Algorithms and Data Structure

Lecture 1. Introduction to Algorithms

Dangers of Recursion

* Elegance vs. Pragmatism
* Missing base case
* No guarantee of convergence
* Excessive memory requirements

Lecture 2. Searching & Sorting

Fisher-Yates Shuffle: A correct random shuffle algorithms.

Selection Sort

* Efficient in terms of swaps
* May be relevant if very slow memory.

Insertion Sort

* Comparisons are only made against the sorted subcollection on the left.
* Generally the preferred brute-force sorting algorithm.

Lecture 3. Analysis of Complexity.

Simplifications

* Each operations has the same cost (c)
* Find the basic operation of an algorithm. Find the most important, most executed operation.
* Determine its order of growth with respect to input size n.

Common Running Times.

* Linear Time (n)
* Constant (1)
* Linarithmic Time (n log n)
* Logarithmic (log n)
* Quadratic Time (n^2)
* Cubic Time (n^3)
* Exponential Time (2^n) and Factorial ($n!$)

Asymptotic Notation.

* O-notation: Establishes an upper bound.
* Ω-notation: Establishes a lower bound.
* Θ-notation: Establishes a tighter upper and lower bound, determined by constants c_1, c_2 for $g(n)$.

Analysing Runtime Process

1. Decide upon the input parameter indicating size.
2. Identify the basic operation.
3. Check if basic operation depends only on input size.
   - sum(n) only depends on the magnitude of n.
   - Sort(n) needs worst, best and average cases considered
4. Set up a summation expression.
5. Simplify the summation to determine order of growth.

Experimental analysis results are not general.

Lecture 4.a: Correctness & Time Complexity.

Loop Invariants: good properties

*Initialisation: The loop invariant must be true before executing the loop.

*Update: If the loop invariants is true in one iteration, it must also be true in the next iteration.

*Conclusion: Loop invariant will say something useful about the conclusion of the loop.

Mathematical Induction:

*Base Case *Induction step *Proof.

Determining Complexity of Recurrence using the Master Theorem

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n>1 \\ d & \text{if } n=1 \end{cases}$$

n = problem size

a = number of sub-problems, or branches in the recursion tree.

b = fractional size of the sub-problem, within each branch

c = exponential of additional work to be done

d = some constant

We can produce an estimate of complexity as lookup:

$$T(n) = \begin{cases} \Theta(n^c) & \text{if } a<b^c \\ \Theta(n^c \log n) & \text{if } a=b^c \\ \Theta(n^{\log_b a}) & \text{if } a>b^c \end{cases}$$

Lecture 5.a: Polymorphism

For good OO Design

*Choose appropriate entities to model as objects

- Encapsulate related information (fields) and behaviour (methods)
- Selectively expose object internals to other objects

*Form meaningful relationships.

Relationship

*Inheritance: Used for specialism & reuse

*Composition: Combinations of objects that create more complex objects. If super is destroyed then extended is destroyed.

*Aggregation: Independent objects are gathered together.

Lecture 6a. Abstract Classes and Interfaces

Lecture 6b. Collections and Stacks

We distinguish between types of linear lists depending upon the principal operations that will be performed.

Types of List

* LinkedLists: A linear list where insertions and deletions can occurs anywhere within the list.
* Queues: A linear list for which all of the insertions are made at one end of the list (tail); all delections are made at the other end (head).
* Stacks: A linear list for which all of the insertion and deletion operations are made at one end of the list (top)

Lecture 7a. Hash Tables.

Compression Function: A fast and simple function that reduces the possible range of values.

* MAD Method: $[(a * hash + b) \bmod P] \bmod N$.

  P prime number $> N$.

Managing collisions — Multiple Strategies: Space / Time tradeoff

* Option 1: We decide to sacrifice time and use search.
  - On collision, traverse the table until the next slot free is found.
  - Use that slot for the new value
  - Open addressing or Linear Probing
* Option 2: We decide to sacrifies space, and use a second data structure
  - Separate Chaining.

Separate Chaining

* We introduce a secondary linked list structure.
* Each slot can grow depending on the number of collision.
* Where collisions occur, use a doubly-linked list.

Load Factor: α = entries (n) / slots (N).

Resizing Tables: Grow the table as α approaches 1.

* Typically threshold (e.g. 0,75) as a trigger.
* Need a factor of growth.
* Rehash keys for new table.

Performance of hash table:

Average complexity of search by index, insert, deletion are all $O(1)$. Worst cases are all $O(n)$.

Lecture 7b. Amortized Analysis.

Aggregate Analysis

* $T(n)$ is total cost of a sequence of operations.
* Average cost will be $T(n)/n$: amortized cost
* All operations are considered to have the same cost.

Accounting Method.

* Imagine that each operation has a financial cost.
* Different operations have different costs.
  - Guess these through trial and error.
* Principal idea: Overcharge on basic operations.
* Invariant: We cannot go into dept.

Lecture 8a. Trees.

In-order Traversal:
- left child
- self
- right child

[Returning a sorted view]

Pre-order Traversal:
- self
- left child
- right child

[Copying the binary tree]

Post-order
- left child
- right child
- self

[Deletion].

Lecture 8b. Binary Search Tree.

BST
* Ordered binary tree
* Tree governed by the BST properties:
  - Keys in the left sub-tree are all less than k.
  - Keys in the right sub-tree are all greater than k.
* Most operations of a BST are considered fast.
  - Proportional to the height of the tree.
  - Complexity of O(n)
* Insertion is trivial, deletion is more complicated.

Lecture 9a. Graphs

Implementation of Graph API. 2 main representations.

* Adjacency matrix
  - Fast check
  - For dense graph
  - Slow change
* Adjacency list
  - Space efficient
  - Fast operation
  - Slow check
  - Slow removal

Lecture 9b. Graph Traversals

DFS & BFS Traversal

Lecture 10a. Empirical Analysis.

Lecture 10b. Quicksort.