Typecasting only tells compiler to ignore type-checking duties, doesn't actually change anything

- Dynamic types - types are evaluated @ runtime (v-table)
- Dynamic Method Dispatch - actual method called @ runtime is determined based on object's actual type
  - for overriden methods, actual method invoked decided @ runtime

```
for (int x : set) {
    System.out.println(x);
}
```

<=>

```
Iterator<Integer> seer = set.iterator();
while (seer.hasNext()) {
    int x = seer.next();
    System.out.println(x);
}
```

if access not defined, defaults to public

== compares memory locations (bit comparison), equals() can be implemented as in interface

'default' in interface means function is implemented same way unless overridden

Overloading only takes into account parameters (# & type)

subclass constructors call super constructor first (super();) by default

instance variables can access static (Main m = new Main(); m.staticVar; is valid)
(m.staticVar = newVal; changes Main.staticVar and for all other instantiations

can cast a class to parent interface

```
class Ch<E> {
    static E se() {
        return null;
    }
}
```

does not compile

arr.length not size()

Set API ← HashSet

- add(E element) - returns T if element is new in set, else F
- contains(E element)
- isEmpty()
- remove(E element) - returns T if element was in set, else F
- size()
- toArray()

Hash Map API ← Map

- put(K key, V value)
  - returns prev val associated w/ key or null
- get(K key)
  - returns val associated w/ key
- isEmpty()
- size()
- remove(K key)
  - returns val associated or null

instance functions cannot be called by static functions

Strings are immutable ← treat as primitive

assignment & parameter passing copies bits (new var for primitives)
(copies pointer for reference types)

Class<Obj> c = new Class<>();

$\underbrace{\texttt{Obj[] arr}}_{\text{declaration}} = \underbrace{\texttt{new Obj[size];}}_{\text{initialization}}$

Obj[]arr = new Obj[]{element, ..., element}

c instanceof Class cClass

checks if c is instance of Class, if yes, typecasts to Class & renames to cClass

WQU w/ PC

constructor $\Theta(N)$

isConnected $O(\log N)$

connect $O(\log N)$

no path compression

M ops on N nodes $O(M \overbrace{\lg^* N}^{\leq 5})$

for $M \to \infty$

$= O(M\alpha(N))$

| size | parent |

LLRBs

rotateLeft(x)
y = x.right.left
x.right.left = x
x.right = y

rotateRight(x)
y = x.left.right
x.left.right = x
x.left = y

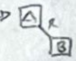
normal BST

- 1-1 map w/ 2-3 tree
- red link is "glue" link
  - always on left
- no more than ~2x height of 2-3 tree corresponding

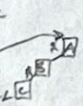
insert into BST then use rotations

Rules:

- when inserting, use red link
- if right red link, rotateLeft(node w/ right red link)



- if two consecutive left red links, rotateRight(node w/ two consecutive left links)



- if a node has 2 red children, flipColor(node)
  - flip color of every link touching node

all operations $O(\log N)$

Topological Ordering/Sort

- DFS traversal from every vertex w/ indegree 0, not clearing markings b/w traversals
- record postorder in list
  - TO is reverse of list
- DFS from arbitrary vertex
  - if not all marked, pick unmarked vertex & repeat
  - repeat until done

$O(V+E)$ time

array rep. of heap

floordiv

- parentIndex = nodeIndex/2
- root @ index 1
- left child @ 2index
- right child @ 2index+1

BST Hibbard Deletion

- no children
  - just remove
- one child
  - move child to nodes place
- two children
  - promote either rightmost (grand..) child of left tree or leftmost (grand..) child of right tree

