Pre-order

```
while (!end) {
  print(node.value)
  recursive(left)
  recursive(right)
}
```

In-order

```
while (!end) {
  recursive(left)
  print(node.value)
  recursive(right)
}
```

post-order

```
while (!end) {
  recursive(left)
  recursive(right)
  print(node.value)
}
```

Binary Heap (rooted at index 0)

parent(n) = (n-1)/2

left child = 2*n + 1

right child = 2*n + 2

k-nary Heap (rooted at index 0)

parent(n) = (n-1)/k

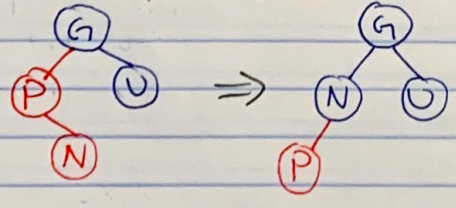
i-th child(n) = k*n + i

Red-black

- root is black
- if a node is red, both children are black
- new nodes are inserted as red

1 red root → flip color at each level

1 red child → swap to left



String concatenation is linear time

$x << n = x \cdot 2^n$

$x >> n = \lfloor x / 2^n \rfloor$ (rounded down)

Bitwise Operators

- AND (&)
- OR (|)
- XOR (^)
- Left Shift (<<)
- Right Shift (>>)
- Bitwise complement (~)
- Unsigned right shift (>>>)
  - shifts zero to left most

X	Y	x&Y	x|Y	x^Y	~(x)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

$x \,\&\, (x-1)$ → rightmost 1 in binary

1010 → 0010

$x \,\&\, (-x)$ → rightmost 1 in binary

1010 → 0010

$x \,|\, (1 << n)$ → number with the nth bit set

x = 10 n = 2

(1010) | (0100) = (1110)

Integer

byte	8-bits	signed
short	16-bits	signed
char	16-bits	unsigned
int	32-bits	signed
long	64-bits	signed

converting from lower bits to higher ok

Arithmetic Series

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Geometric

$$\sum_{i=1}^{n} a_i = a\left(\frac{1-r^n}{1-r}\right)$$

Finite

|r| < 1

Infinite $= \frac{a}{1-r}$

Hashing

External chaining

contains $\theta(Q)$

insert $\theta(Q)$

Q = length of longest list

Tree

Dot

- Preorder: root, left, right — left
- Inorder: left, root, right — bottom
- Postorder: left, right, root — right
- Level order: layer by layer

- Pre-order = postorder ⇒ 1 node or null
- Inorder = postorder ⇒ only left children
- preorder = Inorder ⇒ only right children

Heap

reheapify
- Sink down starting at bottom
- swim up from the top

Runtime

① Branching Factor ② Work @ each node

③ height of tree

Counting and radix can be linear time

Assuming size of alphabet R is constant

K = # of inversions

× not stable

✱ Comparison $\Omega(N \log N)$

Worst Space	Sorting		Best	Worst	Avg
O(1)	✱ Insertion	move each to the left	$\Omega(n)$	$O(n^2)$	$O(N+K)$
O(1)	✱ Shell's	↗ but sort distant to reduce inversions	$\Omega(n \log n)$	$O(n^2)$	
O(1)	✱ selection	pick min of each subarray	$\Omega(n^2)$	$O(n^2)$	
O(1)	× ✱ heap	build max heap then remove max and reheapify	$\Omega(n)$	$O(n \log n)$	
O(n)	✱ merge	divide and sort halves, merge results	$\Omega(n \log n)$	$O(n \log n)$	
O(n)	× ✱ quicksort	find pivot and move it to the right place	$\Omega(n \log n)$	$O(n^2)$	
O(n+k)	LSD Radix	sort each position from L to R	$\Omega(kn)$	$O(kn)$	k = avg length
O(n+k)	MSD Radix	" " R to L	$\Omega(n)$	$O(kn)$	
O(n+k)	Counting	data as keys, count occurrences	$\Omega(n+k)$	"Good" $\Theta(N+k)$	k = max key val

k is # answers to query

Func	Unordered List	Sorted Array	Bushy Search Tree	Hash Table	Heap
find	$\Theta(N)$	$\Theta(\log N)$	$\Theta(\log N)$	$\Theta(1)$	$\Theta(N)$
add (amortized)	$\Theta(1)$	$\Theta(N)$	$\Theta(\log N)$	$\Theta(1)$	$\Theta(\log N)$
range query	$\Theta(N)$	$\Theta(k + \log N)$	$\Theta(k + \log N)$	$\Theta(N)$	$\Theta(1)$
find largest	$\Theta(N)$	$\Theta(1)$	$\Theta(\log N)$	$\Theta(N)$	$\Theta(1)$
remove largest	$\Theta(N)$	$\Theta(1)$	$\Theta(\log N)$	$\Theta(N)$	$\Theta(\log N)$

