CS 61b: Final Review

Data Structures

Steve Sinha and Winston Liaw

DISCLAIMER

We have NOT seen the exam.
We do NOT know the format of the exam

What we are presenting is what we *"think is important" for the exam*

Review Topics

- Inheritance, Method Calls
- Asymptotic Analysis
- Data Structures
  - Binary Search Trees
  - B-Trees
  - Heaps
  - Hash Tables
- Graphs
  - DFS, BFS
  - Topological Sort
  - Dijkstra
  - Kruskal
- Sorting
- Skip Lists
- Threading, Synchronization

Inheritance/Method Calls

- Given the class definitions on the next slide, which lines in class foobarbaz are illegal?

Inheritance

```
package foo;
public class foo {
   static void f1() {…}
   protected boolean f2(int x) {…}
   private String f3(String s) {…}
}
```

```
package foo;
public class baz extends foo {
   private String f3(String s) {…}
}
```

```
package bar;
import foo.foo;
public class bar extends foo {
   protected boolean f3(int x) {…}
}
```

```
package foo;
import bar.bar;
public class foobarbaz {
   static void main(String[] args) {
      foo f = new foo();
      bar r = new bar();
      baz z;
      r.f3(3);
      f.f2(3);
      z = (baz) f;
      f = new baz();
      f.f2(3);
      z = (baz) f;
      z.f1();
      r.f1();
      ((foo) r).f1();
   }
}
```


Inheritance/Method Calls

- Access table:

	world	package	child	definer
public	X	X	X	X
private				X
protected		X	X	X
<default>		X		X

- Static methods called according to static type
- Child type can be assigned to parent variable without a cast, but the reverse requires one, and the dynamic types must match

Inheritance

```
package foo;
public class foo {
   static void f1() {…}
   protected boolean f2(int x) {…}
   private String f3(String s) {…}
}
```

```
package foo;
public class baz extends foo {
   private String f3(String s) {…}
}
```

```
package bar;
import foo.foo;
public class bar extends foo {
   protected boolean f3(int x) {…}
}
```

```
package foo;
import bar.bar;
public class foobarbaz {
   static void main(String[] args) {
      foo f = new foo();
      bar r = new bar();
      baz z;
      r.f3(3);          // ILLEGAL
      f.f2(3);
      z = (baz) f;      // ILLEGAL
      f = new baz();
      f.f2(3);
      z = (baz) f;
      z.f1();
      r.f1();           // ILLEGAL
      ((foo) r).f1();
   }
}
```


Asymptotic Analysis

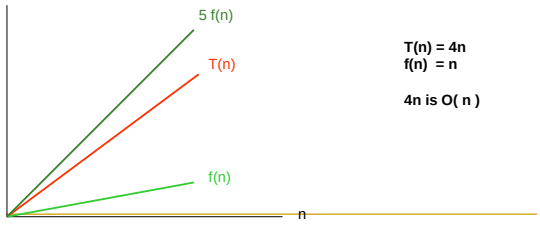
- O – Upper bound/Worst case
- Ω – Lower bound
- θ – both
- o – strictly Upper bound

More detail…

Asymptotic Analysis

T(n) is O(f(n)) if and only if there exists positive constants C and N such that

T(n) <= C f(n) for all n >= N



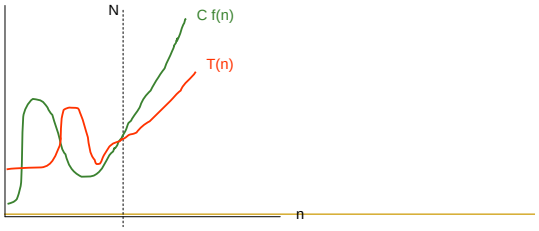
T(n) = 4n
f(n) = n

4n is O(n)

Asymptotic Analysis

T(n) is O(f(n)) if and only if there exists positive constants C and N such that

T(n) <= C f(n) for all n >= N



Asymptotic Analysis

T(n) is O(f(n)) if and only if there exists positive constants C and N such that

T(n) <= C f(n) for all n >= N

T(n) is Ω(f(n)) if and only if there exists positive constants C and N such that

T(n) >= C f(n) for all n >= N

Asymptotic Analysis

T(n) is θ(f(n)) if and only if

- T(n) is O(f(n))

  and
- T(n) is Ω(f(n))