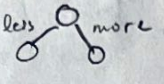
Hash sets

$O(N/M)$ for get

for $M = \Theta(N)$, $\Rightarrow O(1)$

put - $O(1)$

BST



$\text{floorMod}(a,b) = (a - \text{floorDiv}(a,b) * b)$

$1+2+4+\dots+\frac{n}{4}+\frac{n}{2}+n = 2n-1 = \Theta(n)$

Heap

Binary min-heap

- min-heap
  - every node is ≤ its children
- complete
  - only missing items @ bottom level, all items as far left as possible
- add $\Theta(\log N)$
  - add to bottom, leftmost "swim" up ← swap
- removeSmallest $\Theta(\log N)$
  - swap rightmost ← in bottom layer into root "sink" down ← swap
- getSmallest $\Theta(1)$

spanning tree of UG G

- connected
- acyclic
- includes all vertices

cut - assignment of graph's nodes to 2 non-empty sets

- crossing edge connects nodes from diff sets

cut property - for any cut, minimum weight crossing edge is in MST

General MST algo

- find a cut w/ no crossing edges
- add smallest crossing edge to MST
- repeat until V-1 edges

Prim's Algo

- repeatedly add shortest edge that has one node in MST under construction
- repeat until V-1 edges
- insert all edges into fringe PQ in order of dist from tree
- remove closest v from PQ, relax all edges pointing from v
- same O as Djikstra

Kruskal's Algo - $O(E \lg E)$

- consider edges in order of increasing weight
- add to MST unless cycle is created
- repeat until V-1 edges
- use PQ for edges

only for directed acyclic graph

DAG SPT Algo

- visit vertices in topological order, relaxing edges as you go

$O(V+E)$ time

valid hash

- deterministic
  - equals works
- consistent
  - repeatable

good hash

- valid
- uniform distr.
- quick
- int.

Comparator(i, j)

- 0 if i = j
- - if i < j
- + if i > j

B-Trees → non leaf w/ k items must have k+1 children

- when leaf is overstuffed, move second smallest to parent, smallest as new leaf
- always insert rightmost
- all operations $O(\log N)$
- order L = n ⇒ # of items allowed in node
- n_1-n_2-n_3 ⇒ # of children node may have
- n_1-n_2 tree ⇒ n_1 items, n_2 children
- all leaves same distance from root
- node w/ k items has k+1 children
- always balanced

Tree - exactly one path between nodes - no cycles

DFS → counterclockwise around graph from top

level-order sort

- preorder → visit when pass on left
  - visit a node, then traverse its children
- inorder → visit when cross bottom of node
  - traverse left child, visit, then traverse right child
  - leftmost → rightmost
- Postorder → visit when pass on right
  - traverse left, then right, then visit

A* algo

- visit nodes in order of $d(\text{source}, v) + h(v, \text{goal})$ ← estimate
- insert all vertices into PQ fringe, storing in order above
- remove best v from PQ, relax edges from v

heuristic

- admissible
  - $h(v,w) \leq$ true $d(v,w)$
- consistent
  - for each neighbor of w,
    - $h(v,a) \leq \text{dist}(v,w) + h(w,a)$ ← weight of edge

Relaxing edge p → q w/ weight w

- if distTo[p] + w < distTo[q]
  distTo[q] = distTo[p] + w
  edgeTo[q] = p
  PQ.changePriority(q, distTo[q])

Shortest Paths	Djikstra's	$O(E\log V)$ ← if E > V
MST	Prim's	$O(E\log V)$
MST	Kruskal's	$O(E\log E)$
MST	Kruskal's w/ pre-sorted edges	$O(E\log^* V)$

uses WQUPC

load factor = N/M

Arr Rep of Heap:

Root @ 1 index
Left Child @ $2i$
Right Child @ $2i+1$
Parent @ (int) $i/2$

always assume $N \to \infty$

- O - upper bound ~~(worst case)~~
- Ω - lower bound ~~(best case)~~
- Θ - tighest bound (use when O = Ω)