Regex

✱ Include (?s) in beginning to match new line

- . any characters but not new line
- \d any digit [0-9]
- \D non-digit [\^0-9]
- \s space [\t\n\x0B\f\r]
- \S non-space [\^\s]
- \w word char [a-zA-Z_0-9]
- \W non-word [\^\w]

- [abc] matches a or b or c
- [^abc] everything except a, b, or c
- [a-c] range, a, b, or c
- [a-c[f-h]] union, abc, fgh
- [a-c&&[b-c]] intersection, only b or c
- [a-c&&[^b-c]] subtraction, only a

need to put a \ in front of \d and others "\d\d" two digits

Quantifiers

- * zero or more
- + one or more
- ? zero or one

(...) - group

Given P and Q are patterns

P|Q → a P or a Q

(?m) matches empty string

[...]{3} specified number of times

Boundary matchers

- ^ beginning empty
- $ end empty

Eg ^M

n$

Escape

\ Eg. what? matches what?

$\sum_{i=0}^{h}$ branching factor^i * work/node

Geo Sum $= a \cdot \frac{r^n - 1}{r - 1}$

$\log_b(m \cdot n) = \log_b M + \log_b N$

$\log_b\left(\frac{M}{N}\right) = \log_b M - \log_b N$

$\log_b(M^k) = k \cdot \log_b M$

$\log_b(1) = 0$

$\log_b b = 1$

$\log_b(b^k) = k$

$b^{\log_b(k)} = k$

$\log_b(x) = \frac{\log_a(x)}{\log_a(b)}$

$\sum_{i=0}^{\text{height}} \frac{\text{work}}{\text{node}} \cdot \underbrace{\frac{\#\text{ of nodes}}{\text{level}}}_{\text{branching factor}}$

$1 + 2 + 3 + 4 \cdots N^2$

$1 + 2 + 4 + 8 \cdots N$

$\sum_{m=1}^{\log N} 2^m$ bounded by $2N$

$\sum_{i=0}^{\log N} 10 \cdot 10^{**i}$

$10 \cdot 10^{\log n} = 10n = n$

Stirling's Approximation

$\log N! \in \Theta(N \log N)$

Graph

Topological sort == reverse DFS post-order

Start at indegree = 0

Stack → DFS

Queue → BFS

PQ → shortest path

consistent ⇒ admissible

Dijkstra $O(\underbrace{|E| \log |E|}_{\text{PQ updates}} + \underbrace{|V| \log |V|}_{\text{removing vertices}})$

MST $E <= V^2$

Kruskal's Algorithm

$O(\underbrace{|V|}_{\text{a set for every vertex}} + \underbrace{|E| \log |E|}_{\text{sort edges}})$

Prim's Algorithm

$O(\underbrace{|V|}_{\text{add each edge}} + \underbrace{|E| \log |E|}_{\text{sort edges}})$

Union Find

Amortized $O(\log^* N)$

A*

Admissible - heuristic never overestimate true dist.

Consistent - any 2 nodes A, B, $h(A) <= \text{dist}(A, B) + h(B)$

$O(|E|)$ worst case performance

$O(|V|)$ worst case space

$1 + 2 + 4 + \cdots + (N/2) + N = \sum_{i=0}^{\log(N)} 2^i \in \Theta(N)$

$1 + 2 + 4 + \cdots + 2^{N-1} + 2^N = \sum_{i=0}^{N} 2^i \in \Theta(2^N)$

More DS	add	remove	get	contains
Stack, Queue	$\Theta(1)$	$\Theta(1)$	$O(N)$	$O(N)$
SkipList	$O(N), \Theta(\log N)$	$O(N), \Theta(\log N)$	$O(N), \Theta(\log N)$	$O(N), \Theta(\log N)$
Arraylist	$O(N), \Omega(1)$	$O(N), \Omega(1)$		$O(N)$

Polymorphism	Static variable	static method	Instance Var	Instance Method
Compiler	Static	Static	Static	Static
Runtime	Static	Static	Static	Dynamic

- Instance variable lookup : start from static type. Error if not found.
- Instance method lookup : start from dynamic type. If not there, go to superclass until object class.
- Variables aren't polymorphic. Only instance methods are.

