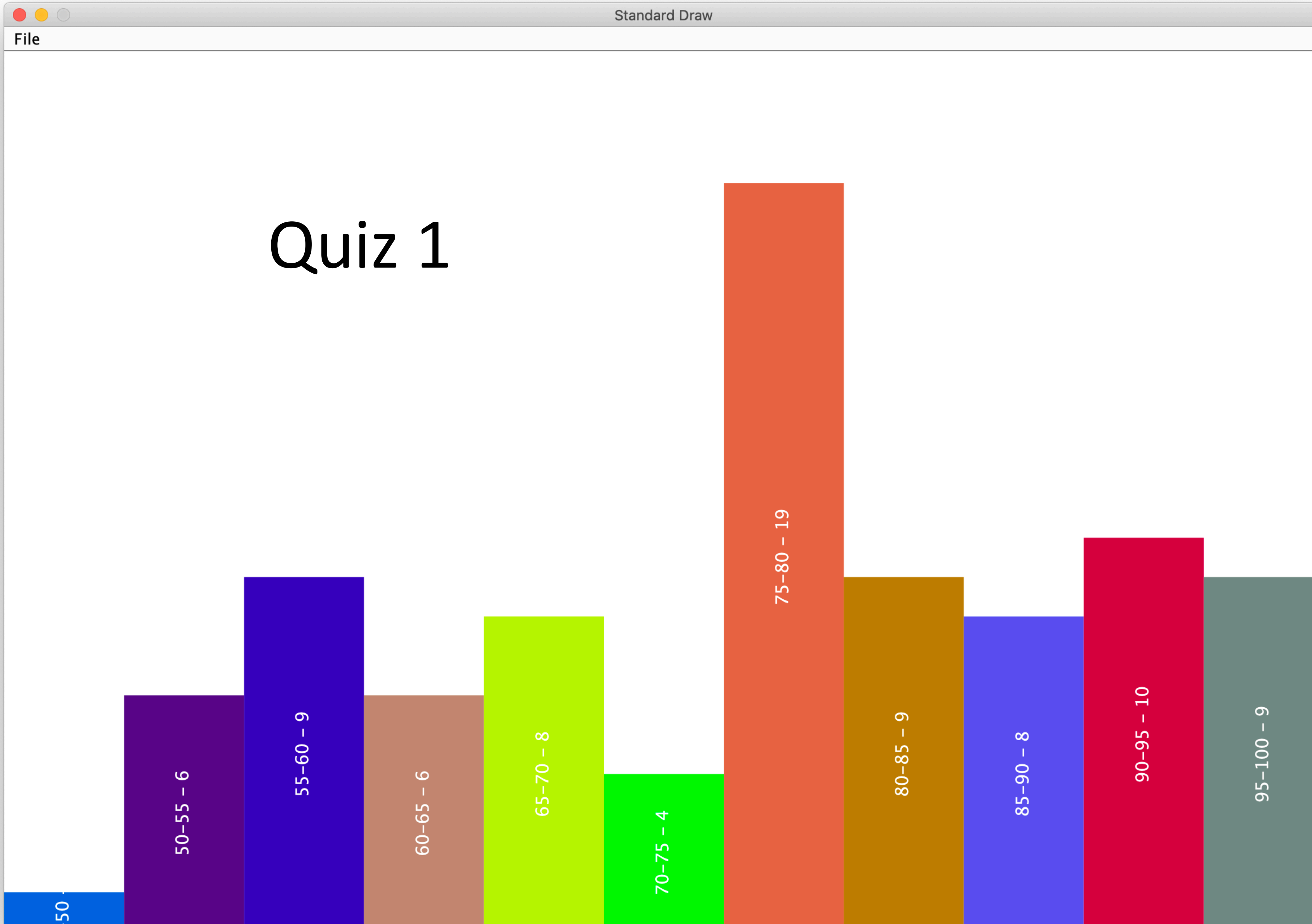
CSCI 1102 Computer Science 2

Meeting 11: Tuesday 3/9/2021

Priority Queues; Binary Heaps


Standard Draw
File
Quiz 1
–50
50–55 – 6
55–60 – 9
60–65 – 6
65–70 – 8
70–75 – 4
75–80 – 19
80–85 – 9
85–90 – 8
90–95 – 10
95–100 – 9

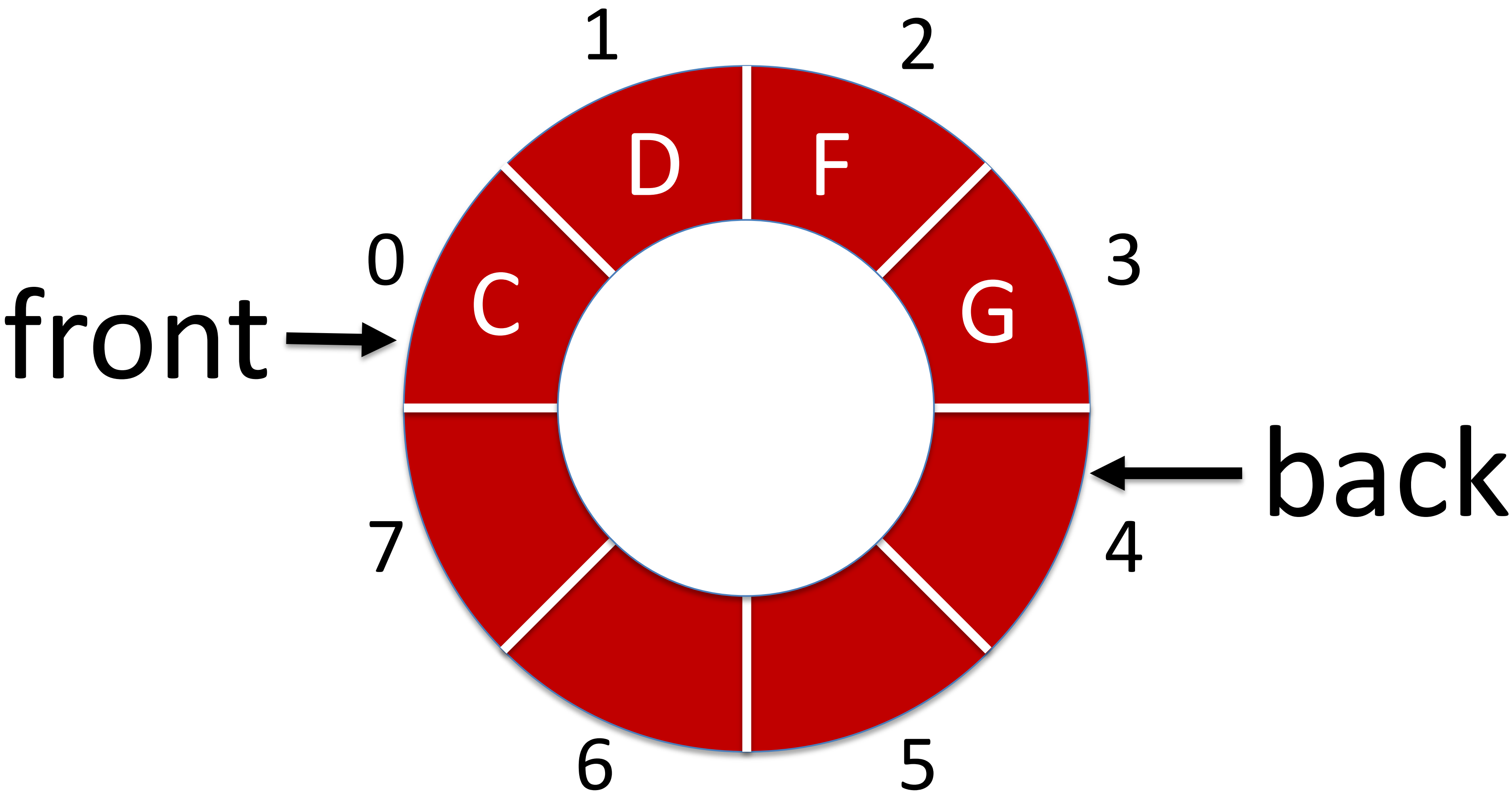
Order Sensitive Data Structures

Example

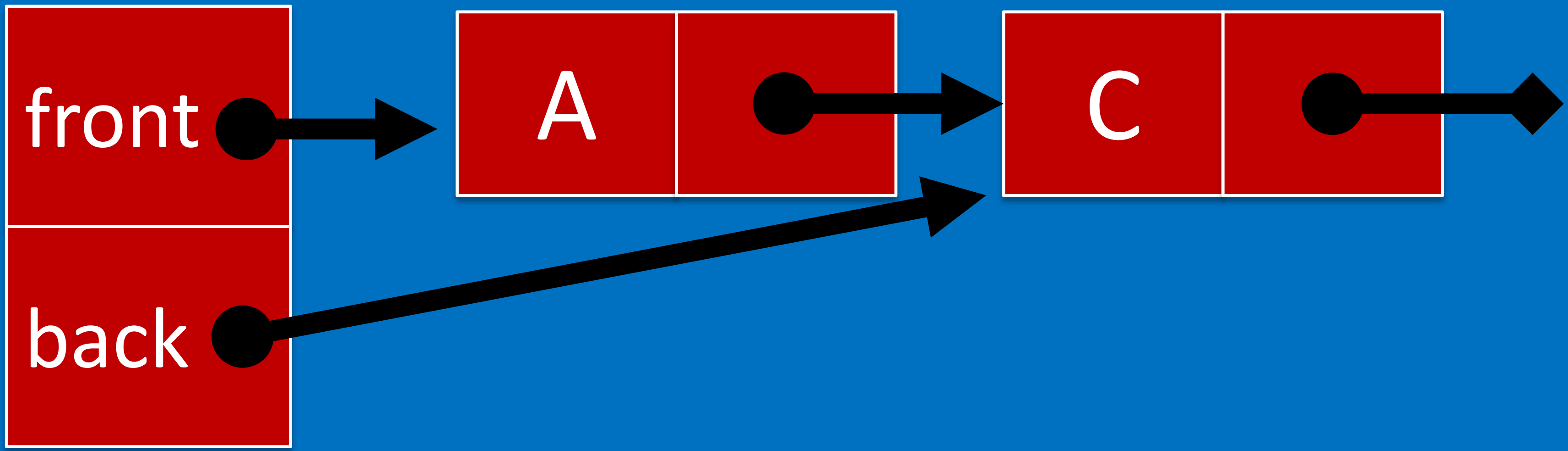
- There are ~10^{60} small molecules that might be therapeutic drugs;
- Only a small handful can be tested *in vitro* in a lab setting;
- Many fewer still can be tested *in vivo* or in clinical trials;
- Rank these candidates from highest to lowest priority, place in a Maximum Priority Queue (MaxPQ)

