3

Programming Wid

Definition of $T(n) = o(g(n))$

$T(n) = o(g(n))$ if there exists large enough input size n_0 such that for any $n > n_0$: $T(n) < c_{up} \cdot g(n)$

$T(n) = w(g(n))$ if there exist a large enough input size n_0, that for any $n > n_0$ $c_{low} \cdot g(n) < T(n)$

Stack.

Running cost of Linked list stack push: $\Theta(1)$ pop $\Theta(1)$ isEmpty $\Theta(1)$
Paid N memory cells to get $\Theta(1)$ push/pop

Stack: array implementation:

possible overflow, benefit => no pointers
cost pop/push/isEmpty $\Theta(1)$ worst case

Resizing array (double when full) insert best case $\Theta(1)$ worst: $\Theta(n)$ time
Amortised cost = (cost of N operations) / N starting from empty structure
push has amortised $O(1)$
Shrink by half when array quarter full

Linked List v array List for Stack

LL: push/pop take $\Theta(1)$ time in worst case
Uses N extra space for N items

Array: push/pop take $O(1)$ amortised time
push/pop worst case $O(n)$
Uses between 1 and 3N+1 extra memory for N items

4

Queue LL implementation
Worst case chyph enque: $\Theta(1)$
dequeue: $\Theta(1)$
Extra space to store N element N(+ 2 for head and tail pointers)

Union Find
array represents each node
Node value under union box 0 hue it is connected
Quick union cost Initialise Union Find
N N N
Weighted: If size i < size j rd[i] = j. size j += size i.
points to bigger tree
COST: Initialise Union Connected
N LogN logN.

Quick union with path compression: Just after computing the root of p,
set the rd of each examined node to point to that root

Algorithm	worst case time
Quick Find	MN
Quick union	MN
Weighted QU	N + MLogN
QU + path comp	N + MlogN
weighted QU + path com	$N + M\log^{*}N$

What is Binary Search Tree?
-Binary tree
- Key in each node is unique
-each node contains a key which is:
-greater than keys in left subtree
-smaller than keys in right subtree

Programming Notes

Loop Invariant.

Is a property which is true   Worst case $\Theta(n^2)$

- At beginning of algorithm
- At end of algorithm
- Before each iteration   For Insertion Sort, $1 \le j \le n$ Sorted$[1 \ldots j-1]$, unsorted$[j \ldots n]$

Insertion Sort.

5 4 6 1 2 7 3

Start at position 2, if > pos 1, swap backwards,
Keep swapping backwards, move onto next element

Asymptote Notation O, Ω, o, W

- When we are giving exact bounds we write $T(n) = \Theta(f(n))$
- When we are giving upper bounds: $T(n) \le \Theta(f(n))$ or alternatively $T(n) = O(f(n))$.
- When we are giving non-tight upper bounds: $T(n) < \Theta(f(n))$ or $T(n) = o(f(n))$
- When we are giving lower bound: $T(n) \ge \Theta(f(n))$ or $T(n) = \Omega(f(n))$
- When giving non-tight lower bounds: $T(n) > \Theta(f(n))$ or $T(n) = \omega(f(n))$

Θ bounds are the most precise asymptote performance bounds we can give
O/Ω bounds may be overly general
o/ω bounds are definitely over general

Binary Search

Check middle, re adjust midvalue for each search if key is lower or higher
if key is found hi = low = key = mean
Running time $T(n) = \Theta(\log n)$

2

Definition of $T(n) = \Theta(g(n))$

$T(n) = \Theta(g(n))$ if

- There exists large enough input size n_0 such that for any $n > n_0$: $C_{low} \cdot g(n) \leq T(n) \leq C_{up} \cdot g(n)$

  (for some constant C_{low} and C_{up})

Example Insertion Sort:

Precise running: $T(n) = (3/2)n^2 + (3/2)n - 1$

Asymptotic $T(n) = \Theta(n^2)$

For $n > 10$: $n^2 \leq (3/2)n^2 + (3/2)n - 1 \leq 2n^2$

Definition of $T(n) = O(g(n))$

$T(n) = O(g(n))$ if:

- There exists large enough input size n_0 such that for any $n > n_0$:

  $T(n) \leq C_{up} \cdot g(n)$ for some constant C_{up}

Insertion Sort: Precise: $T(n) = (3/2)n^2 + (3/2)n - 1$

Asymptotic: $T(n) = O(n^2)$

$= O(n^3)$

$= O(2^n)$

Definition of $T(n) = \Omega(g(n))$

$T(n) = \Omega(g(n))$ if:

There exists large enough input size n_0 such that any $n > n_0$

$C_{low} \cdot g(n) \leq T(n)$ for some constant C_{low}

Insertion sort: precise: $T(n) = (3/2)n^2 + (3/2)n - 1$

Asymptotic: $T(n) = \Omega(n^2)$

$T(n) = \Omega(n)$

$T(n) = \Omega(1)$