CSCA48

Ta:

Email: [trooper.cooper@utoronto.ca](mailto:trooper.cooper@utoronto.ca)

ADT: abstract data types

Binary conversion:

I.E 5

5/2 = R1

2/2 = R0

½ = R1

Read bottom to top

::-1 = reverse list

UML:

Name

Attributes

Methods with givens required and return

Path is the nodes to reach a node

The height is the longest path

Depth is the length of a unique path from root -> node

Priority Queue:

A queue in which each container has a priority value

And when looking for values you retrieve the highest priority

Generally we just return the first with lower priority

Implementation can be done with a heap

A heap is a tree sorted by priority

Min heap is lowest goes first, max heap is highest goes first

In a heap the least number is always on top

When adding then we just add in an order such that the lowest numbers are closer to the root and the tree is balanced. When extracting we move the next smallest to the previous position

Time complexity of the algorithm is the function f:N ->N where f(n) is the maximum number of steps that the algorithm can take when running with a input of n

We compare two algorithms using runtime

Big-OL given two functions we can express one as being an asymptotic upper bound of the other within a constant factor

* There exists c > 0 s.t. for all n>= B, f(n)<= c\*g(n)
  + Where c is a constant factor and B is the point where g(n) > f(n)

The above point may not exist

Some ways of writing this

* F is BigO of g
* F = O(g)
* F set O(g)
* F is order g

So what we have is that if we have two functions then f(n) and g(n) and f is BigO of g then we say that A is at least as efficient as B

We say that g is a tight asymptotic bound of f, iff f = O(g) and g = O(f) algorithms whose running times are tight asymptotic bounds of each other are said to be equally efficient

Note about notationL

Suppose f and g are defined by

f(n) = 3n

and g(n) = 5n+4

then 3n = O(5n+4)

c = 1, b = 1 then For all n>=1 3n <= (5n+4)

consequently the opposite is impossible as we cannot chose a c or B such that 3n >= 5n+4

Working with BigO

1. If f(n) = C\*g(n) where C is some constant then f = O(g) [and also g = O(f)]
2. If f(n) = t\_1(n) + t\_2(n) + … + t\_k(n) and each t\_i = O(t\_1) then O(t\_1) [and also t\_1 = O(f)]

Put another way if a function is the sum of multiple terms then it is BigO of the dominant term

By convention we try using the simplest function

I.E. instead of O(5n^3 + 6logn + 13) = O(n^3)

Analyzing BigO

1. We must first define n
2. List then n = len(list)
3. String n = len(string)
4. Integer n = int
5. Any code that does not loop or make function calls runs O(1) time

BEWARE: to simplify our analysis we make certain assumptions here, for example we say that integer addition runs in constant time and this is obviously untrue

1. Consider a loop if each iteration of the loop runs in O(f) time and the loop iterates O(g) times then the whole operation is O(fg) time
2. Recursive? Not bothering for now

Some searches:

Linear is well linear

Binary is split in half and then compare and do it again

Some sorts:

Bogosort: the just throw shit in the air method

Bubblesort: swap if not in order

HeapSort: put it into a heap (O(nlog(n))), extractions (O(n log(n)))

Total = O(2nlogn) = O(n log n)

BinarySearchTree: put it into a search tree

Insertion:

Insert values into new list one by one

QuickSort:

Choose a key value (usually middle) and move everything greater to the right and lower to the left.

O(n log n)

Recurse with the different sides

Merge sort, split list to singular bit and reconstruct so that lowest is at one end and greatest is at the other

This works as at some middle list we are comparing between the two lists choosing the smallest and placing them one by one. If you need a diagram check wiki they have a great gif

O(n log n)

What is a stable sort? (a sort that is stable?)

Stable sorts are sorts that also sort to a second degree if necessary

Remember different sorts have massive advantages and disadvantages