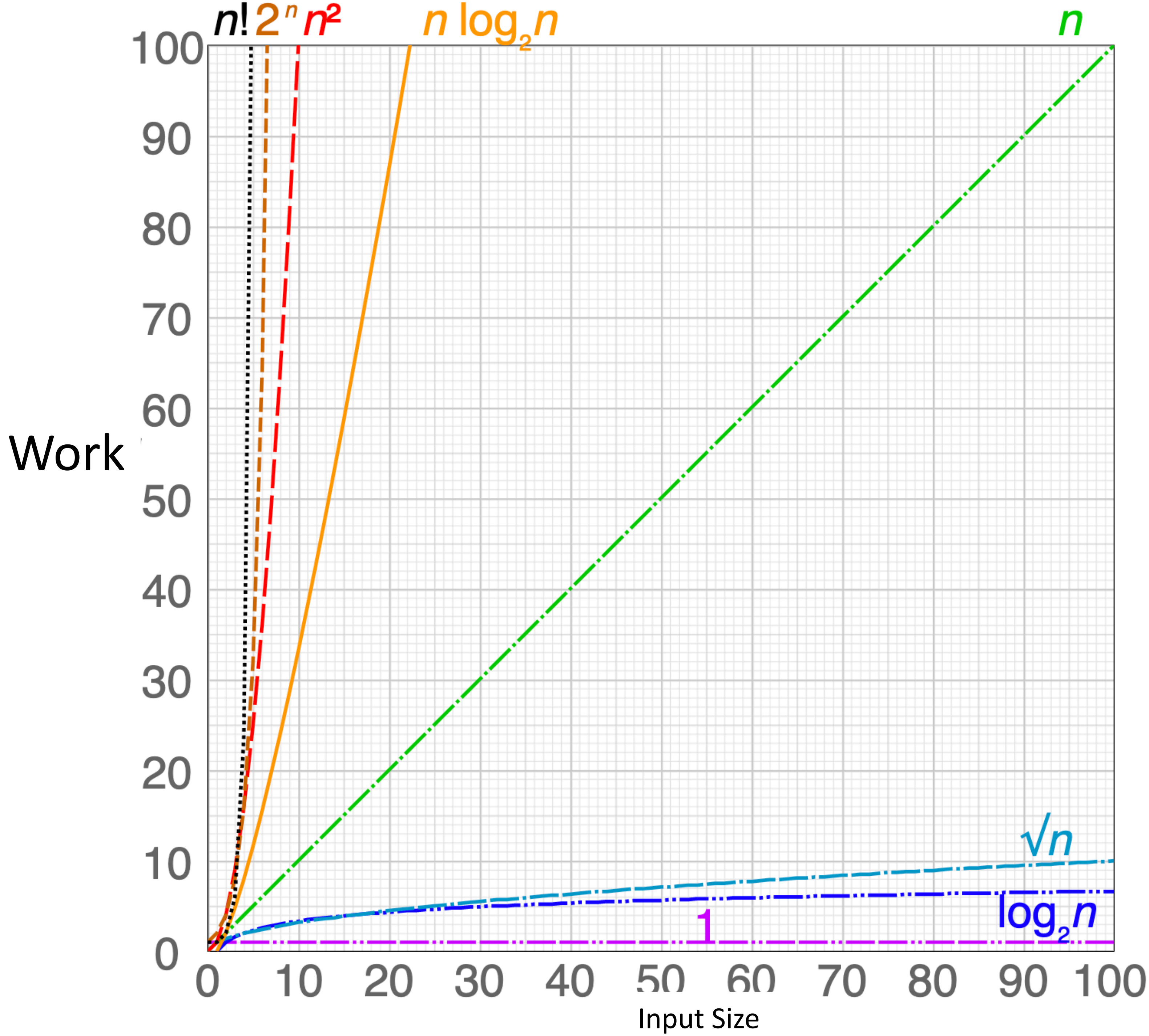
Simple Linked or Sequential Allocation lead to *linear time* enqueue.

enqueue(E)



enqueue(B)





$n!$ 2^n n^2 $n \log_2 n$ n
$\sqrt{n}$
$\log_2 n$
1
100
90
80
70
60
50
40
30
20
10
0
0 10 20 30 40 50 60 70 80 90 100
Work
Input Size

```java
public interface PriorityQueue<T extends Comparable<T>> {
    void enqueue(T item);

    T dequeue();

    boolean isEmpty();

    int size();

    String toString();
}
```

Trees

Symbol

$A_2 \quad B_1 \quad C_0$

Arity

S is a Set of Symbols

$$S = \{A_{2,}\ \ B_{1,}\ \ C_0\}$$

Trees(S)

$$\text{Trees}(S) = \{\ A_k(t_1, \ldots, t_k) \mid k >= 0,\ A_k \text{ in } S \text{ and } t_i \text{ in Trees}(S)\ \}$$

Trees are defined recursively.

Example

$$\text{Trees}(\{A_2, \; B_1, \; C_0\}) = \{\; C_0(), \\ B_1(C_0()), \\ A_2(C_0(), C_0()), \\ A_2(B_1(C_0()), C_0()), \\ \ldots \\ \}$$

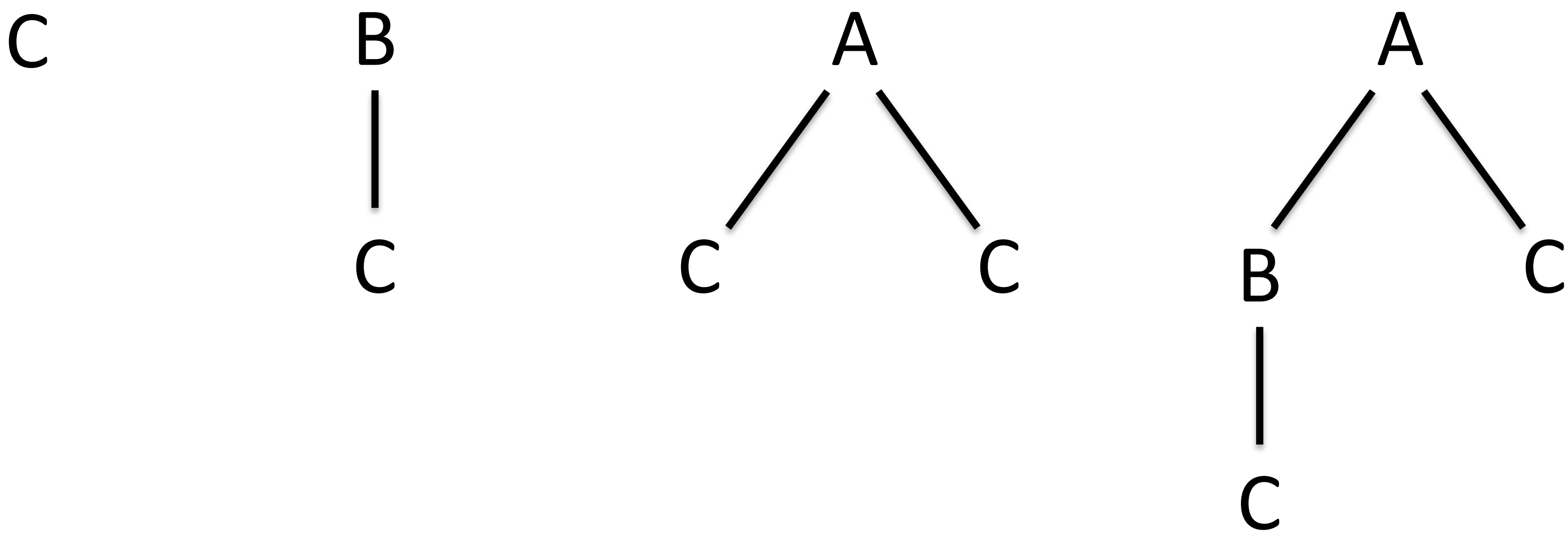
Simplifying the Notation

- If the arity is 0, leave out the parentheses. So $C_0()$ is written as C_0.

- Omit the arities

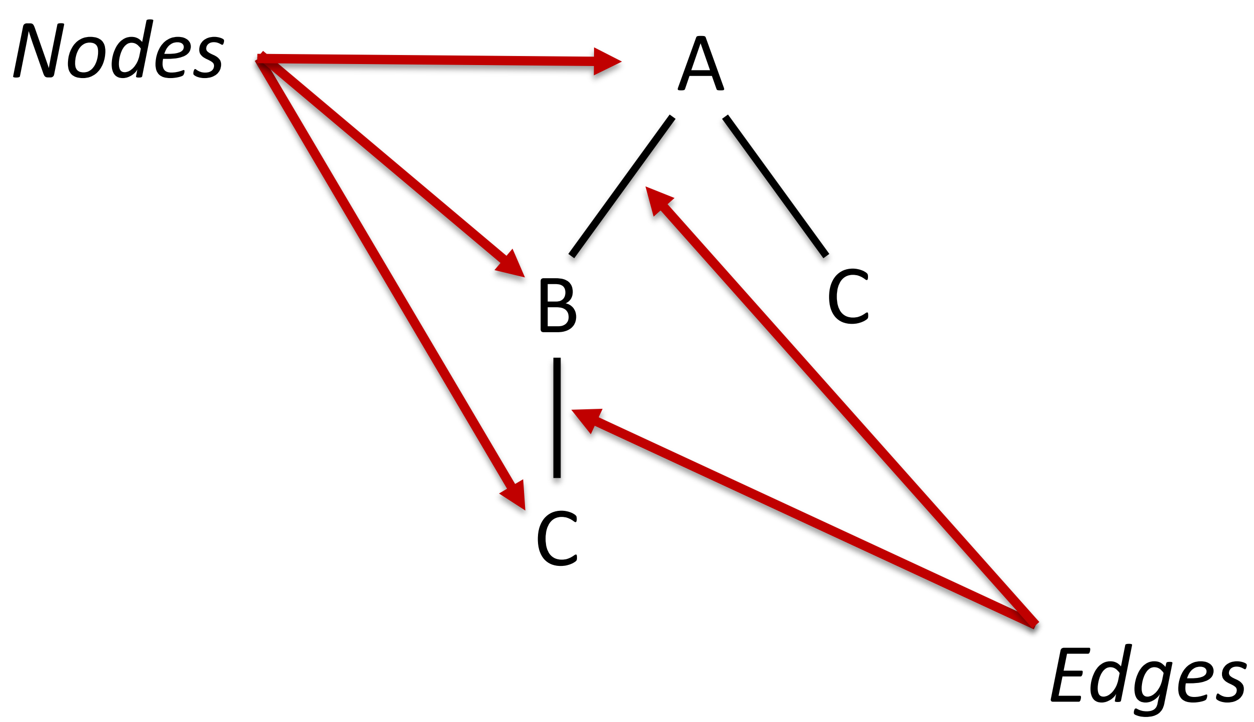
$$\text{Trees}(\{A_{2,}\ \ B_{1,}\ \ C_0\}) = \{C, B(C), A(C, C), A(B(C), C), \ldots\}$$

