What is a D.S.?

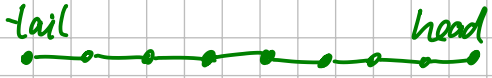
black-box approach

data = operations

D.S. — static (build, query)
D.S. — dynamic (query, modify)

D.S. — interface
D.S. — implementation

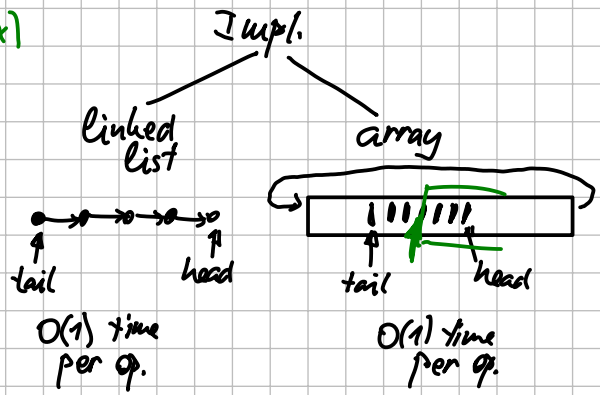
Examples



Queue

Enqueue (x)
Dequeue
Is Empty

Impl.



Stack

Set

$X \subseteq \mathcal{U}$ ← universe
↑ a finite subset

Insert (a)
Delete (a)
Find (a) a.k.a. Member, Lookup
Build (a_1 — a_n)

	Ins	Del	Find	Build
list	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
array	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
sorted array	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n \log n)$
search tree	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(n \log n)$
hash tables	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$

$\Theta(1)$ if we know that $a \notin X$

good as static

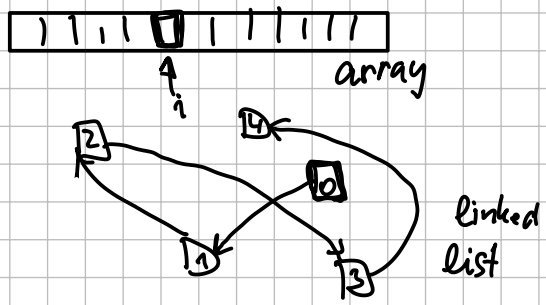
↑ average/expected

comparison model

uses arithmetics

Dictionary

{(key, value)}
↑ unique



Ordered set

Min
Max
Pred (a)
Succ (a) := $\min\{x \in X \mid x > a\}$

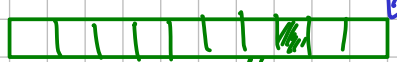
$\Omega(\log n)$
lower bound
(by bound on sorting)

Multiset

Model of Computation

Random-Access Machine (RAM)

- memory



memory cells
- contain integers
- addressed by integers

- memory allocator

variables ... fixed addresses

arrays

start address

records (structs)

- arithmetics + − * / %(modulo)
  bitwise AND OR XOR NOT << >>
- control instructions if □ < □ then jump □
- size of integers — word size (in bits)
  - sufficient to store: numbers from input, size of the input, anything polynomial in these
- time # instructions
- space # cells used

At the start memory contents are arbitrary

max (x_1 — x_n)
$\sum x_i$
n^3

Amortized Analysis

- worst-case >> typical
- idea: consider whole sequence of operations

If we insert n items

Total time spent on reallocations is

$\Theta(2^1 + 2^2 + \_\_\_ + 2^k) = \Theta(n)$

$2^{k-1} < n \le 2^k$ $\quad n \le 2^k < 2n$

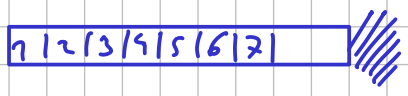
$2^k < 2n$ $\quad 2^k \in \Theta(n)$

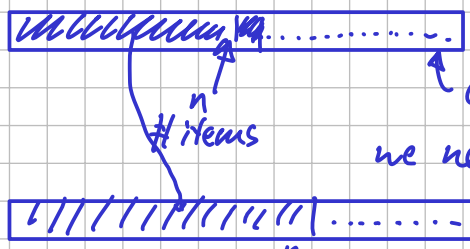
$2^0 + \_\_ + 2^k = 2^{k+1} - 1$

$1 + 10 + 100 + \ldots + 10\underbrace{\ldots}_{k}0 = 1\underbrace{\ldots}_{k+1}1$

aggregation method

Flexible Array



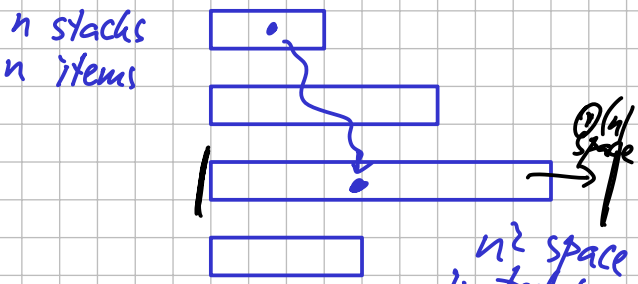


we need $n \le C$

reallocation takes $\Theta(C')$ time $\quad C' > C$

idea: $C' \leftarrow 2 \cdot C$ $\quad C, C' \in \Theta(n)$

$1, 2, 4, \ldots, 2^k$



Shrinkable Array

if $n > C$: $C' \leftarrow 2C$

if "n is small" rel. to C : $C' \leftarrow \max(1, C/2)$

$n < C/4$

Analyse total time spent by re-allocations over a sequence of Push/Pop operations

reallocations

time for realloc ending a block is $O(\#\text{ops in the block})$ → then sum over all blocks

1st block: trivial $O(1)$

at the start

if we grow: at least n pushes

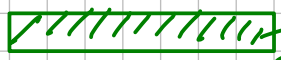
if we shrink: at least $n/2$ pops

$\Omega(n)$ operations occurred in the block

$O(n)$ time for realloc

$O(1)$ time per operation

Attempt: shrink if $n < C/2$



start with $n = C = C_0$

Push	C_0+1	$2C_0$	$\Theta(C_0)$
Pop	C_0	$2C_0$	.
Pop	C_0-1	C_0	$\Theta(C_0)$
Push	C_0	C_0	.

$\Theta(C_0)$ time for 4 operation

accounting method

- amortized time per op. is $O(1)$
- space is always $\Theta(n)$

Next week: more amortization, Splay trees