# CS 61B Midterm 1 Notes

Jeffrey Shen
3034555577

## Pointers

- Passes a pointer to array as parameter for a method
- pass by value : primitive types

## Classes / Inheritance

- Always implicit call to Superclass constructor from subclass
- Dynamic method selection

| Compile Time | Runtime |
|---|---|
| Static | Dynamic |
| 1) Choose method<br>throw error if not found | 2) Find exact signature match in subclasses |

| Variable | Static | Dynamic |
|---|---|---|
| d | Dog | Corgi |
| c | Corgi | Corgi |

- Remember to use Package Name for public class if using from another package ( ex) P1. C1 x = new, ... )

Access Modifiers

| Modifier | Class | Package | Subclass | Global | |
|---|---|---|---|---|---|
| public | ✓ | ✓ | ✓ | ✓ | - Avaliable anywhere |
| protected | ✓ | ✓ | ✓ | X | - Avaliable within subclasses outside package |
| package (default) private | ✓ | ✓ | X | X | - Avaliable within same package |
| private | ✓ | X | X | X | - Avaliable only within same class |

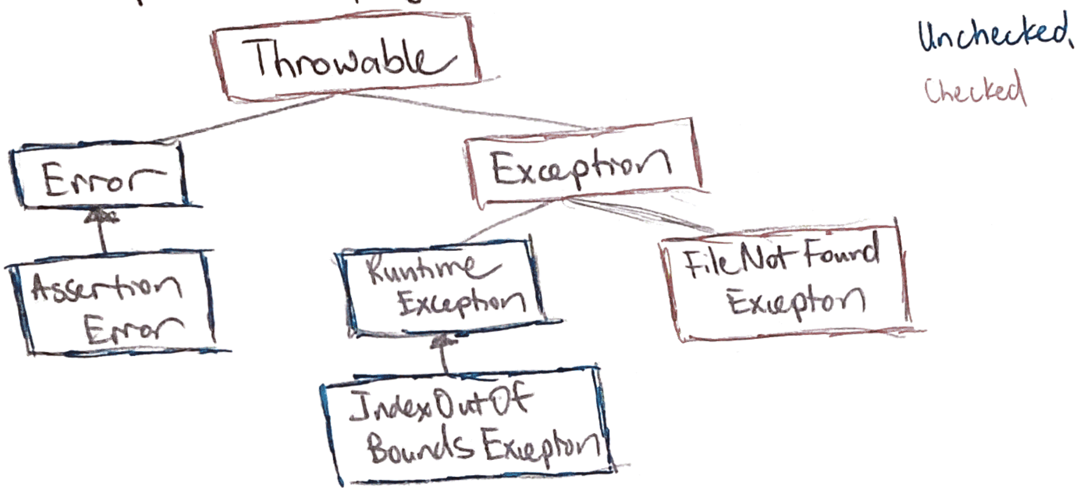
- static methods cannot be overridden
- Overriding : same method signature in subclass
- Overloading : Same method name but different numbers or types of arguments
- Static belong to class , non-static belongs to instance

## Exceptions

Ex) "throw new IllegalArgumentException();"

try { //code } catch ( SomeException e) { //code };

- Checked Exceptions: non-programmer errors declared in method header
- Unchecked Exceptions: programmer errors




## Unit Testing

Ex) assertArrayEquals (int[] expected, int[] actual)

assertEquals ( double expected, double actual, double delta)

## Interfaces : implements

```
public interface Comparable<T> {
    int compareTo( T x);
} // whether this <,=,> x
```

```
public interface Comparator<T> {
    int compare(T o1, T o2);
    boolean equals ( Object obj);
}
```