Examples

$5n^2+1$ is $\theta(n^2)$

3n is $O(n^2)$, but 3n is NOT $\theta(n^2)$ because 3n is not $\Omega(n^2)$

Asymptotic Analysis Problem

- Find the running time of the following code:

```
int foo(int x) {
   int ans = 1;
   for (int i = 0; i < x; i++) {
      for (int j = 0; j < i; j++) {
         ans *= (i + j);
      }
   }
   return ans;
}
```

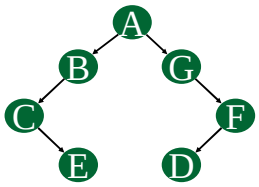

Asymptotic Analysis Solution

- The nested loops give away the answer: the outer loop executes x times, the inner loop an average of x/2 times, for a running time of $O(x^2)$.

```
int foo(int x) {
   int ans = 1;
   for (int i = 0; i < x; i++) {
      for (int j = 0; j < i; j++) {
         ans *= (i + j);
      }
   }
   return ans;
}
```


Trees: Binary Tree

Tree:

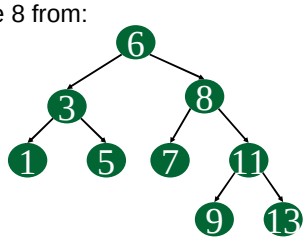


What are the Pre-, In-, and Post-order traversals of this tree?

Preorder : ABCEGFD
Inorder : CEBAGDF
Postorder: ECBDFGA

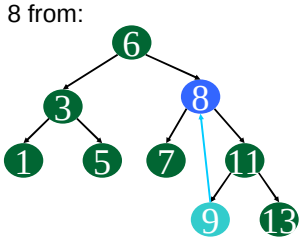
Trees: BST Problem

- Remove 8 from:



Trees: BST Problem

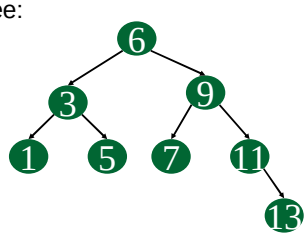
- Remove 8 from:



Replace with successor (left-most node in right subtree)

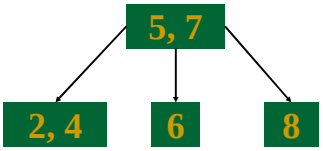
Trees: BST Solution

- Final tree:



Trees: B-Tree of Order 4 / 2-3-4 Tree

- Suggest a sequence of operations that would create the 2-3-4 tree.
  You can use removal as well as insertion.



Trees: 2-3-4 Tree Solution

- Insert: 4, 5, 6, 7, 8, 9, 2 Remove: 9

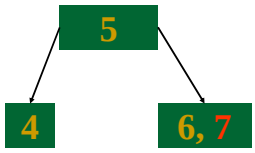
Trees: 2-3-4 Tree Solution

- Insert: 4, 5, 6, 7, 8, 9, 2 Remove: 9

Touch node size = 3

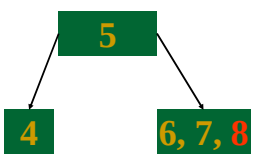
Trees: 2-3-4 Tree Solution

- Insert: 4, 5, 6, 7, 8, 9, 2 Remove: 9



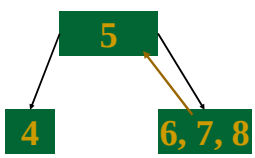
Trees: 2-3-4 Tree Solution

- Insert: 4, 5, 6, 7, 8, 9, 2 Remove: 9



Trees: 2-3-4 Tree Solution

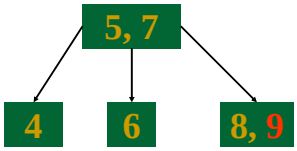
- Insert: 4, 5, 6, 7, 8, 9, 2 Remove: 9



Touch node size = 3

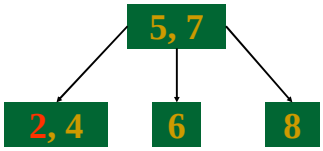
Trees: 2-3-4 Tree Solution

- Insert: 4, 5, 6, 7, 8, 9, 2 Remove: 9



Trees: 2-3-4 Tree Solution

- Insert: 4, 5, 6, 7, 8, 9, 2 Remove: 9



Priority Queues – Problem

- Add 9, 76, 54, 3, 33, 21 to a max heap, using only the array based representation

