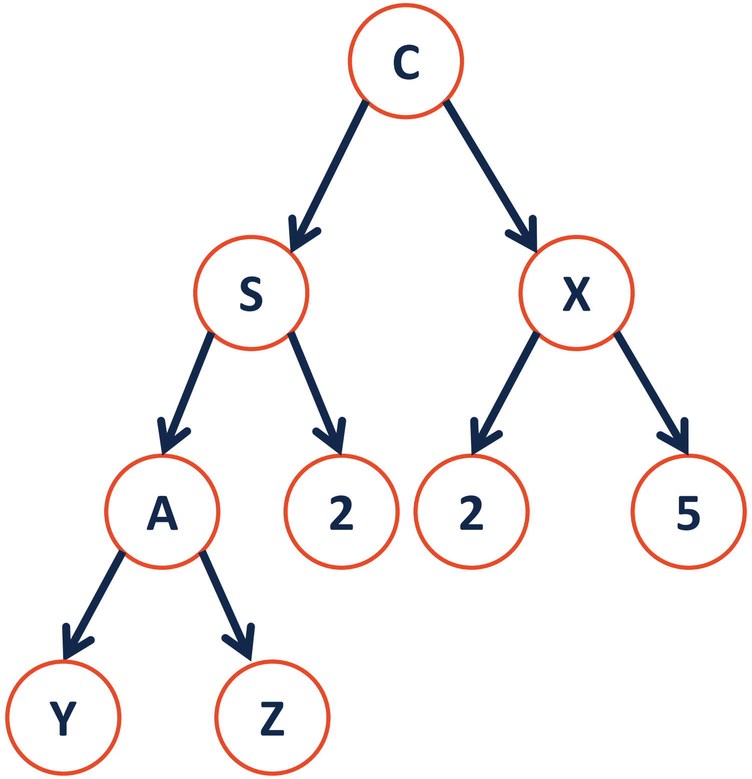
OR

T_L is ____________ and T_R is ___________



Tree Property: complete

Is every **full** tree **complete**?



If every **complete** tree **full**?