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

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
}
```

## Tips

- Check variable types and return, indexes
- go back and check numbers after filling outline
- Int List for loops

Ex) for (p.tail = null, p = result; p.tail != null; p = p.tail) {

CS 61 B Midterm 2 Notes Jeffrey Shen 3034555577

## Integers

| Type | Bits | Signed |
|---|---|---|
| byte | 8 | Yes |
| short | 16 | Yes |
| char | 16 | No |
| int | 32 | Yes |
| long | 64 | Yes |

### Bit Twiddling

| | | | Input | Use |
|---|---|---|---|---|
| Mask | & | "and" | 2 | Mod (last digits (n-1)) |
| Set | \| | "or" | 2 | Add |
| Flip | ^ | "not equal" | 2 | Unequal bits |
| Flip all | ~ | "not" | 1 | |
| Shift Left | << | shift left, falls off | | Multiply |
| Arithmetic Right | >> | Brings sign bit | | Divide |
| Logical Right | >>> | Starts with 0 | | Divide |

## Complexity

1) Consider Worst Case
2) Pick proxy for overall runtime
3) Ignore lower order terms
4) Ignore multiplicative constants

| | | |
|---|---|---|
| Big Theta | $\Theta(N)$ | Same order of growth |
| Big O | $O(N)$ | Upper bounded by N |
| Big Omega | $\Omega(N)$ | Lower bounded by N |

### Collections Interface

List : Indexed sequences w/ duplication (ex. ArrayList, Linked List)
Set, Sorted Set : Collections w/o duplication (ex. HashSet, TreeSet)
Map, Sorted Map : Dictionaries, key value pairs (ex. HashMap, TreeMap)

### Time Complexities

Avg Time [Worst Time]

| Data Structure | Access | Search | Insertion | Deletion |
|---|---|---|---|---|
| Array | $\Theta(1)$ | $\Theta(n)$ | $\Theta(n)$ | $\Theta(n)$ |
| Stack / Queue / Linked List | $\Theta(n)$ | $\Theta(n)$ | $\Theta(1)$ | $\Theta(1)$ |
| Hash Table | N/A | $\Theta(1)$ [$O(n)$] | $\Theta(1)$ [$O(n)$] | $\Theta(1)$ [$O(n)$] |
| BST/Heaps | $\Theta(\log n)$ [$O(n)$] | $\Theta(\log n)$ [$O(n)$] | $\Theta(\log n)$ [$O(n)$] | $\Theta(\log n)$ [$O(n)$] |

Inserting n elements with logn time each becomes nlogn

$\Theta(n)$: $n$, $\frac{n}{2}$, $\frac{n}{4}$, … ] logn

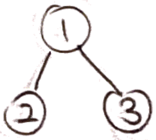
$\Theta(n\log n)$: $n$; $\frac{n}{2}$ $\frac{n}{2}$; $\frac{n}{4}$ $\frac{n}{4}$ $\frac{n}{4}$ $\frac{n}{4}$ — n, n, n ] logn

$\Theta(n\log n)$: 1, 2, 4, 8, … ] n

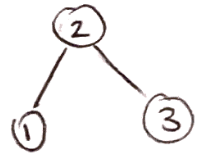
# Trees

Inorder on a BST will give sorted lst

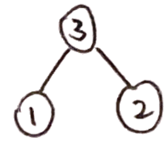
Pre order




In order



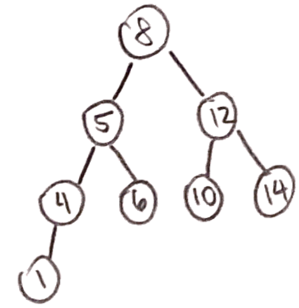

Post Order




Depth First: Stack (LIFO)

Breadth-First: Queue (FIFO)
(Level-Order)

## Binary Search Tree (BST)




- Nodes in left subtree have smaller keys
- Nodes in right subtree have larger keys

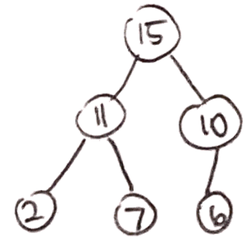
Insertion: Search for node, input where should be

Deletion: Find smallest number in right side and replace

## Game Trees


- Assign heuristic to board, node is position, edge is move
- I choose max value, opponent chooses min val
- Alpha-beta pruning: prune as we search, sends down values that are acceptable and discontinues if not in range

## Heaps




- Min/Max Heap: Labels of both children are less/greater than node

Insert: Put at next avaliable on botton row, bubble up