Priority Queues – Insertion

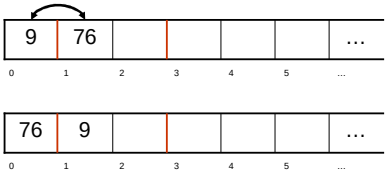
- Insert at the last position in the heap
- Reheapify up: if the element is greater than its parent, swap them and repeat
- For an element at position n, its children are at 2n+1 and 2n+2
- For an element at position n, its parent is at floor[(n-1)/2]

Priority Queues – Solution

- Add 9, 76, 54, 3, 33, 21 to a max heap, using only the array based representation

Priority Queues – Solution

- Add 9, 76, 54, 3, 33, 21 to a max heap, using only the array based representation



Priority Queues – Solution

- Add 9, 76, 54, 3, 33, 21 to a max heap, using only the array based representation

76	9	54				…
0	1	2	3	4	5	…

Priority Queues – Solution

- Add 9, 76, 54, 3, 33, 21 to a max heap, using only the array based representation

76	9	54	3			…
0	1	2	3	4	5	…

Priority Queues – Solution

- Add 9, 76, 54, 3, 33, 21 to a max heap, using only the array based representation

76	9	54	3	33		…
0	1	2	3	4	5	…

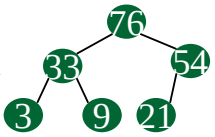
76	33	54	3	9		…
0	1	2	3	4	5	…

Priority Queues – Solution

- Add 9, 76, 54, 3, 33, 21 to a max heap, using only the array based representation

76	33	54	3	9	21	…
0	1	2	3	4	5	…

Tree Representation



Priority Queues – Problem

- Remove the max from the heap

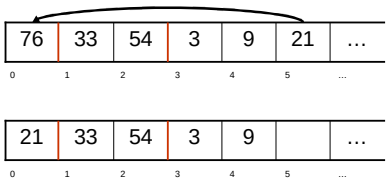
76	33	54	3	9	21	…
0	1	2	3	4	5	…

Priority Queues – Removal

- Replace the max element with the last element in the heap
- Reheapify down: if one or both of its children is larger than it, swap with the larger of the children and repeat
- For an element at position n, its children are at 2n+1 and 2n+2
- For an element at position n, its parent is at floor[(n-1)/2]

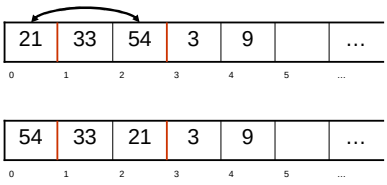
Priority Queues – Solution

- Remove the max from the heap



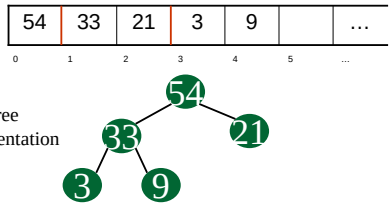
Priority Queues – Solution

- Remove the max from the heap



Priority Queues – Solution

- Remove the max from the heap



Hash Table Problem

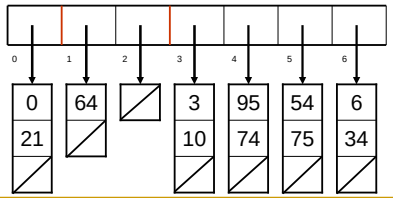
- Draw the structure of a size 7 hash table after insertion of keys with the following hash codes: 0, 95, 21, 6, 64, 74, 3, 54, 34, 75, 10.