$1+2+3+\dots+(N-2)+(N-1) = \frac{N(N-1)}{2} \to N^2$

$1^k+2^k+3^k+\dots+(N-1)^k+N^k = \Theta(N^{k+1})$, $k \geq 0$

$1+2+4+8+\dots+2^Q = 2(2^Q)-1 = \Theta(2^Q)$

$k^0+k^1+k^2+\dots+k^Q = \Theta(k^Q)$, $k > 1$

	OrderedArr	BushyBST	HashTable	Heap
add	$\Theta(N)$	$\Theta(\log N)$	$\Theta(1)$	$\Theta(\log N)$
getSmallest	$\Theta(1)$	$\Theta(\log N)$	$\Theta(N)$	$\Theta(1)$
removeSmallest	$\Theta(N)$	$\Theta(\log N)$	$\Theta(N)$	$\Theta(\log N)$

PQ

Graphs

simple

- no self-connecting edges
- no parallel edges

traversal s-t connectivity

- mark s
  - DFS - $O(V^2)$ - matrix
    - $O(V+E)$ ← adj list
- BFS - $O(V^2)$ - matrix
  - fringe queue
    - until fringe is empty
      - remove v from fringe
      - for each unmarked neighbor n of v, mark n, add n to fringe, set edgeTo[n] = v, distTo[n] = distTo[v] + 1
  - $O(V+E)$ ← adj list
- Djikstra's Algo
  - add all vertices into fringe PQ, storing in order of distance from source
    - remove closest vertex v from PQ, relax all edges pointing from v
  - visits vertices in order of total dist. from source
  - for each node, visit unvisited neighbors
    - pick next node based on smallest weight
  - $O(V\log V + V\log V + E\log V)$
    - add $O(V\log V)$
    - removeSmallest $O(V\log V)$
    - changePriority $O(E\log V)$
  - $O(E\log V)$

Sort Table

	AddnltMem	Best Run	Worst Run	Stable
election	$\Theta(1)$	$\Theta(N^2)$	$\Theta(N^2)$	N
sertion	$\Theta(1)$	$\Theta(N)$	$\Theta(N^2)$	Y
ergesort	$\Theta(N)$	$\Theta(N \log N)$	$\Theta(N \log N)$	Y
eapsort	$\Theta(1)$	$\Theta(N \log N)$	$\Theta(N \log N)$	N
uicksort	$\Theta(1)$-Hoare $\Theta(\log N)$ 3way	$\Theta(N \log N)$	$\Theta(N^2)$	N-Hoare Y-3 way

Stable ← preserves order of identical elements

↑ if $\Theta(1)$, ⇒ in-place

Heapify:

```
i = N/2
while i > 0:
  bubble down item @ idx i
  i = i-1
```

Radix Sorts

radix - base of #ing system

no comparisons used

sorts. N items in $\Theta(N+R)$ ← size of alphabet

LSD - least sig. digit

sort start @ rightmost digit

$\Theta(WN + WR)$ ← width of item (# digits)

stable

MSD - most sig. digit

sort start @ leftmost dig.

$O(WN + WR)$

↳ $\Theta(N+R)$ best case, only one scan ↳ all MSD unique

Selection Sort - move smallest item in unsorted part of arr, move to end of sorted pt of arr

sorted | unsorted → front to back → front are final
arr

Insertion Sort - for item i in arr, swap backwards until in correct place → front to back → last part same for as long as ...

↑ very fast for almost sorted array / small arr

Merge Sort - split into 2 ~even pieces, sort each recursively, then merge in order → id by left not interact w/ right until end

Heap Sort - heapify array, remove max & replace w/ last item in heap, then add max item @ back of unsorted arr ↑ then bubble down

↳ id by sorted back to front final

Quick Sort - select pivot, everything < pivot on left, everything > pivot or right

3-way Partitioning - ~~choose first element as pivot~~

↳ 3 arr (less, equal, greater)

Hoare Partitioning - let first item be pivot, two ptrs L & G, L @ left (right of pivot), immediate G @ right most

L likes small elem, G likes big elem

Move ptrs tog. stopping on "disliked" elements (L dislikes larger/equal, G dislikes smaller/equal)

if both stopped, swap & move ptrs

done when ptrs cross, then swap G & pivot

DMS

- compiler starts @ static method, then goes down inheritance tree until it hits dynamic method / overriding methods
- static methods use static type