Remove: Swap with element in bottom right, bubble down

Change: Find node, bubble up or down

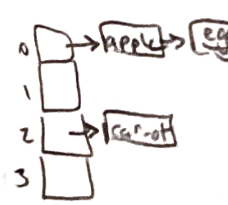
## Hashing

Convert key to bucket number using hash funct

Avoid collisions: keys that hash to equal values, Valid if hash func is same for same keys

Load Factor: N(items)/M(buckets) = L

Resize when Load Factor > some value, resize table, rehash all items




Add, lookup, deletion: Θ(1) but can't find largest/smallest

## Pattern Matching, Regular Expressions (RegEx)

Character class ( [0-9 abd-qs-z] )

- Any of the single characters

Wildcard ( . )

- Period can match any character

Compliment, Not ( [^abe] )

- Matches any single character other than those listed

Character Class shortcut ( \s , \d )

\s - whitespace  \d - [0-9]  need to use "\\d" to get \key

Repetitions ( * , + , ? )

p* - "0 or more repitition of p"

P+ - "1 or more Ps"

P? - "0 or 1 Ps"

Or (P|Q)

- Either "P" or "Q"

Group ((P))

- Sub pattern to refrene later

Escape ( \? , * , \. , \+ )

- Need to use two-character escape sequences to match (\\?)

## Exam Tips

- Pay attention to object types
- Remember case where input is null
- Don't forget capital and lowercase RegEx, edge cases
- Linked List in HashMap takes linear time to add, (check for repeat)

# CS 61 B Final Notes

Jeffrey Shen 3034555577

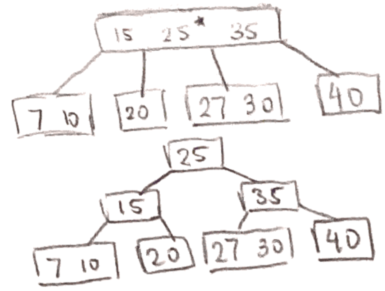
## Sorts

LSD/MSD → Selection → heap → quick → merge → insertion

| Sort | Description | Runtime Avg [Worst] | Stable | Diagram |
|---|---|---|---|---|
| Insertion Sort | - Add each item from unsorted sequence, insert into ordered subsequence<br>> for i=1 to n-1<br>while arr[i] < arr[i-1]<br>swap(i, i-1)<br>- Good for small data sets or almost ordered | $\Theta(kN)$ $[O(N^2)]$<br>where k is # of inversions | Y | 6 1 3 0 5<br>1 6 3 0 5 |
| Selection Sort | - Repeatedly finding min element and placing in front<br>> for i=0 to n-1<br>for i to n-1<br>find min element<br>swap (min, i) | $\Theta(N^2)$ | N | 6 1 3 0 5<br>0 1 3 6 5 |
| Heap sort | - Sort into max heap and keep selecting largest<br>> for i=n/2-1 to 0<br>heapify<br>for i=n-1 to 0<br>swap (0, i)<br>heapify 0 to i | $\Theta(N \log N)$<br>Best: $\Theta(N)$ | N | heap<br>6 5 3 1 0<br>heap<br>5 3 1 0 6 |
| Merge Sort | - Divide data into equal parts, recursively sort halves, merge results<br>> sort (int[] arr, int l, int r){<br>if l<r<br>sort (arr, 0, middle)<br>sort (arr, middle, r)<br>merge (arr, l, m, r) | $\Theta(N \log N)$ | Y | 6 1 3 0 5<br>1 6 0 3 5 |
| Quicksort | - Partition data into pieces everything > pivot at high<br>everything <= pivot on low end<br>- Can do insertion sort when partition is small enough<br>> quicksort (arr[], low, high)<br>if low < high<br>partition index = partition (arr, low, high)<br>quicksort (arr, low, pi-1)<br>quicksort (arr, pi+1, high) | $\Theta(N \log N)$<br>$[O(N^2)]$<br>if choosing bad partitions | N | Choose last as partition<br>1 3 0 5 6<br>0 1 3 5 6 |