Hash Tables

- High-level idea – 2 components
  1. Big array called *hash table* of size M
  2. Function *h* which maps keys to integer values
- For (key, item), use h(key) % M to find location of item in table
- Linked list in each entry that stores all items that map to that location (chaining)

Hash Table Solution

- Draw the structure of a size 7 hash table after insertion of keys with the following hash codes: 0, 95, 21, 6, 64, 74, 3, 54, 34, 75, 10.



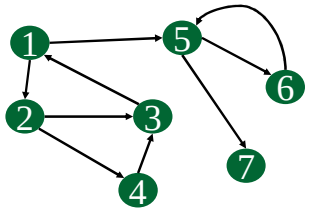
Searches (BFS and DFS)

- BFS uses a queue, DFS uses a stack

```
public void BFS/DFS(Node start) {
    Queue/Stack s = new Queue/Stack();
    s.enqueue/push(start);
    while (!s.empty()) {
        Node n = s.dequeue/pop();
        mark(n);
        for (all children that are not yet marked) {
            s.enqueue/push(child);
        }
    }
}
```


Searches (BFS and DFS) Problem

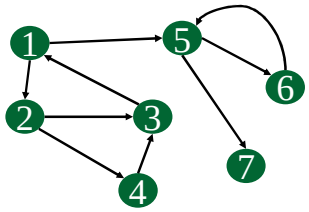
- Perform BFS and DFS on the graph, starting at node 1



Searches (BFS and DFS) Solution

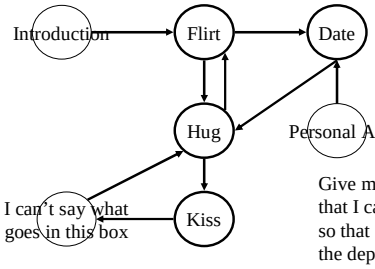
- Perform BFS and DFS on the graph, starting at node 1

BFS
1
2
5
3
4
6
7



DFS
1
2
3
4
5
6
7

Example graph

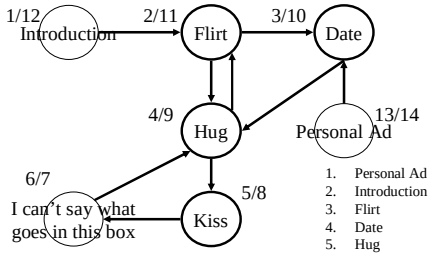


Give me an ordering that I can do these in so that I don't violate the dependencies (or my date).

Topological Sort

- Topological sorting gives us an ordering which won't violate these dependencies.
  - Perform a DFS from each source (root), marking start and finish times.
  - Now, our ordering is simply the nodes we visited in decreasing finishing time.

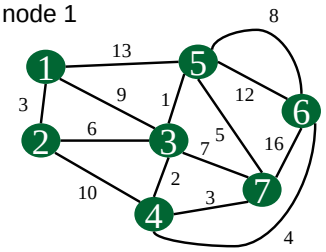
Example graph



1. Personal Ad
2. Introduction
3. Flirt
4. Date
5. Hug
6. Kiss
7. Umm…

Dijkstra's Algorithm Problem

- Find the shortest distances to each node from node 1

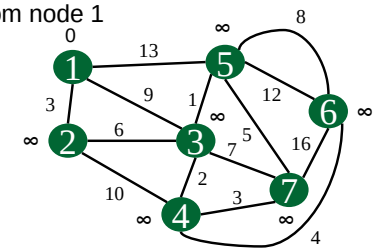


Dijkstra's Algorithm

- Set all distances initially to ∞, except the start node, which should be set to 0
- Construct a min priority queue of the nodes, with their distances as keys
- Repeatedly remove the minimum element, updating each of its adjacent node's distances if they are still in the queue and if the updated distance is less than the current distance

Dijkstra's Algorithm Solution

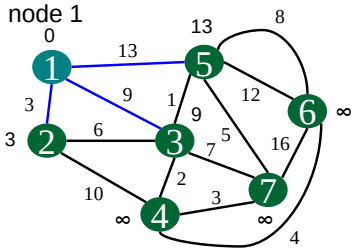
- Find the shortest distances to each node from node 1



