CS2521 Algorithms

Set is a collection of distinct items, order is irrelevant. Cannot contain multiple copies of same objects.

Permutation – orderings of some items. (1,2,3,4) and (4,3,2,1) are two permutations of the same integers.

Tuple – ordered set of objects with the same size and type. (a,5), (c,9). Two- pair, three – triple, n-tuple. Cartesian product of n sets is the set of n-tuples.

Subset – selection form a set of items. Subsets are also sets.

Trees- hierarchical relationships, like object DOM. They are a type of graph.

Graphs represent relationships between pairs of objects. Can be directed or undirected, have nodes and edges.

Point – locations in space

Polygons – regions in space

Intervals – regions on a line.

Strings – represent sequence of characters or patterns.

Recursion:

Most objects are composed of smaller versions of themselves, into a base case – atomic object. E.g. an empty string.

E.g. removing characters from a string results in a shorter string.

E.g. Point set – a single point.

These recursive definitions are useful for proofs – induction.

# 2- Analysis

Measure efficiency – time, operations, memory, network usage.

We want our analysis to be machine independent.

We use the RAM model – we assume a single processor with unlimited memory.

Simple operations (+,-,=,if etc) takes on step.

Memory access takes one step.

Loops and subroutines take as many steps as they run in total.

You measure time by counting the steps an algorithm uses.

However, the RAM model is unrealistic.

E.g. multiplication takes longer than addition.

Or loop unrolling.

L1 cache is closer to the CPU, L2 is slower and L3 is slower.

Disk is slowest.

The ram model is still used for a certain level of abstraction.

To evaluate whether an algorithm is good, bad or ugly we must see how it works over all possible instances.

For sorting, input instances for all arrangement of n keys and all possible values of n.

Worst case is most important. Best, worst average case e.g. when playing poker machines. Complexities can be represented as a numerical function of time.

Travelling salesman problem.

O – notation

Multiplicative differences are implementation dependant – python vs assembly.

Upper down

Lower down

Type down

O-notation can be used to compare algorithms.

## Contiguous and linked data structures

Linked – lists, trees: elements of the data structure hold references to other elements. Made up of distinct chunks of memory.

Contiguous – arrays, heaps, hash tables. Occupies a single chunk of memory.

Arrays are essential.

Located in memory, find the first element at position 0x5555 in memory, each element is 4 bytes long. The 5th element will be in position 0x5559. Computers can typically retrieve any element by its address in constant time.

Advantages:

Constant time access by index.

Space efficient.

Continuous in memory, fast cache access to later elements.

No ability to alter size dynamically.

When it gets too long.

Copy all elements to new array.

Resizing arrays is very common. A better approach: Now adding elements requires log n resizes of the array. When you add a new element, you have to copy all the stuff over. Better: double the size of the element.

## Pointers and linked structures

Pointers- represent memory address of a data structure.

Linked list- each node holds a value and a pointer to the next node.

We can search, insert and delete.

Singular linked list –hold pointer value for next value.

Double linked list – hold pointer value for next and previous value.

Head of the list – start of linked list.

Searching through:

Take in node and a value.

If it’s the current node, return that -> good

Otherwise, go to the next node and check that. Pass the address and the value.

Insert into linked list: take node, value and index

Index = are we where we want to place the value.

Decrement the index by 1, then go through the nodes, next node and check if index is 0. If it is, insert the value here.

When the node is inserted, you make a new node which points to the next node and ALTER the pointer of the previous node to the new node. Ie laita se sinne väliin.

Deleting:

Find where to delete, and make previous node ‘skip’ the deleted node, make it point to the next node.

All functions use recursion.

Linked lists have many advantages over arrays.

Disadvantages: pointers require extra space. If only integers are stored, space used is doubled.

Cannot find n element without going through all lists. -> no random access

Retrieve an element n is an order of n operations.

Poor cache performance.

## Stacks and queues

Stacks are LIFO (Last in first out)

Queues are FIFO (First in first out)

Stack: Push & Pop like in assembly

Queue: Enqueue and dequeue

Call stack: Java you can call a method within a method within a method. Call stack pops to original method.