Tree Diagrams

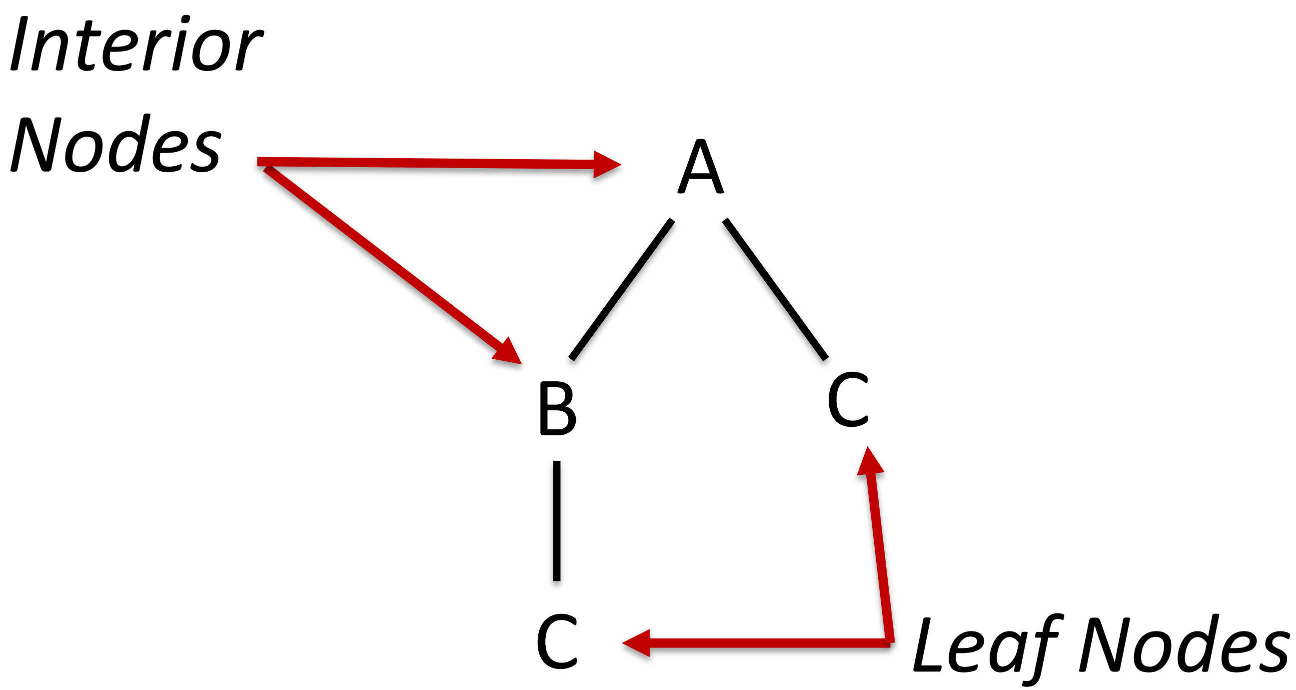


C B(C) A(C, C) A(B(C), C)