P.Q.
1 (0)
2 (∞)
3 (∞)
4 (∞)
5 (∞)
6 (∞)
7 (∞)

Dijkstra's Algorithm Solution

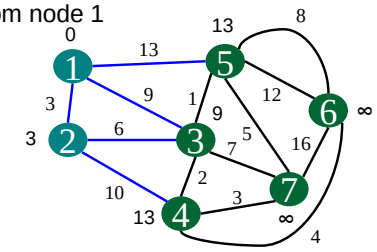
- Find the shortest distances to each node from node 1



P.Q.
x1 (0)
2 (3)
3 (9)
5 (13)
4 (∞)
6 (∞)
7 (∞)

Dijkstra's Algorithm Solution

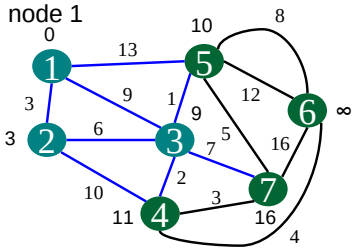
- Find the shortest distances to each node from node 1



P.Q.
x1 (0)
x2 (3)
3 (9)
5 (13)
4 (13)
6 (∞)
7 (∞)

Dijkstra's Algorithm Solution

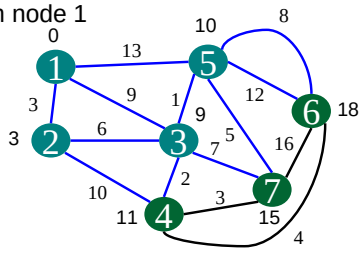
- Find the shortest distances to each node from node 1



P.Q.
x1 (0)
x2 (3)
x3 (9)
5 (10)
4 (11)
7 (16)
6 (∞)

Dijkstra's Algorithm Solution

- Find the shortest distances to each node from node 1



P.Q.
x1 (0)
x2 (3)
x3 (9)
x5 (10)
4 (11)
7 (15)
6 (18)

Dijkstra's Algorithm Solution

- Find the shortest distances to each node from node 1

P.Q.
x1 (0)
x2 (3)
x3 (9)
x5 (10)
x4 (11)
7 (14)
6 (15)

Dijkstra's Algorithm Solution

- Find the shortest distances to each node from node 1

P.Q.
x1 (0)
x2 (3)
x3 (9)
x5 (10)
x4 (11)
x7 (14)
6 (15)

Dijkstra's Algorithm Solution

- Find the shortest distances to each node from node 1

P.Q.
x1 (0)
x2 (3)
x3 (9)
x5 (10)
x4 (11)
x7 (14)
x6 (15)

Kruskal's Algorithm Problem

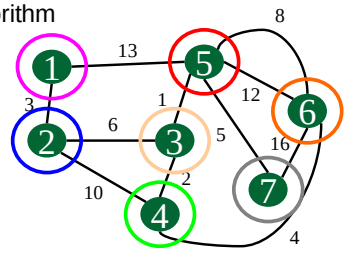
- Find the MST of the graph, using Kruskal's Algorithm

Kruskal's Algorithm

- Put each node into a set by itself
- Sort all the edges in ascending order by their weights
- Pick the least-weight edge, if the edge connects two nodes in different sets, add the edge to the MST and merge the two sets

Kruskal's Algorithm Solution

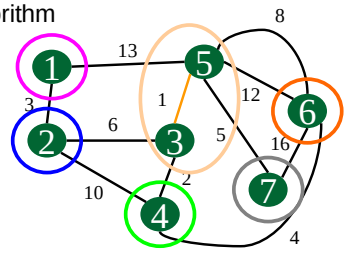
- Find the MST of the graph, using Kruskal's Algorithm



Edges
3-5 (1)
3-4 (2)
1-2 (3)
4-6 (4)
5-7 (5)
2-3 (6)
5-6 (8)
2-4 (10)
5-6 (12)
1-5 (13)
6-7 (16)