OOP

Constructors

- no return types
- can access instance / static variables
- can call instance/static methods

Instance variables

- Belong to each instance
- instancename.<instance var>
- cannot be referenced by Class.<instance var>
- cannot be referenced in static method
- can be referenced w/ or w/o this depending on name conflicts

Static Variables

- Belong to the whole class
- can be referenced by class
- can also be referenced by instance (but not recommended)
- can be referenced in constructor, instance and static methods
- can be referenced with or w/o this or Class depending on conflicts

Instance Methods

- have access to this ⇒ can access static & instance variables
- can call both instance and static method
- must have an object to invoke

Static Method

- no access to this
- only access to static variables
- can only call other instance methods
- no instance method or variable allowed inside a static method
- can be invoked with class or w/ instance

Extends

- Inherits instance and static variables + methods and nested classes

Overriding

- signature must match
- can't override static methods

Abstract Classes

- cannot be dynamic type
- subclass must implement all abstract instance unless it's also abstract
- may have abstract instance methods with no body
- can have concrete methods, variables, and constructors

Interfaces

- automatically abstract — can't be dynamic type
- generally has a set of abstract instance method w/ no body
- classes that implement an interface must implement all methods unless it's also abstract.
- can implement multiple interfaces
- can use as static type

Inner method calls

Def: a method of class C calling another method of class C

- C must also have that instance method or inherits it
- For runtime, only dynamic type matters still!
- For static method, call the one in the class you are in before calling the inner method

Fields

Fields = instance variables and static variables

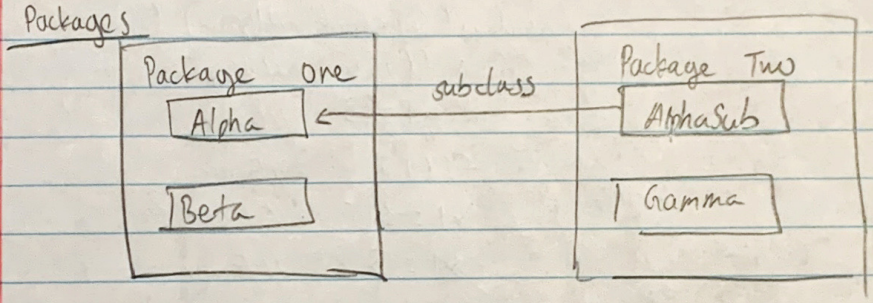
same behavior as static methods

Arrays

System.arraycopy(arr, k, arr, k+1, arr.length-k-1);

from to # to copy

Packages



Visibility (Packages)

Modifier	Alpha	Beta	AlphaSub	Gamma
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Access of Members of Class

Modifier	class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

7 14 15 17

Package Examples

```
package SomePack;
public class A1 {
   int f1() {
      A1 a = ...
      a.x1 = 3;
   }
   protected int y1;
   private int x1;
}
```

name conflicts

```
// Anonymous Package
class A2 {
   void g(SomePack.A1 x) {
      x.f1();    // Error
      x.y1 = 3;  // Error
   }
}

class B2 extends SomePack.A1 {
   void h(SomePack.A1, x) {
      x.f();     // Error
      x.y1 = 3   // Error
      f1();      // Error
      y1 = 3 ;   // OK
      x1 = 3 ;   // Error
   }
}
```

Inner non-static class

- need to access w/ an object of the outer class
- subclass outside of package that has a reference to an instance of superclass (let's say a protected method) cannot access the protected method using that superclass reference.
- no access to the protected method but "has" the method
- can access by Inheritance only. (the instance must be of the subclass)

Primitive

byte, short, int, long, float, double, boolean, char

Static methods cannot call instance methods!

Iterator → hasNext() , next() , and remove()

Iterator

```
import java.util.Iterator

public class FixedList implements List<Integer>, Iterable {
    protected int[] values;
    int count;

    private class FixedListIter implements Iterator<Integer> {
        int nextIndexToReturn;

        public Integer next() {
            ...
        }
        public boolean hasNext() {
            ...
        }
        ...
    }
}
```

To use

```
Iterator<Integer> iter = list.iterator();
```

- Class that implements Iterable must provide an iterator method that returns an Iterator.

- If a class implements Iterable, can use : operator to iterate.
- class that implements Iterable must provide an Iterator that allows iteration through that object.