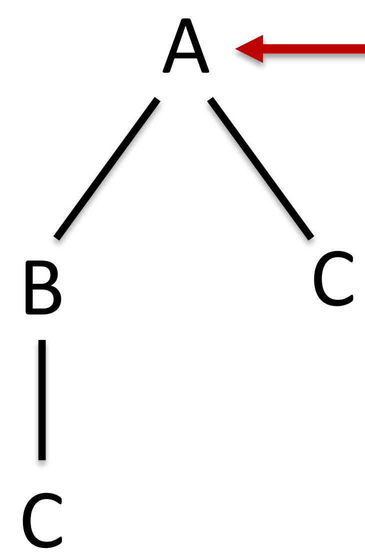
Terminology



Terminology

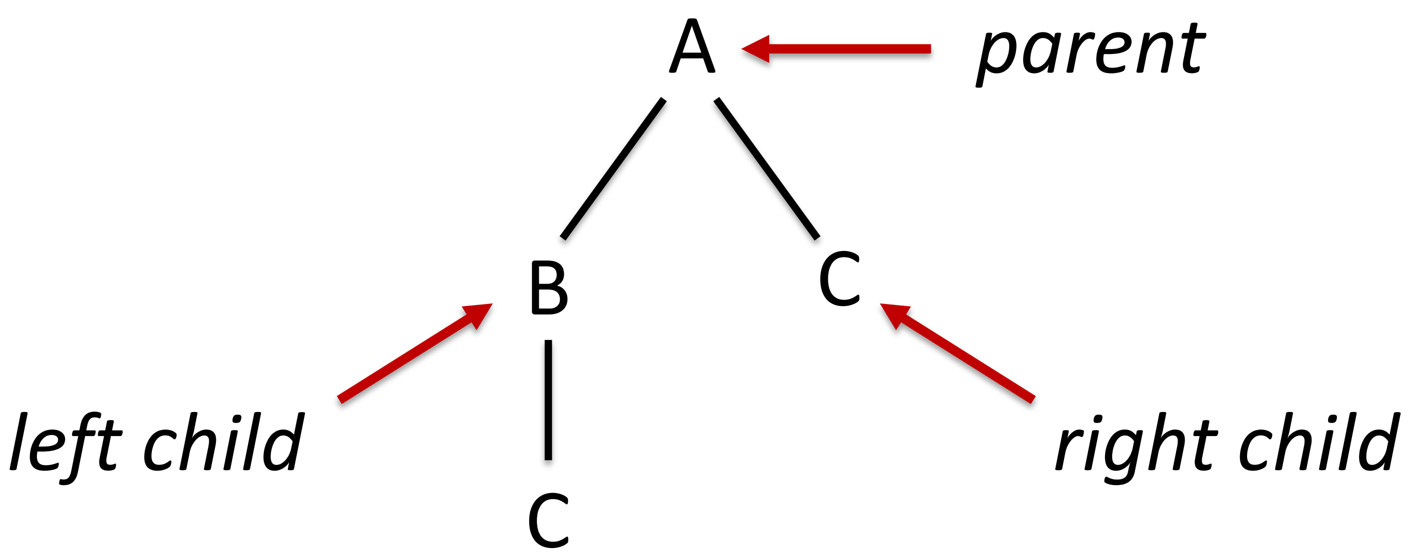


Terminology

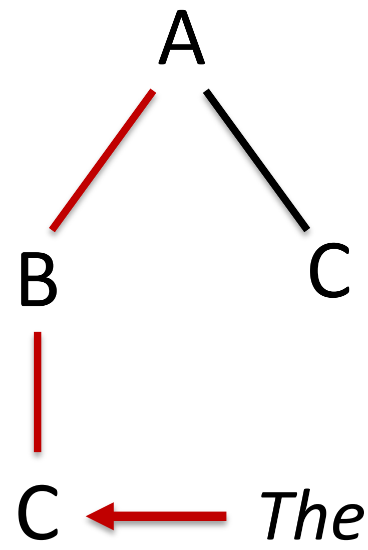


A unique ***root*** *node*

Terminology

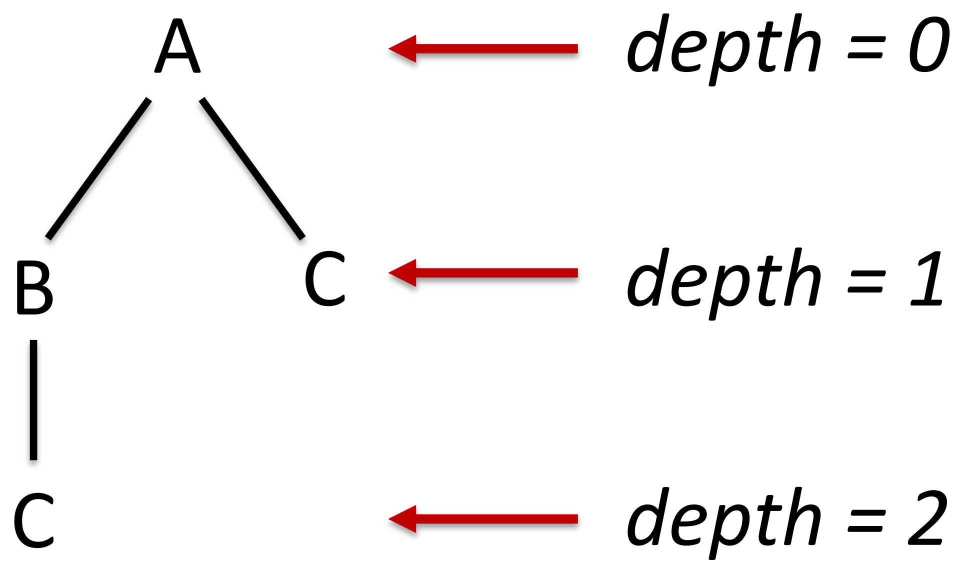


A *path* is a Sequence of Edges

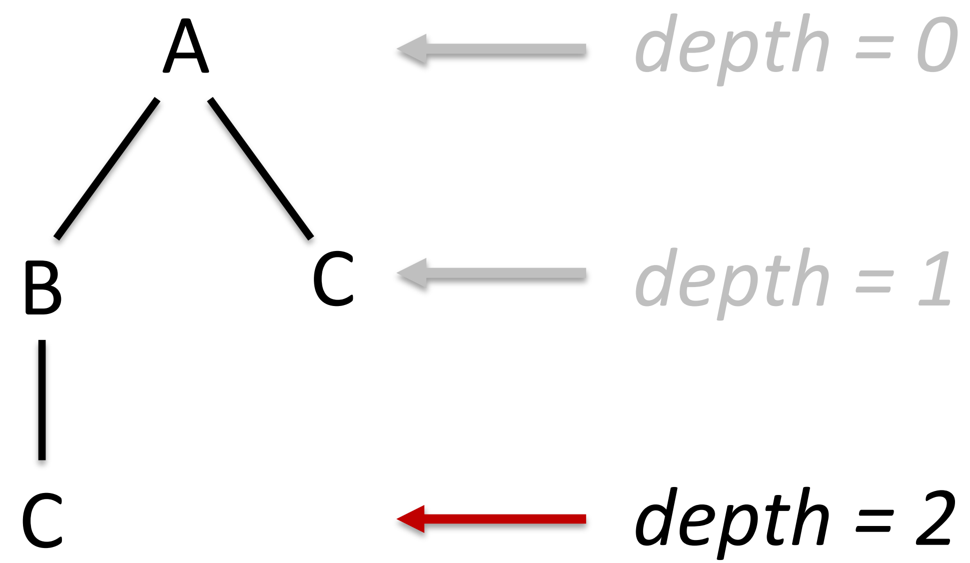


The path from the root to this node is of length 2.

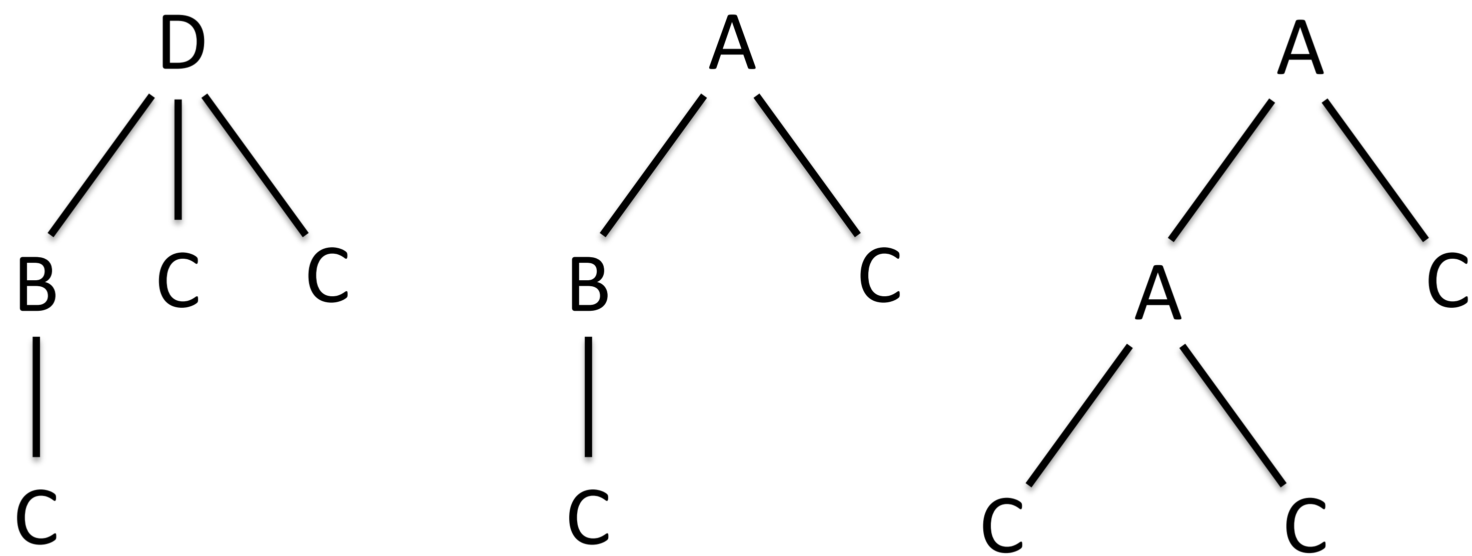
Depth of a Node – length of path from root