Kruskal's Algorithm Solution

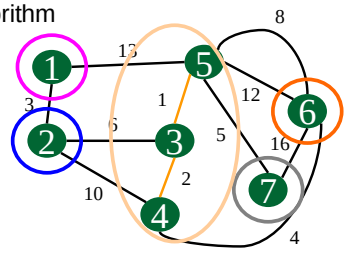
- Find the MST of the graph, using Kruskal's Algorithm



Edges
x3-5 (1)
3-4 (2)
1-2 (3)
4-6 (4)
5-7 (5)
2-3 (6)
5-6 (8)
2-4 (10)
5-6 (12)
1-5 (13)
6-7 (16)

Kruskal's Algorithm Solution

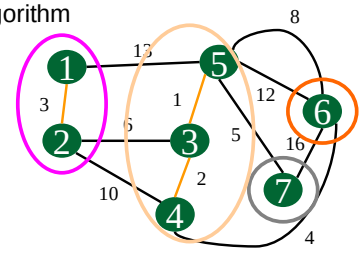
- Find the MST of the graph, using Kruskal's Algorithm



Edges
x3-5 (1)
x3-4 (2)
1-2 (3)
4-6 (4)
5-7 (5)
2-3 (6)
5-6 (8)
2-4 (10)
5-6 (12)
1-5 (13)
6-7 (16)

Kruskal's Algorithm Solution

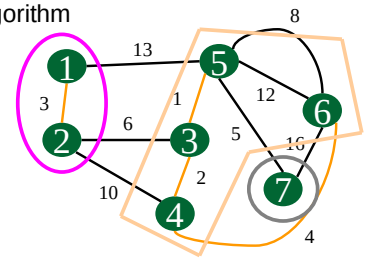
- Find the MST of the graph, using Kruskal's Algorithm



Edges
x3-5 (1)
x3-4 (2)
x1-2 (3)
4-6 (4)
5-7 (5)
2-3 (6)
5-6 (8)
2-4 (10)
5-6 (12)
1-5 (13)
6-7 (16)

Kruskal's Algorithm Solution

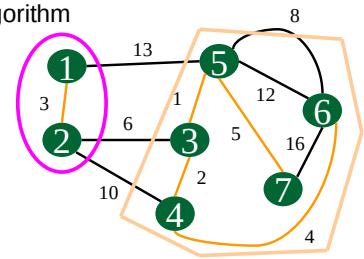
- Find the MST of the graph, using Kruskal's Algorithm



Edges
x3-5 (1)
x3-4 (2)
x1-2 (3)
x4-6 (4)
5-7 (5)
2-3 (6)
5-6 (8)
2-4 (10)
5-6 (12)
1-5 (13)
6-7 (16)

Kruskal's Algorithm Solution

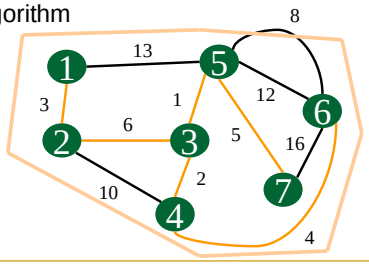
- Find the MST of the graph, using Kruskal's Algorithm



Edges
x3-5 (1)
x3-4 (2)
x1-2 (3)
x4-6 (4)
x5-7 (5)
2-3 (6)
5-6 (8)
2-4 (10)
5-6 (12)
1-5 (13)
6-7 (16)

Kruskal's Algorithm Solution

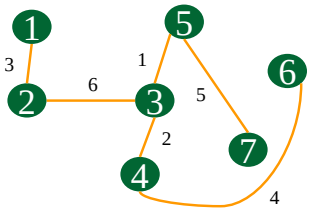
- Find the MST of the graph, using Kruskal's Algorithm



Edges
x3-5 (1)
x3-4 (2)
x1-2 (3)
x4-6 (4)
x5-7 (5)
x2-3 (6)
5-6 (8)
2-4 (10)
5-6 (12)
1-5 (13)
6-7 (16)