| Distribution Counting | - put integers into N buckets of counts then have running sum of indexes, then insert into final array<br>> for i=0 to n<br>count [arr[i]] ++    set up count array<br>for i=0 to K<br>count [i] += count [i-1]    set up running sum<br>for i=n-1 to 0<br>output [count[arr[i]]-1] = arr[i]    copy to output<br>count [arr[i]]-- | $\Theta(N+K)$<br>where K is range of input | Y | count<br>1 1 0 1 0 1 1<br>0 1 2 3 4 5 6<br>running sum<br>0 1 0 2 0 3 4<br>0 1 2 3 4 5 6<br>output<br>0 1 3 5 6 |
| Radix Sort (LSD, MSD)<br>Least, Most Significant Digit | - Sort keys one at a time<br>- Good for small keys<br>> for each digit :<br>for i=0 to n<br>count[arr[i] % 10] ++;    count digit<br>for i=0 to 10<br>count[i] += count[i-1]    running sum<br>for i=n-1 to 0 | $\Theta(B)$<br>when B is # bytes, size of key data | Y | count<br>1 1 0 1 0 1 1<br>0 1 2 3 4 5 6<br>running sum<br>0 1 0 2 0 3 4<br>0 1 2 3 4 5 6 |

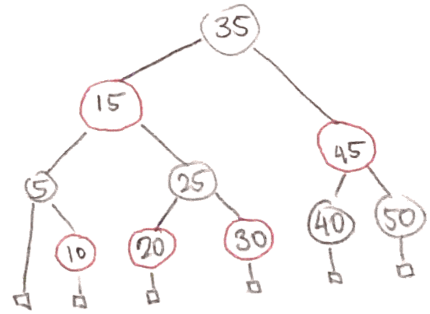
# Balanced Search Structures
- Want to maintain bushy trees to perform fast operations

## B-Trees
Ex) 2-3 trees, 2-4 or 2,3,4 trees




Order M B-tree has max of M children for node and max of M-1 elements per node, follows BST structure w/ left nodes smaller than element and right nodes larger. If overfill, move one node up and split.

## Red-Black Tree (Left-Leaning Red Black Trees)



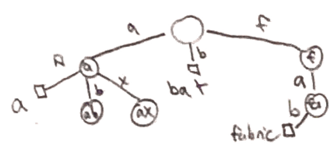

- Representation of B tree that is easier to implement
- Properties:
  1) root is black
  2) Every leaf node has no data and is black
  3) Every leaf has same number of black ancestors
  4) Every internal node has two children
  5) Every red node has two black children

> insert as red node in correct place

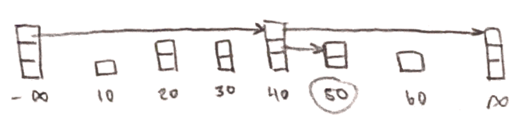
- Fixup tree:
  1) Convert right-leaning to left-leaning
     > right().isRed() and left().isBlack()? rotate left
  2) Rotate linked red nodes into normal 4 node
     > left().isRed() and left().left().isRed()? rotate right
  3) Break up 4 nodes into 3 nodes or 2nodes
     > left().isRed() and right().isRed()? colorFlip(tree)
  4) Turn root black after fixups




## Trie
- Each internal node coorsponds to letter, possible prefix
- Can use DataIndexedCharMap, Bushy BST, or Hash Table
- use for prefix can combine with priority Queue for autocomplete




## Skip Lists
- Search tree where we choose to put keys at "random" heights
- Start at top layer, search until next step would overshoot, go down one layer and repeat




## Performance

| B-Trees / Red-Black Trees | Tries | Skip Lists |
|---|---|---|
| $\Theta(\log N)$ | $\Theta(B)$ | $\Theta(\log N)$ |
| searches, insertions, deletions | searches, insertions, deletions | searches, insertions, deletions |
| | B is length of key | randomized |

# Hash functions

function f

Cryptographic Hash Functions - are so unlikely to have collision we can ignore

- Pre-image resistance: given $h = f(m)$ computationally infeasible to find $m$
- Second pre-image resistance: given message $m_1$ infeasible to find $m_2 \neq m_1$ st $f(m_1) = f(m_2)$
- Collision resistance: difficult to find any two messages $m_1 \neq m_2$ st $f(m_1) = f(m_2)$

Ex) SHA1 160 bit hash codes of contents in hex

# Graphs

Graphs have set of nodes (V) and edges (E), can be directed, cyclic or acyclic

## Recursive Depth-First Traversal

Stack

- mark nodes as we traverse, don't traverse previously traversed

Preorder - mark, visit, traverse edges