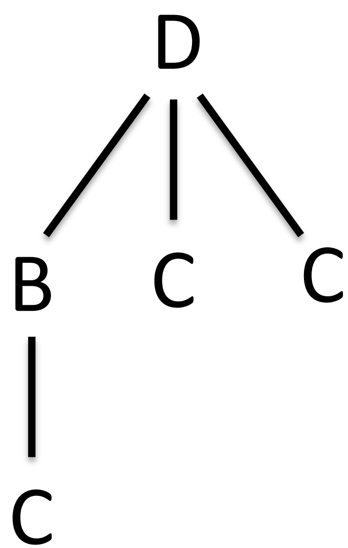
Height of a Tree – maximum depth

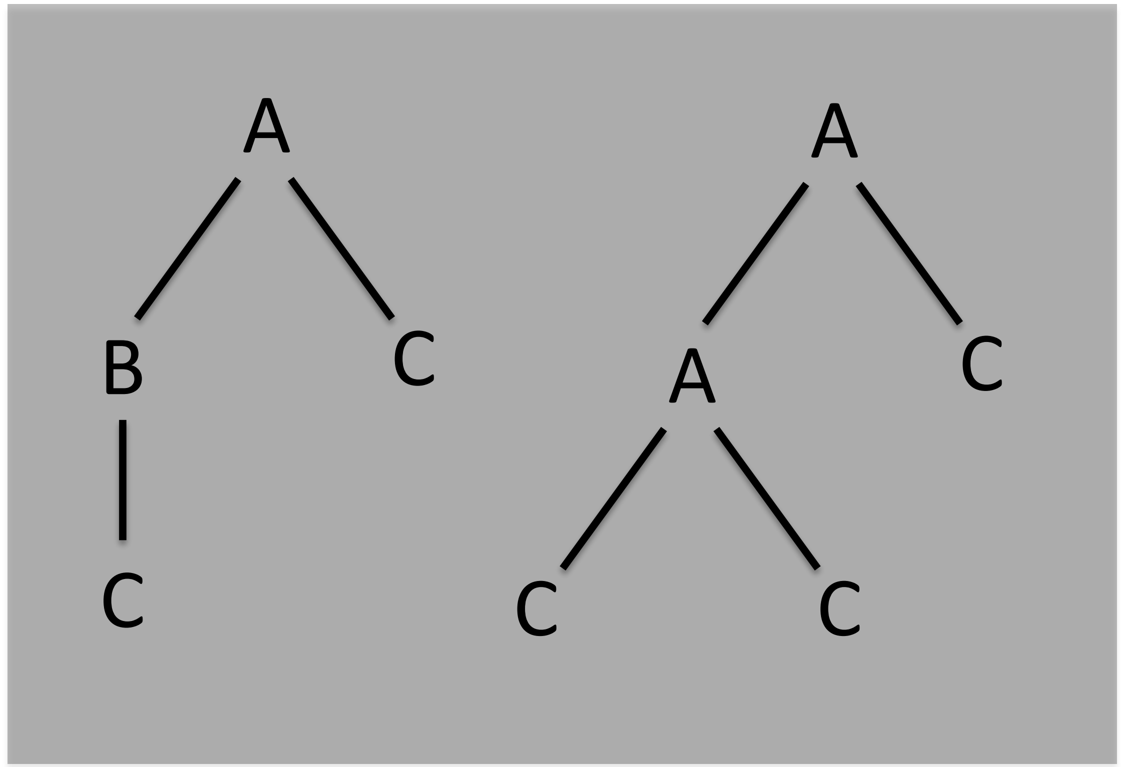


Binary Tree – Maximum arity is 2

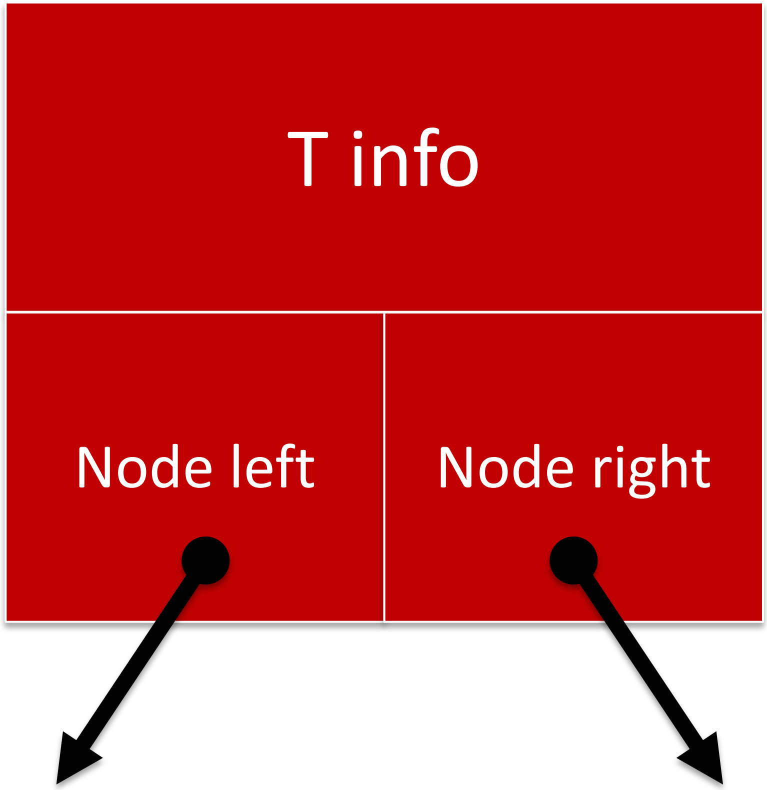


Binary Tree – Maximum arity is 2

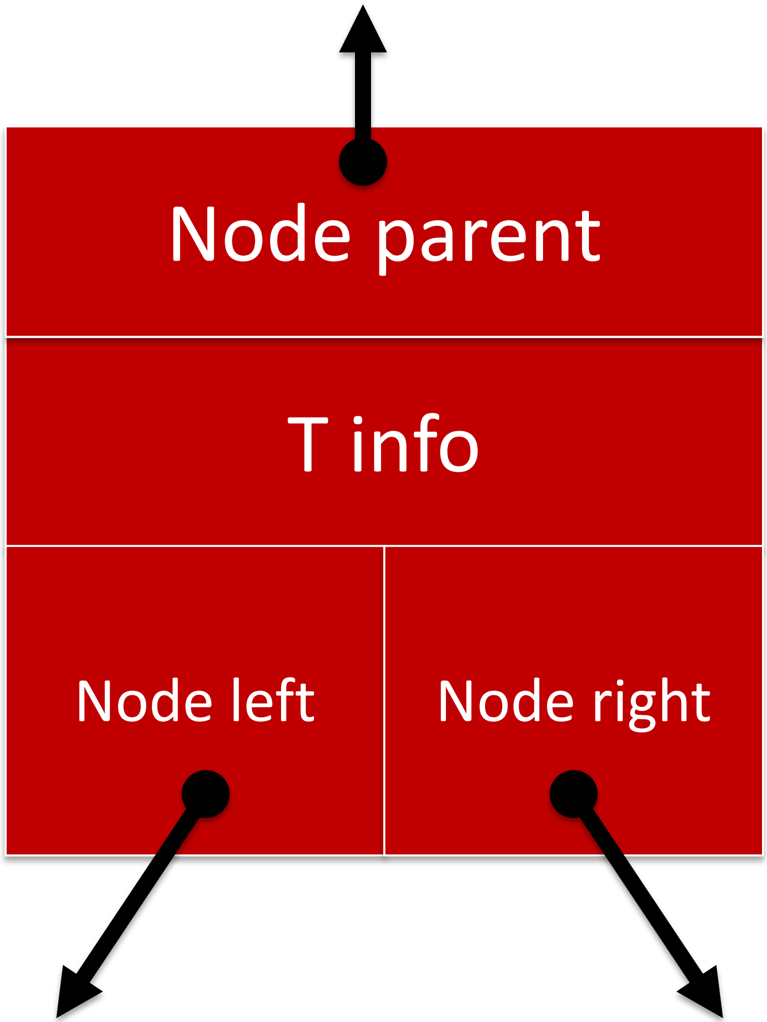




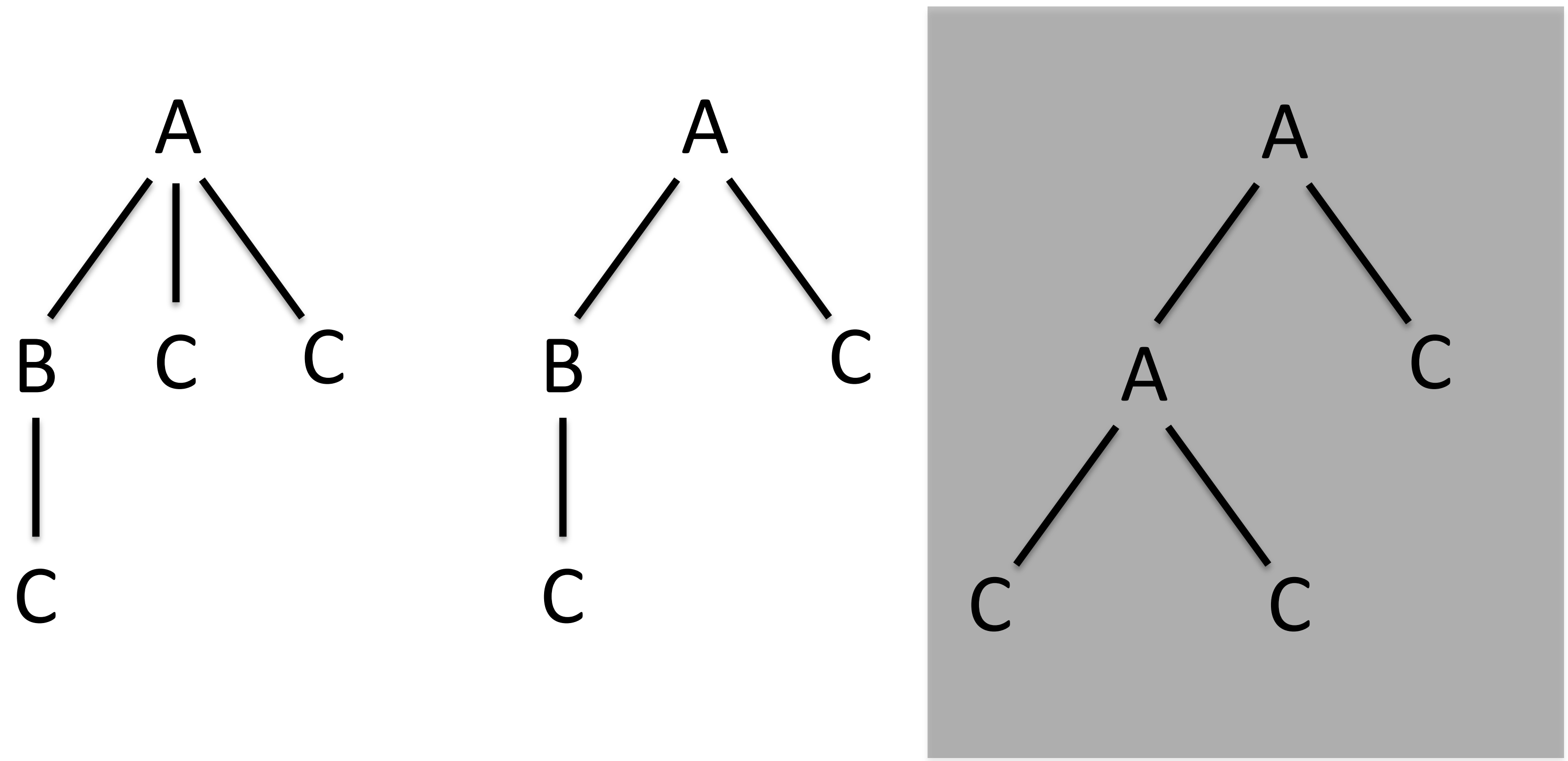
Representations of Nodes in Binary Trees



