**Look at code of traversals  
Sequence**

Binary Search Remove ect theory  
------------------------------------------------------------  
Trees  
-IsInternal  
**Positions**  
-replace(p,o)

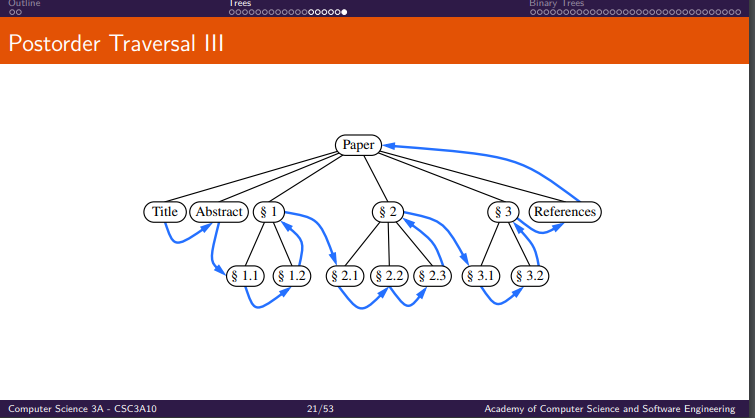
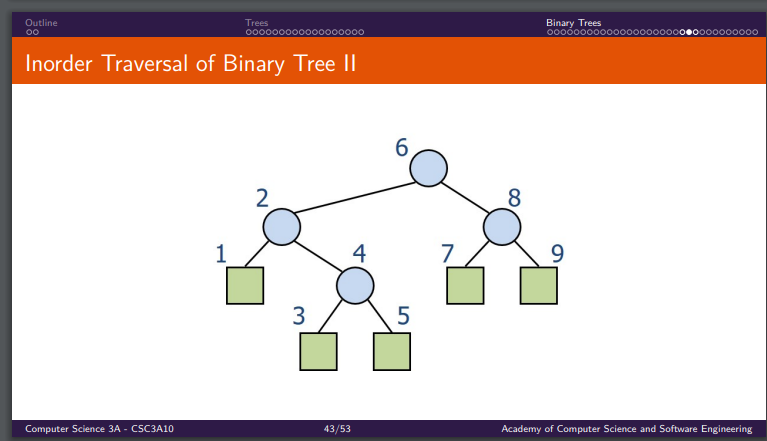
Binary Trees  
-left(p)  
-right(p)  
-hasLeft(p)  
-hasRight(p)  
-Also uses positions like Trees  
Proper if every node has 2 children  
  
For array Binary tree; each node has rank;   
root rank = 1  
left node rank = 2\*rank parent   
right node rank = 2\*rank parent +1

**Performance:**

inOrder: Most left, from bottom -> RIght  
PostOrder: Visit node after descendends (have visit after loop)  
PreOrder: Visit node then its descendents (have visit before loop)  
Euler: Left, parent, right

A screenshot of a computer

Description automatically generated with medium confidence

A computer screen shot of a diagram

Description automatically generated with low confidence

**Priority Queues**

**NB**Allows arbitory insertions  
Collection of prioritized elements  
Supports removal of elements in order of priority  
Stores elements according to Priority  
Exposes no notion of position to User  
Key  
-Not necessaryily unique  
-Can be any type   
-paired with a value, has a weight for priority

**Methods  
insert(k,o)  
removeMin()  
min()**

Performance:   
Sorted:   
Insert O(n)  
removeMin + min O(1)  
  
  
Unsorted:  
Insert O(1)  
removeMin + min O(n)

**Total Order Relations:**1) Keys can be arbitory objects on which an order is defined  
2) There can be duplicate keys (Two distinct entries with the same key value)  
3)  
A picture containing text, font, screenshot, algebra

Description automatically generated

-selection sort (This is Priority Queue with a unsorted list)  
O(n^2): Unsorted list has O(n^2) removals, and O(n) inserts  
  
-Insert Sort (This is Priority Queue with a sorted list)  
O(n^2): Sorted list has O(n) removals, and O(n^2) inserts  
  
In Place Insertion: Uses hops instead of modifying sequence  
   
**+ Heaps**Key of parent must be smaller than key of child

Height: log(n)

