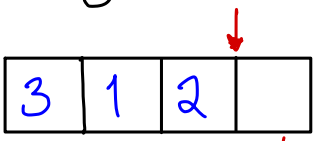
Resizable Arrays

- Python list, Java ArrayList
- Rust vector, C++ vector.
- Most basic d/s

Traditional basic idea:

- Keep a fixed array (malloc)
  - grow when full
  - shrink when too "sparse"
- Textbook solution: geometric expansion/shrinking
  aka. doubling trick.
- Recaps



full: when capacity == len:
- double capacity (or grow by $(1+\alpha)$)
- cost: • alloc • free
  • copy $= N + O(N)$
  Space: $N + \alpha N$

Too sparse:
when len $\leq$ capacity/4
(generally: len $\leq$ cap/$(1+\alpha)^2$)
- shrink cap in half (or cap/$(1+\alpha)$)

- reality: $\alpha \simeq 0.25 \to 1$.

Ultimate idea: after a costly action, the array must be only moderately occupied so that the next costly op is "a while" away.

Analysis: "amortized cost" — pretend to charge a costly op. to other less expensive ops.

- to grow to $(1+\alpha)N$.
  ① Full with cap = N
  ② Must have been $N - \frac{N}{1+\alpha}$ appends
  ③ Cost to grow = N

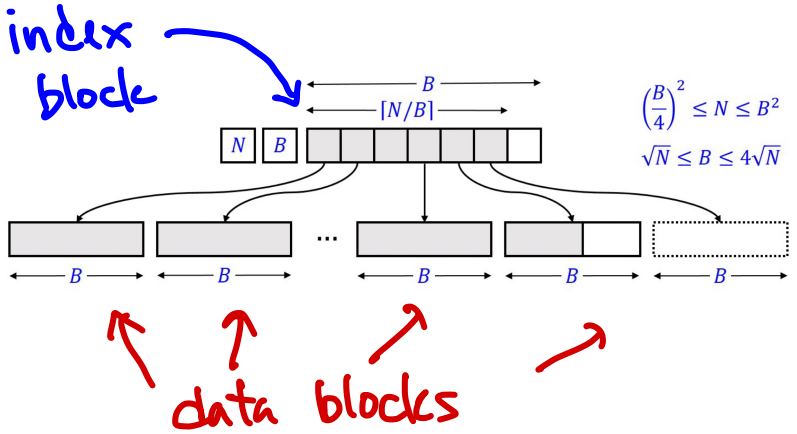
$$\frac{N}{N - \frac{1}{1+\alpha}N} = \frac{1+\alpha}{\alpha}$$

- to shrink to $\frac{N}{1+\alpha}$.
  ① Sparse with $N = \frac{N}{(1+\alpha)^\alpha}$
  ② Must have had $\frac{N}{1+\alpha} - \frac{N}{(1+\alpha)^2}$ deletes
  ③ Cost to shrink: $\frac{N}{(1+\alpha)^\alpha}$

↪ $\frac{1}{\alpha}$ per op.

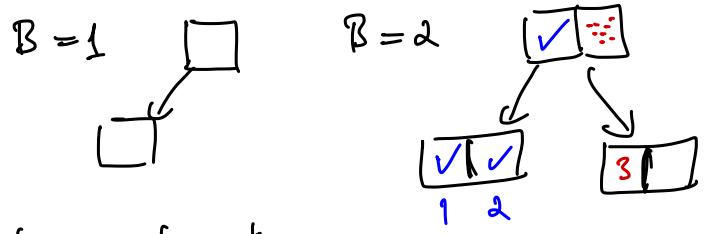
Sitarski's idea (1996) – Hashed Array Tree (HAT)

- No hashing & no trees



B is the control knob. (relating to N).

Small examples:



Add: May need to create a new data block

→ Nice if B is a power of 2 : if $B = 2^k$

Access: a[i] → Block[i / B] (i >> k)

↳ index in the block = i % B (i & (1 >> k -1))

Space: $N + O(\sqrt{N})$

Full → Grow When no more room, i.e. $N = B^2$.

→ Double B & copy data over

Note: Once B is doubled, capacity $= (2B)^2 = 4B^2 = 4N$.

Too sparse → Shrink When $N \le \left(\frac{B}{4}\right)^2$

→ Halve B & copy data over

Note: Once halved, cap $= \left(\frac{B}{2}\right)^2 = \frac{B^2}{4} = 4 \cdot \frac{B^2}{16} = 4N$

Analysis:

- grow costs ~N to copy & otherwise.
- Must have had $B^2 - \left(\frac{B}{2}\right)^2 = \frac{3}{4}N$ ops before grow
- amortized cost $\le \frac{N}{3/4 N} = O(1)$.

Q: Worst-case O(1) append/pop possible? Yes =)

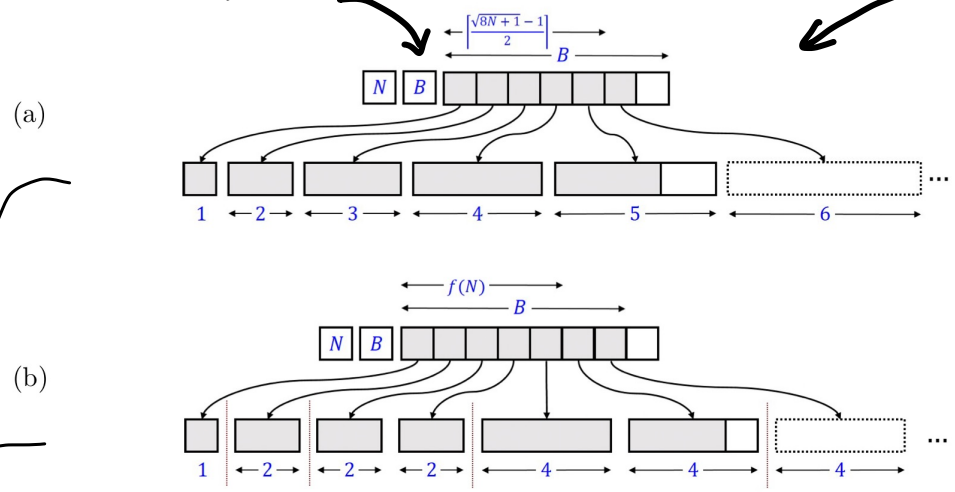
Brodnik et al. (1999)

[background rebuilding]

index

basic version:

invariant - $N \leq \frac{B(B+1)}{2}$



a[i] → Block index : computation requires √ & multiplies

↘ Data index - subtraction

→ a more sophisticated scheme: superblocks SB_i — size $2^{\lceil i/2 \rceil}$ — $2^{\lfloor i/2 \rfloor}$ blocks

- Bit ops are then possible.

i: 0, 1, 2, ...

How good are these in practice?

Activity 1: Characterize performance differences between geometric expansion/shrinking and Sitarski. Dimensions of interest: append latency (how bad is each costly operation), overall performance, access latency (How fast is a[I]?)

Activity 1.1: Let's try to time something small. Choose your fav language: C++, Rust.

```
for x in 0..=n {
    vv.push_back(x); // <— time this thing?
}
```

— very short duration for the most part

→ function calls: clock, time, high-prec time ← func call overhead can be too high.

→ rdtsc (read timestamp counter) ← count # of CPU cycles since reboot.

✓✓ try this and explain your finding.