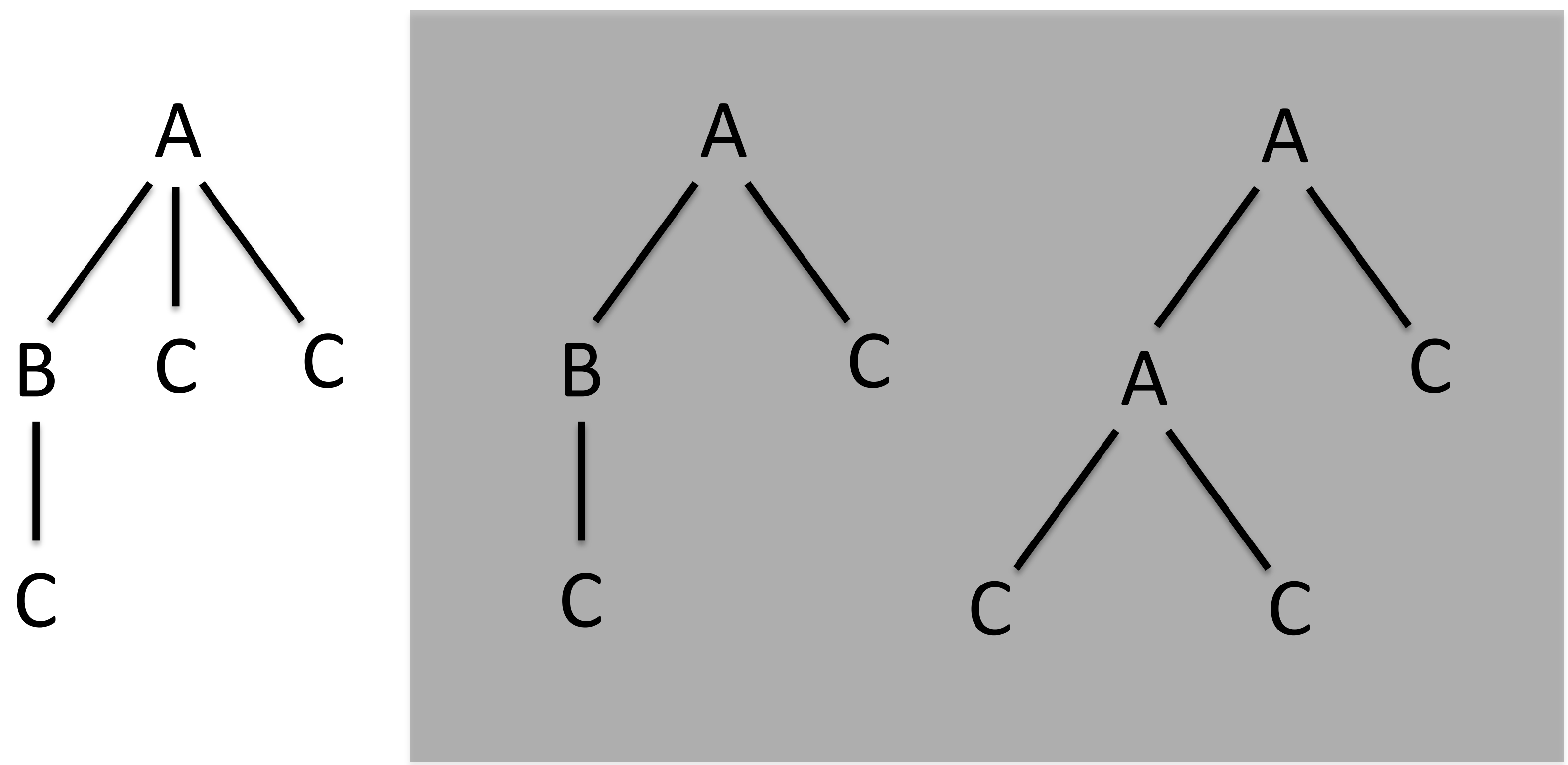
Threaded Representation of Nodes



Full Binary Tree – arity of every interior node = 2

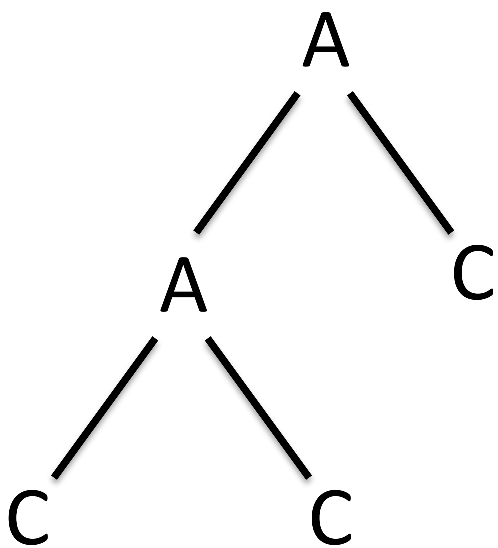


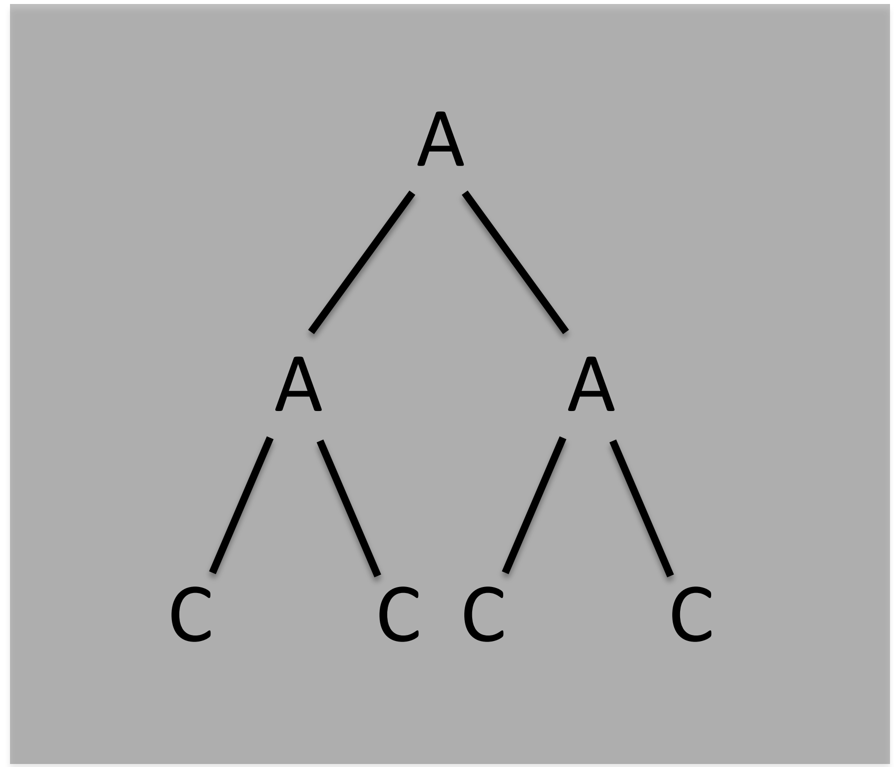
Complete Binary Tree – All depths are full except possibly the last one, then all to the left



The middle tree is complete but not full.

Perfect Binary Tree – All depths are full

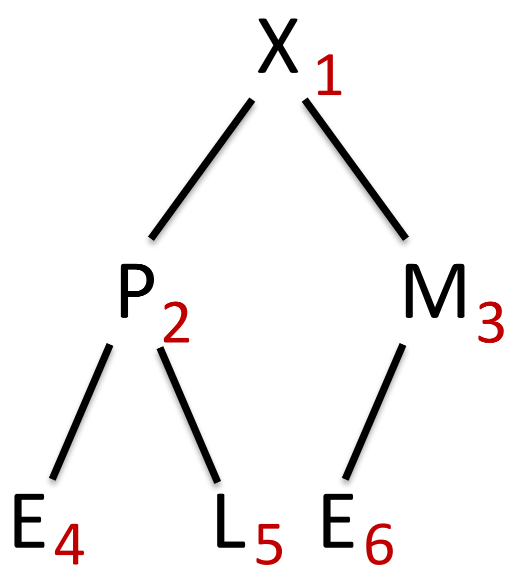




We'll discuss perfect binary trees next time.

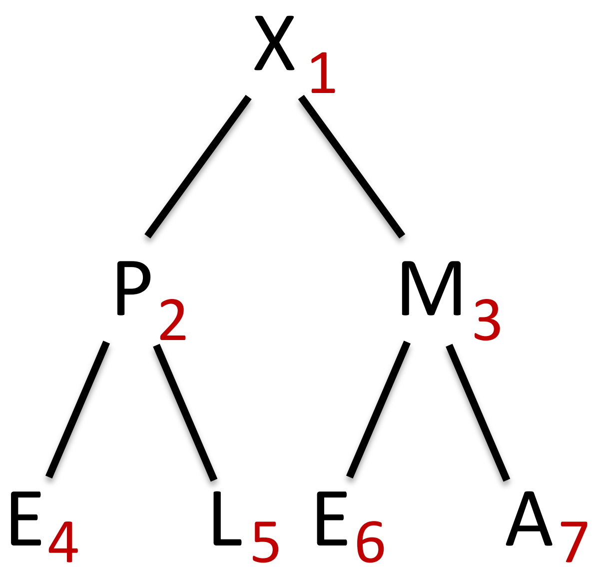
Complete binary trees are of interest because they have

1. a natural sequential representation
2. logarithmic height



0	-
1	X
2	P
3	M
4	E
5	L
6	E

Navigation in Complete Binary Trees



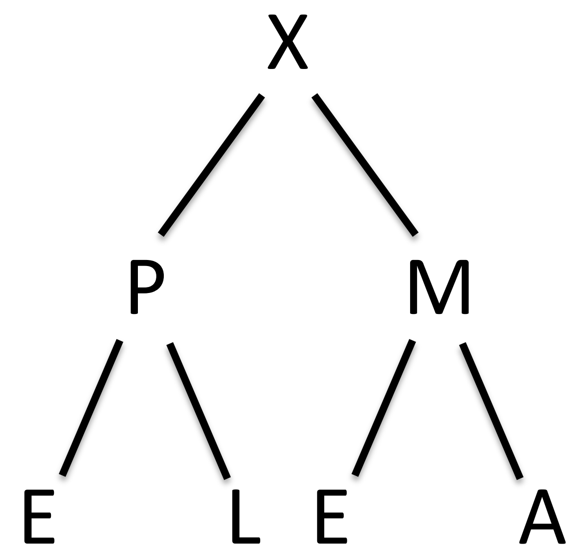
parent(N) = N / 2

leftChild(N) = N * 2

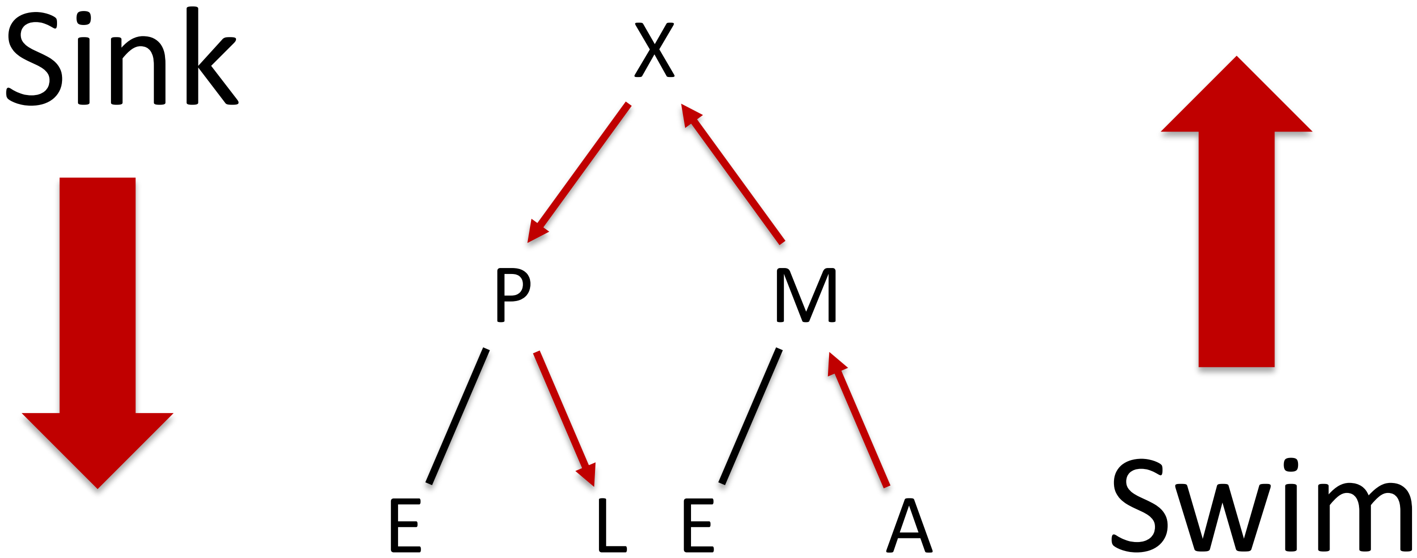
rightChild(N) = N * 2 + 1

Binary Heaps

A (max) *binary heap* is a complete binary tree in which the value at every interior node is greater than or equal to the values in the child nodes.