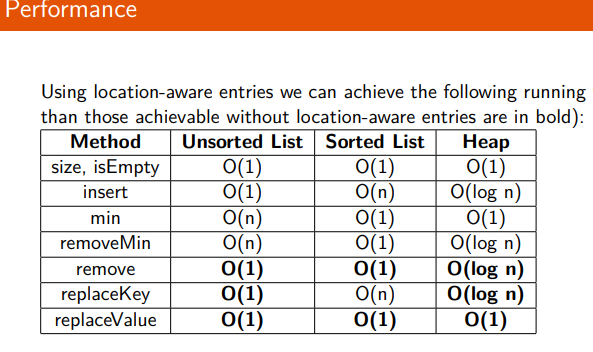
Performance:  
(Heap sort)  
Space: O(n)  
Insert+removeMin: O(logn)  
min + auxilory: O(1)  
  
heap Sort: O(nlogn)  
-This is faster than insertion and selection sort O(n^2)

**In heaps**: You keep track of the last inserted node  
-So when you remove a node, the root is replaced with that last inserted node.  
-You then downheap

Bottoms up construction: start at very bottom row of nodes, then add next layer, downheap, repeat.

**Adaptable priority Queue**

**-**remove(e)  
-replaceKey(e,k): Replace e’s key, with k; then return e’s key  
-replaceValue(e,v): Replace e’s value with v, then return old v

To do this, in addition to key and value, rank is stored (or position) to better find entry.  
  


**Maps**

**Keys Unique**

get(k)  
put(k,v): replace existing value at k returning old value; or add new value returning null

remove(k)  
keys()  
values()  
  
-remember: can use set(p, e)

Performance:   
put: O(1)  
get and remove: O(n)

**and Hashtables**

Goal: disperse keys “randomly”  
-Compression functions: key to k mod 13

Collision:  
Separate Chaining: Linked list in each cell  
-uses lots of memory  
  
Linear proving: +1  
-An issue with this is long chains of clumped keys in an area

Double Hashing: using p – k mod p; p < N and p = prime

**Compression functions**

Division  
- k mod N, where N is usually prime  
  
Multiply, Add and Divide (MAD)  
- (ak+b) mod N  
- a and b are positive integer, such that a mod N != 0  
-as all would map to be in that case

**Hash Codes**

Memory Address

-Interpret memory address as integer   
-good in general: not good for numeric/string keys

Integer Cast

-Interpret the bits of the key as an integer   
-good got key lengths less than or equal to num bits of an integer  
ie byte, short, int, float

Component Sum  
-Partition bits of key into components of fixed length, and sum components (ignore overflow)  
-Good for numeric keys of fixed length greater than or equal to number of bits of an inter (long and double)

Polynomial accumulation  
-Partition bits of key into sequence of components of fixed length (eg 8, 16, 32 bits) a0 a1…an-1  
-We evaluate the polynomial p(z)= a0 + a1z + a2z^2+…+ an-1 z^n-1 at a fixed value z, ignoring overflows  
-Especially suitable for strings

**Performance:** Searches, insertions and removals take O(n)  
Load factor: n/N  
  
Expected run time of all dictionary hash table operations is O(1).

**Dictionaries**

Dictionary is a map, which can have duplicate key values

Find(k): first occurrence  
findAll(k)  
insert(k,v): inserts and returns entry  
remove(e): remove e and return e  
entries()

List based:   
Insert: O(1)  
removal + find : O(n)

Array performance: (this is a **search table**)  
find: O(logn)  
insert: O(n)  
remove: O(n)F  
-Only used if searching is most used or small dictionaries

**and skiplists**Quad node: left, right, above, below + value  
Size: <2n so O(n)  
  
Search: high prob of logn  
  
Height: high prob of logn  
  
That’s why worst case is **O(n)**

**Binary Search Trees**

First()  
last()  
successors(k)  
predecessors(k)

Search Table:  
Array-based sequence of entries, sorted by key  
Performance:  
find: O(logn)  
insert: O(n)  
remove O(n)

Binary search has parent node, left (smaller) and right (bigger)  
  
To perfom insert(k,v), first get key from TreeSearch

Performance of binary  
Space: O(n)  
find,insert, removeL O(**h**)  
  
height Is O(**n**) in worst case, ad O(logn) in best

**AVL Trees**

These are balanced binary trees

**Find, insert, remove are O(logn)**