Kruskal's Algorithm Solution

- Find the MST of the graph, using Kruskal's Algorithm



Disjoint Sets – Problem

Given the following array representation of a disjoint sets data structure:

[2 5 -3 5 3 -3 2 2 3 0]

a) Draw the forest that this array represents.
b) Give a sequence of union and find operations whose execution will convert the array to:

[2 5 -3 2 5 2 2 2 2 0]

Disjoint Sets – Solution

```
 0 1  2 3 4  5 6 7 8 9
[2 5 -3 5 3 -3 2 2 3 0]
```

```
    2              5
   / | \          / \
  0  6  7        1   3
 /                   / \
9                   4   8
```


Disjoint Sets – Solution, cont.

```
    2              5
   / | \          / \
  0  6  7        1   3
 /                   / \
9                   4   8
```

[2 5 -3 2 5 2 2 2 2 0]

```
  -----2-------
 /   / | \  \   \
0   6  7  5  3   8
|         / \
9        1   4
```

Find(4)

Union(2 , 5)

Find(8)

Sorting

- Given the following steps, which sorting algorithms were used in each case?

13 27 89 26 9 37 5 1 38
1 27 89 26 9 37 5 13 38
1 5 89 26 9 37 27 13 38
1 5 9 26 89 37 27 13 38
1 5 9 13 89 37 27 26 38
1 5 9 13 26 37 27 89 38
1 5 9 13 26 27 37 89 38
1 5 9 13 26 27 37 89 38
1 5 9 13 26 27 37 38 89

13 27 89 26 9 37 5 1 38
13 27 26 9 37 5 1 38 89
1 13 27 26 9 37 5 38 89
1 5 13 27 26 9 37 38 89
1 5 13 27 26 9 37 38 89
1 5 9 13 27 26 37 38 89
1 5 9 13 26 27 37 38 89

Sorting

Selection Sort	Quick Sort
13 27 89 26 9 37 5 1 38	13 27 89 26 9 37 5 1 38
1 27 89 26 9 37 5 13 38	13 27 26 9 37 5 1 38 89
1 5 89 26 9 37 27 13 38	1 13 27 26 9 37 5 38 89
1 5 9 26 89 37 27 13 38	1 5 13 27 26 9 37 38 89
1 5 9 13 89 37 27 26 38	1 5 13 27 26 9 37 38 89
1 5 9 13 26 37 27 89 38	1 5 9 13 27 26 37 38 89
1 5 9 13 26 27 37 89 38	1 5 9 13 26 27 37 38 89
1 5 9 13 26 27 37 89 38	
1 5 9 13 26 27 37 38 89	

Sorting

- Do a radix sort on the following sequence, showing each step

(1087 643 2532 954 8174 65 340 1752)

Sorting

- Step 1: sort by ones place

(1087 643 2532 954 8174 65 340 1752)

(340 2532 1752 643 954 8174 65 1087)

Sorting

- Step 2: sort by tens place

(340 2532 1752 643 954 8174 65 1087)

(2532 340 643 1752 954 65 8174 1087)

Sorting

- Step 3: sort by hundreds place

(2532 340 643 1752 954 65 8174 1087)

(65 1087 8174 340 2532 643 1752 954)

Sorting

- Step 4: sort by thousands place

(65 1087 8174 340 2532 643 1752 954)

(65 340 643 954 1087 1752 2532 8174)

Skip List Problem

- Write code for searching a skip list for a key. Assume a skip list node is defined as

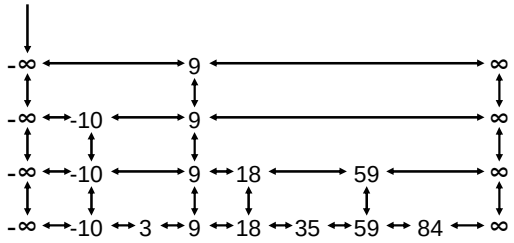
```
class Node {
   Comparable key;
   Node left, right, up, down;
}
```

and that the skip list pointer references the top left node.