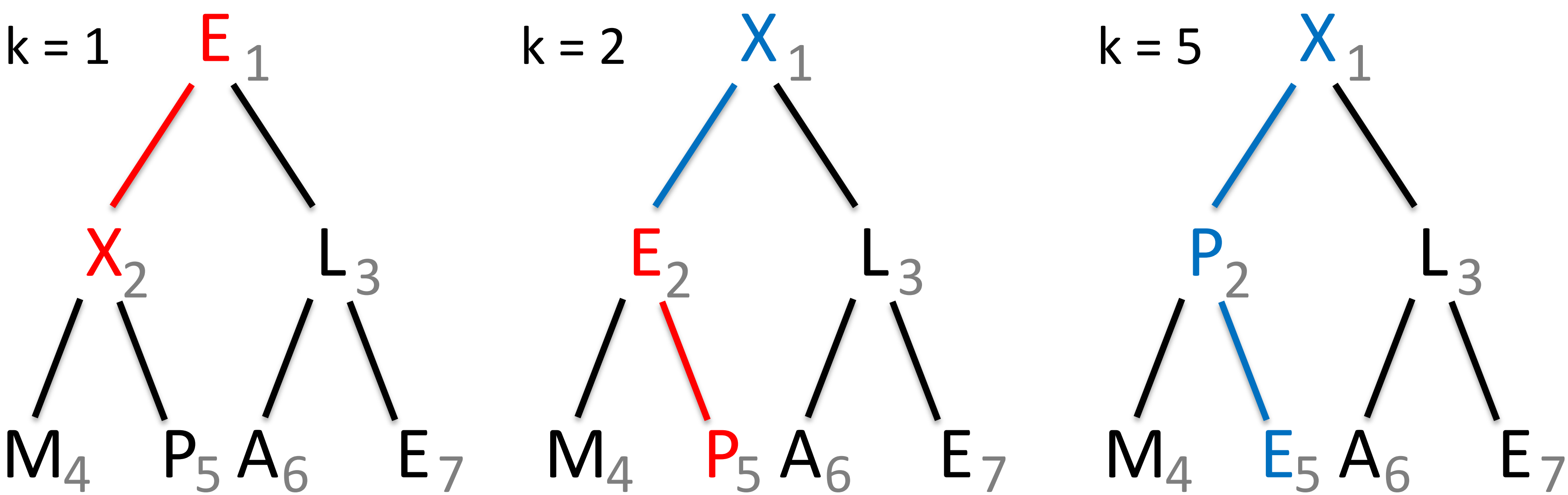


Migrating Values along Paths in a Heap



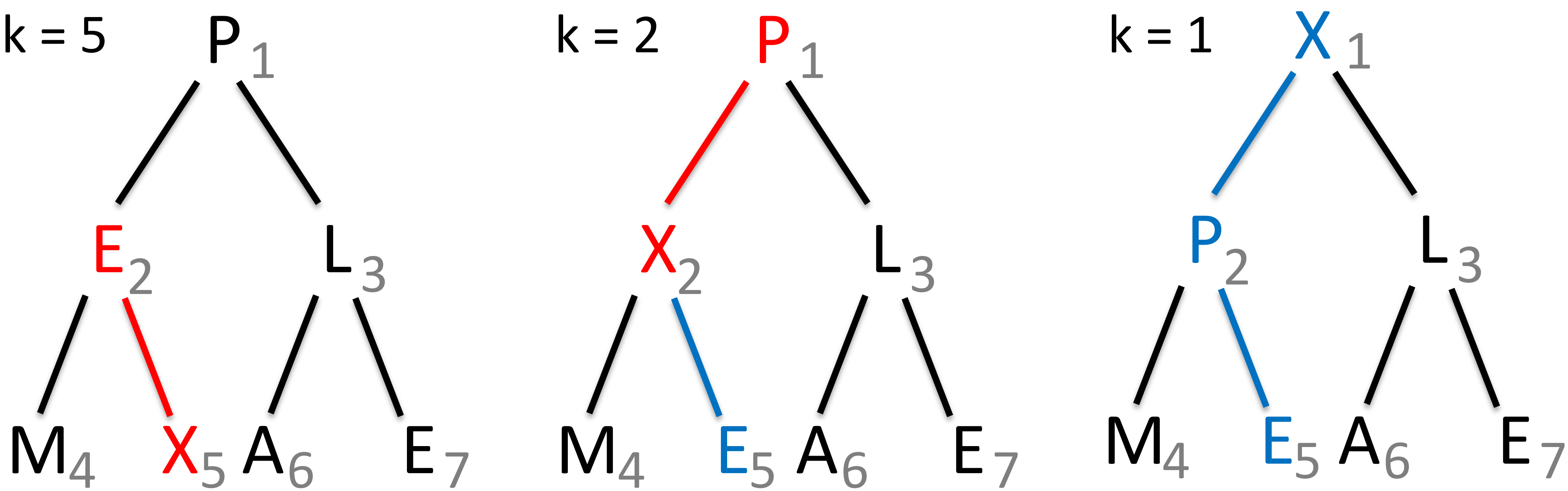
```java
private void sink(int k) {
    while (2 * k <= N) {
        int j = 2 * k;
        if (j < N && less(j, j + 1)) j++;
        if (!less(k, j)) break;
        exch(k, j);
        k = j;
    }
}
```

Sinking



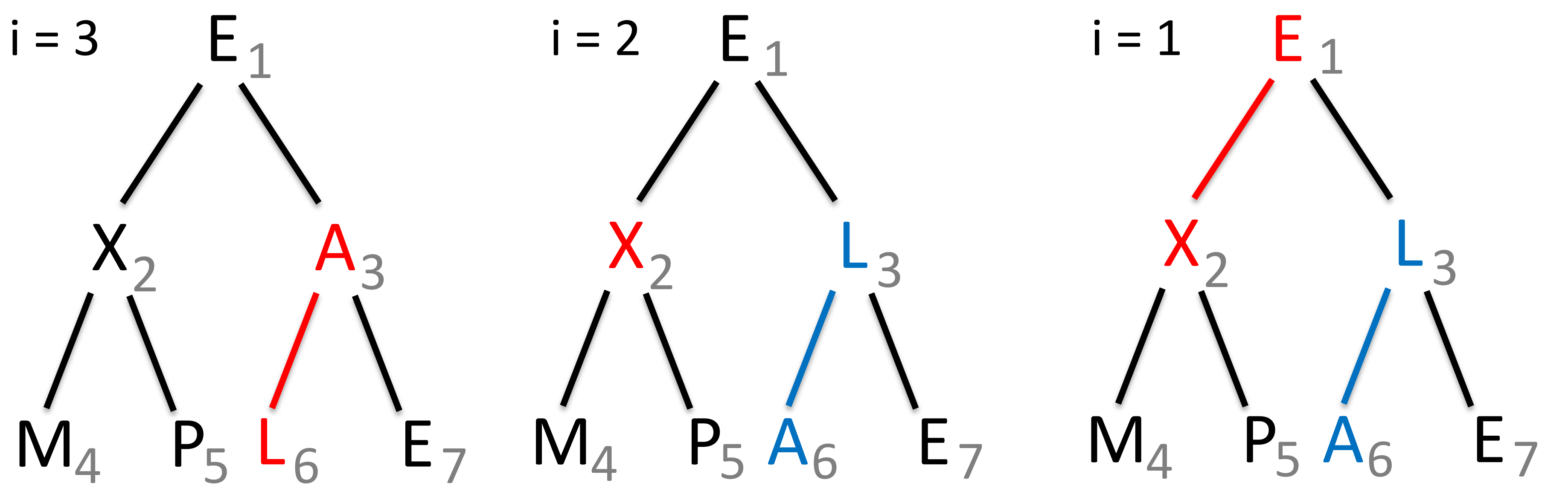
Swimming

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



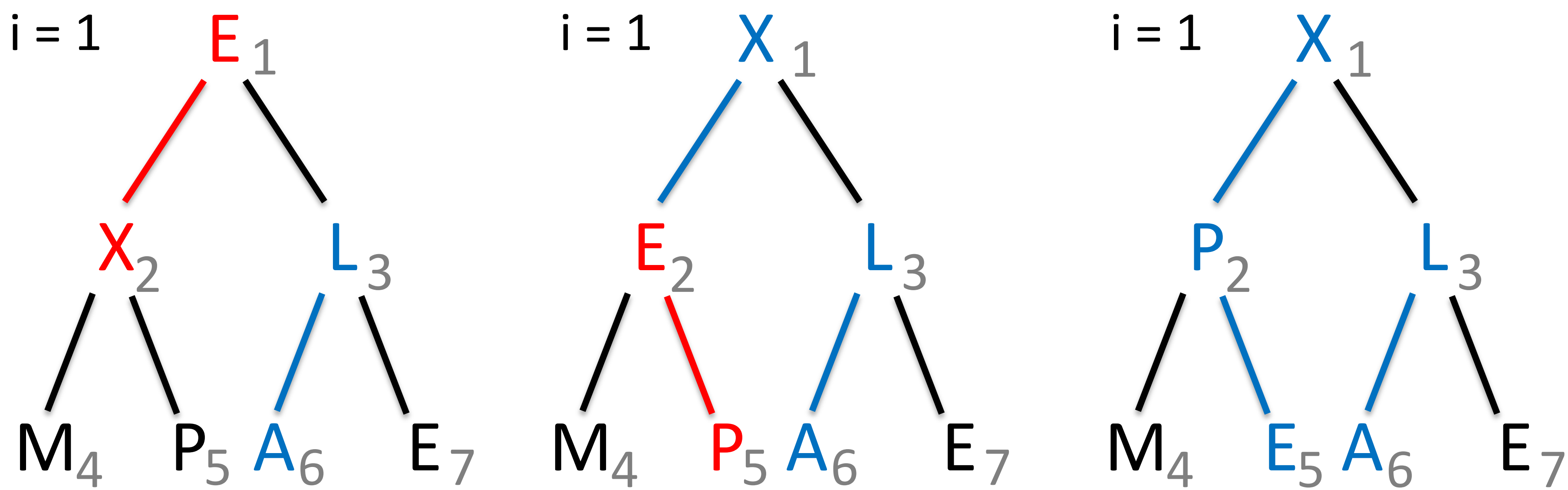
Build

```
private void build() {
    for (int i = n / 2; i > 0; i--)
        sink(i);
}
```