```
> void preorderTraverse(Graph G, Node v){
    if v is unmarked
        mark(v)
        visit(v)
        for Edge(v,w) ∈ G
            traverse(G,w)
```

Postorder - mark, traverse edges, visit

```
> void postorderTraverse(Graph G, Node v){
    if v is unmarked
        mark(v)
        for Edge(v,w) ∈ G
            traverse(G,w)
        visit(v)
```

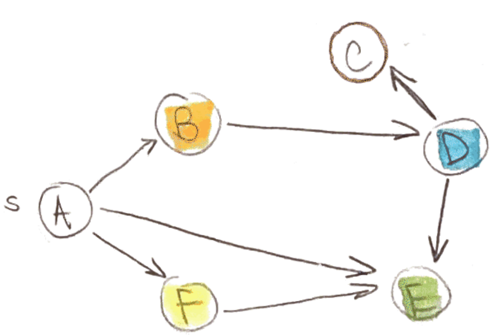
## Breadth-First Traversal

- visit edges and store nodes in a queue for processing

## Topological Sorting

For a Directed Acyclic Graph (DAG), find linear order of nodes where order $v_0, v_1, \ldots$ st $v_k$ is never reachable from $v_{k'}$ if $k' > k$

Ex)




DFS preorder

A B D C E F

DFS postorder

C E D B F A

BFS

A B E F D C

Topological Sort

A
B
F
D
C
E

Adjacency list

A → [D, E, F]
B → [D]
C → [ ]
D → [C, E]
E → [ ]
F → [illegible]

Adjacency matrix

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 0 | 0 | 1 | 1 |
| B | 0 | 0 | 0 | 1 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 0 |
| D | 0 | 0 | 1 | 0 | 1 | 0 |
| E | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 0 | 0 | 0 | 0 | 1 | 0 |

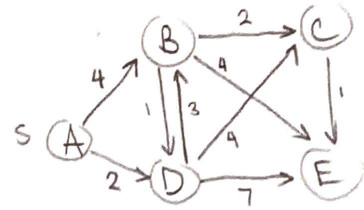
# Dijkstra's Algorithm

Given weighted graph w/ non-negative weights, connected

- Find shortest paths from source vertex s to some target vertex t in weighted graph

fringe: priority queue

> fringe.add(source, 0)
  for other vertices, fringe.add(v, ∞)
  while fringe not empty
    Vertex v = fringe.removeSmallest()
    for each edge(v, w)
      if distTo[v] + weight(v,w) < distTo[w]
        distTo[w] = distTo[v] + weight(v,w)
        edgeTo[v] = v
        fringe.changePriority(w, distTo[w])




| Node | dist | edgeTo |
|---|---|---|
| A | 0 | — |
| B | 4 | A |
| C | 6 | B |
| D | 2 | A |
| E | 7 | C |

fringe: {A: 0, B: 4, C: 6, D: 2, E: 7}

- Visit vertices in order of best known distance, relax edges

# A* search

- Want shortest path from source vertex to desired vertex
- Use heuristic guess h(v) and order by sum of distance + heuristic of remaining dist

Properties of heuristic:
1) Admissible: h(v, NYC) ≤ true distance from v to NYC
2) Consistent: for each neighbor of w:
   h(v, NYC) ≤ weight(v, w) + h(w, NYC)

Consistent ⇒ admissible

## Dijkstra's vs. A*

Both: time = time to remove V nodes from priority queue, + time to update neighbors, reorder queue
$\Theta((V+E)\log V)$

A* searches to particular target node, Dijkstra's finds shortest-path tree

# Minimum Spanning Tree

- Given set of places and distances between, find set of connecting roads w/ min total length

## Prim's Algorithm

- Grow tree from arbitrary node, add shortest edge connecting some node that isn't in tree
- Similar to Dijkstra's, compare weights instead of total distance
  > if w ∈ fringe && weight(v,w) < w.dist()
    w.dist() = weight(v,w); v.parent = v;

## Kruskal's Algorithm

- Consider edges in order of increasing weight, add unless cycle
- Use Union-Find:
  - Find what group, path to root
  - Combine two groups, point one root to other

> for each edge in increasing order of weight, if (v,w) connects different subtrees, combine