Skip Lists

- 2D linked lists
- Bottom level contains all keys, and each subsequent level contains probabilistically half the keys of the previous level
- Each level starts at -∞ and ends at +∞
- The keys in each level are in ascending order

Skip List Example



Skip List Searching

- Start at top left node
- If the current key is equal to the search key, return the node
- If the next key is greater than the search key, go down and repeat search
- Otherwise go right and repeat search

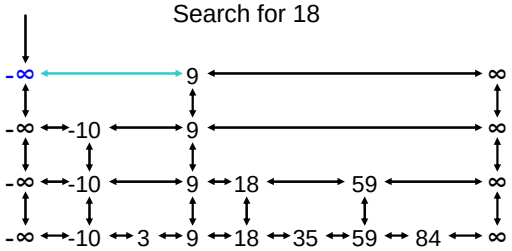
Skip List Solution

- Write code for searching a skip list for a key

```
Node search(Node n, Comparable key) {
   if (n.key.equals(key)) {
      return n;
   } else if (n.next.key.compareTo(key) > 0) {
      return search(n.down, key);
   } else {
      return search(n.next, key);
   }
}
```


Skip List Searching

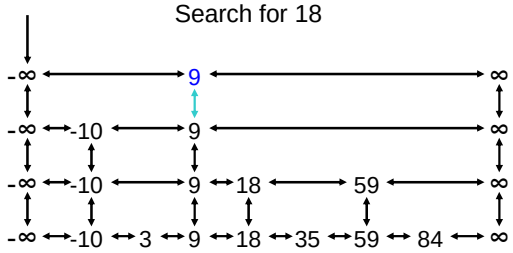
Search for 18



9 is not greater than 18, so move right

Skip List Searching

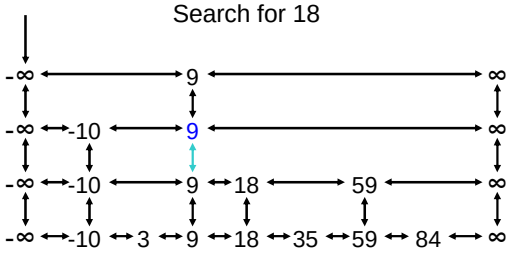
Search for 18



∞ is greater than 18, so move down

Skip List Searching

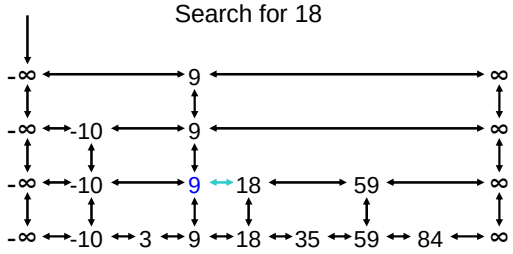
Search for 18



∞ is greater than 18, so move down

Skip List Searching

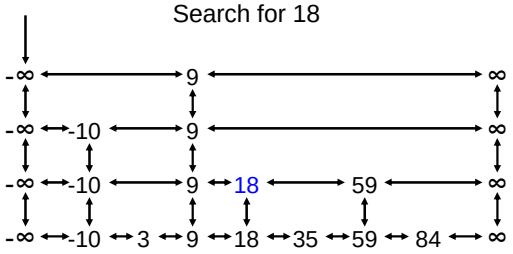
Search for 18



18 is not greater than 18, so move right

Skip List Searching

Search for 18



18 is equal to 18, so return node

Threading

- Motivations:
  - Modeling of simultaneous actions
  - Counteract I/O Latency
- Mechanism: Multiple threads of control
  - Shared memory space, multiple program counters
- Dangers:
  - Shared access to memory can result in conflicts
  - Multiple threads per processor can result in unequal time sharing (see scheduling)
- Conflict types:
  - WAR (write after read)
  - WAW (write after write)
  - RAW (read after write)
- How to avoid shared data conflicts? Locking
- Dangers of locking? Deadlock

Credits

- Thanks to CS 61b staff of
  - Fall 2001
  - Spring 2002
  - Summer 2002
- Thanks to Amir and Jack
- Thanks to
  - CMU – MIT
  - Cornell – Johns Hopkins U

for slide and example ideas

GOOD LUCK!

(and may you not need it)