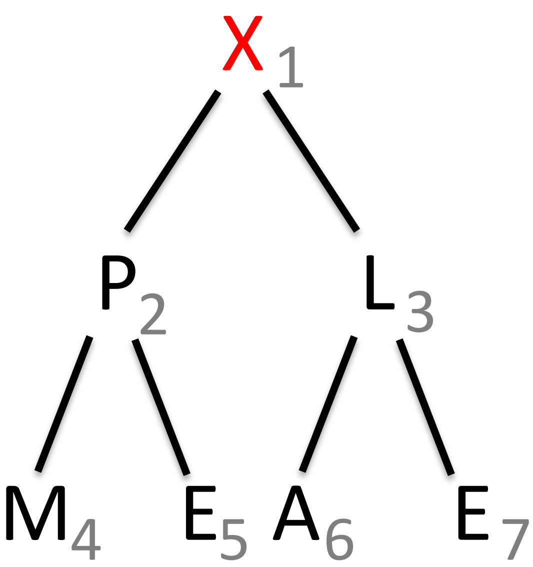
Build

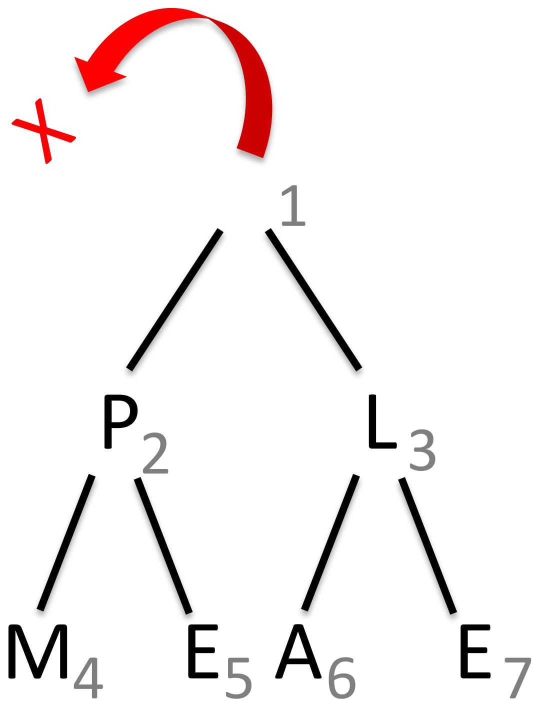
```
private void build() {
    for (int i = n / 2; i > 0; i--)
        sink(i);
}
```



```java
public T removeMax() {
    if (n == 0)
        throw new NoSuchElementException("Binary Heap underflow");
    T result = a[1];
    a[1] = a[n--];
    sink(1);
    return result;
}
```

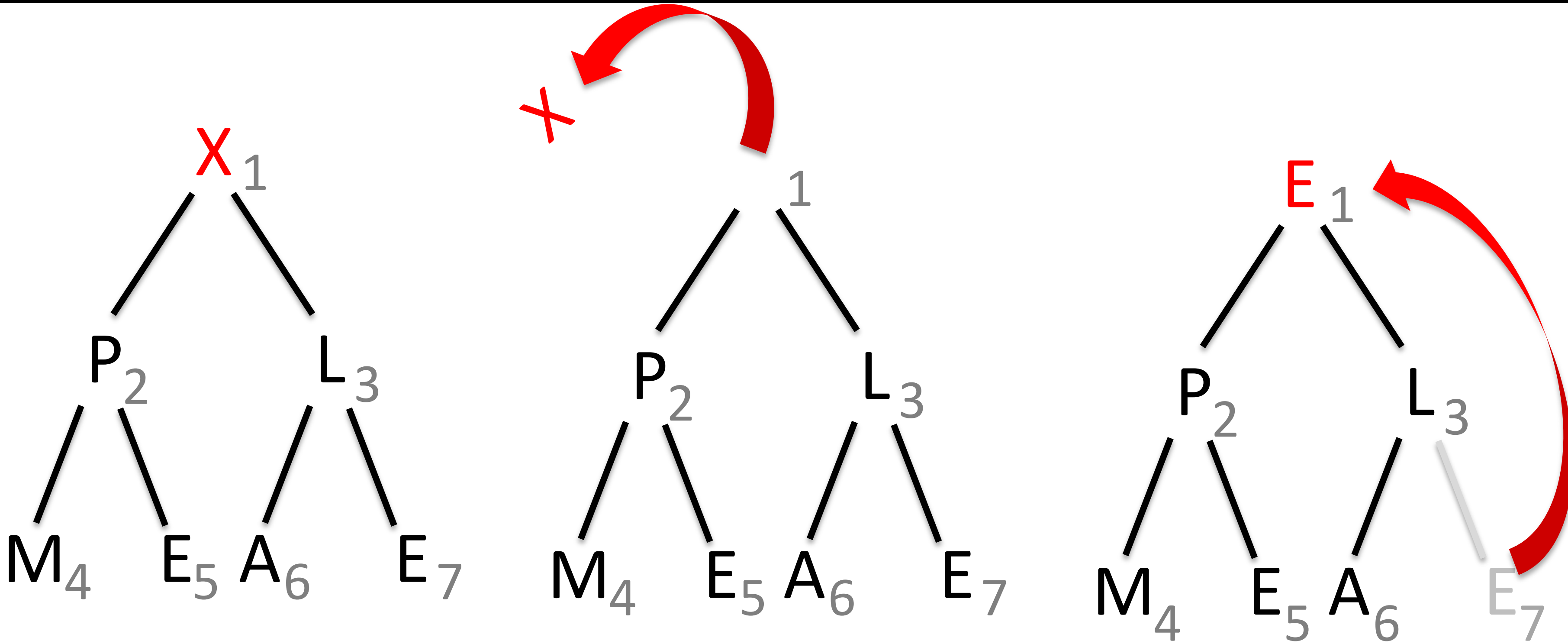
removeMax





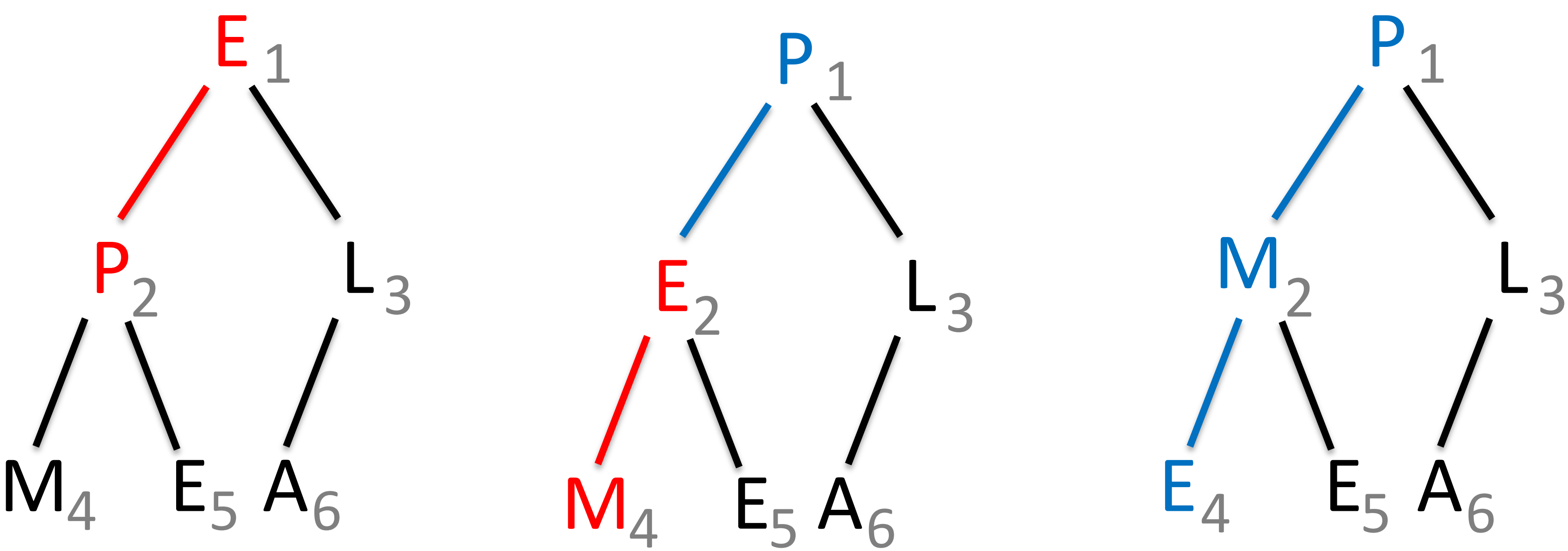
```java
public T removeMax() {
    if (n == 0)
        throw new NoSuchElementException("Binary Heap underflow");
    T result = a[1];
    a[1] = a[n--];
    sink(1);
    return result;
}
```

removeMax



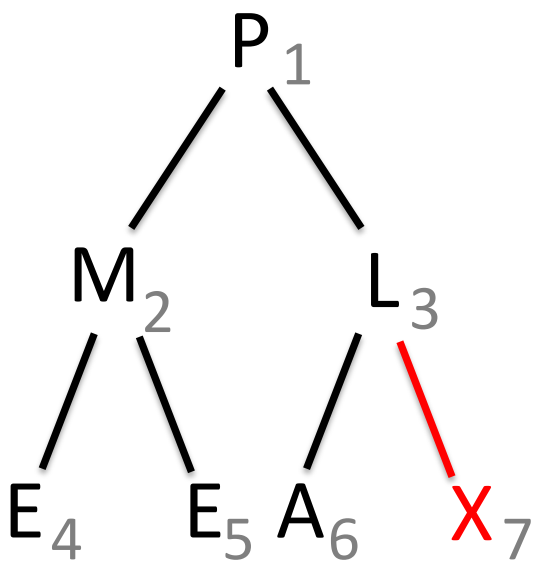
```java
public T removeMax() {
    if (n == 0)
        throw new NoSuchElementException("Binary Heap underflow");
    T result = a[1];
    a[1] = a[n--];
    sink(1);
    return result;
}
```

removeMax



```java
public void add(T key) {
    if (n == a.length - 1) resize(2 * a.length);
    a[++n] = key;
    swim(n);
}
```

add



add

```
public void add(T key) {
    if (n == a.length - 1) resize(2 * a.length);
    a[++n] = key;
    swim(n);
}
```