Queues are used for CPU cycle scheduling.

Dictionary: i.e. phonebook

Associate data with a key.

We can search a dictionary with a key.

We can insert data with a key.

We can delete data with a key.

Create an array of (key, value) pairs and store n, the maximum index.

Searching goes through n is an n order operation.

Sorted dictionaries:

Binary search- much faster.

Tune how dictionary is built on array based on how dictionary will be used. For algorithm runtimes.

Complexities change if we used linked lists vs arrays.

# Trees

Rooted binary tree is either:

- empty

- a node

When tree ‘branches’ split, it splits to the left or right based on the size of the values.

Allow search, insert, delete.

Also allows traverse- moving through keys in a specific ways.

Associates a key with each node such that left.key<root.key<right.key

Sorting

Good sorting algorithms run in O(nlogn)

Naïve ones: O (n2)

Big and small set, m >> n

Find a value that is the same in both

Bad algorithm: Loop through all values in m, all values in n and compare.

Sort the big set, do a binary search with smaller set. Or vice versa.

Better way:

Compare smallest/largest values of m/n and compare to the smallest/largest values in m/n. If it is, remove it and move to the next one. I.e. Cut down values that you know do not match.

Best: Sort elements of small set. Do a binary search with the big set through small set.

This is sort of counter intuitive, but it is the most effective.

Should we sort in increasing or decreasing order?

Sort just the key or entire record?

Equal keys?

Non-numerical data?

Assume the existence of a comparison function

Selection sort:

Iterate over all values.

Find smallest one, set it to first element of array.

Then go to the next value after the first one and swap it with min. Add on.

Use one array instead of two!

Complexity of selection sort: n2 because every element is compared with every other element.

## Can we do better?

Yes, with complexity O(log n) of a heap: -> O(n log n)

A heap keeps a loose ordering.

Data structure that supports priority queue operations insert and find/delete-min.

The each element dominates its children.

In a min-heap a node dominates its children by containing a smaller key.

A max-heap node dominates its children by containing a larger key than they do.

I.e. A tree that always splits into 2 with larger and smaller.

The left child of an element at index k sits at position 2k in the tree, the right child at 2k + 1

The parent of k at position [k/2]

What’s the catch?

All missing nodes take space as the full tree must be represented.

Worst case is array of size 2n to store n elements.

A heap is not a binary search tree. Cannot do binary search.

Have to do a linear search.

We can bubble a heap by putting a value to the bottom of the heap tree, goes up until it does not dominate branch values anymore.

Selection sort using heaps.

Heap sort is basically selection sort with a more efficient way of finding the minimum value.

An array is almost a heap, without the heap property.

There are at most [n/2h+1] nodes of height h

Sorting (cont)

## Comparison sort

Decision trees: a full binary binary tree that represents the comparisons between elements that are performed by a sorting algorithm operating on an input of a given size.

Leaves must represent any possible ordering of the list, so n! leaves.

Each leaf must be reachable from the root, as every possible ordering is possible.

The length of the simple path form the root of a tree to any of its reachable leaves represents the worst case number of comparisons that the corresponding sorting algorithm would perform.

This is equivalent to the height of the tree.

Assume a tree of height h with l reachable leaves for a sort on n elements.

n! =< I =< 2h

Upper bound:

log(n!) <= nlogn

log(n!) = O(nlogn)

Heapsort and mergesort are O(nlogn) they are asymptotically optimal comparison sorts

To do better, we shouldn’t do comparison.

Counting sort:

Assumes that all inputs are in between 0 and an integer k

If k = O(n) it runs in theta(n) time.

Counting sort takes into account how many times numbers are encountered.

Radix sort

Sort most significant digit first, 2nd most significant etc.

This requires many intermediate piles.

Radix sort sorts on the least significant digit.

Cards are gathered from various bins in order and sort proceeds on 2nd least significant digit.

Graphs