index organized by
<age, sal>

B: # of data pages
R: # of records/page
D: avg read/write page
~ 15 ms
C: avg time to process a record
~ 100 ns
(e.g. to compare field)

Hash ⇒ H: time required to apply a hash function
~ 100 ns

Tree Index ⇒ F: fanout
~ 100

Assumptions:

Inserting

(1) Compact storage ⇒ insert in middle, move 1/2 -
Or ⇒ insert at end -
compact storage for heap

(2) When deleting: ~~no need to reclaim space~~
immediately ~~for~~. Can do it later by
simply filling in records in empty slots

(3) ~~Assume~~ (randomly)
for sorted files
expensive

(3) ~~Assume~~ For B+ tree
clustered: (1) assume page
1.5 B

(2) assume space
available in ~~page~~ leaf page
for insert.

(4) Unclustered Tree Index ⇒ Data entry = 1/10 record

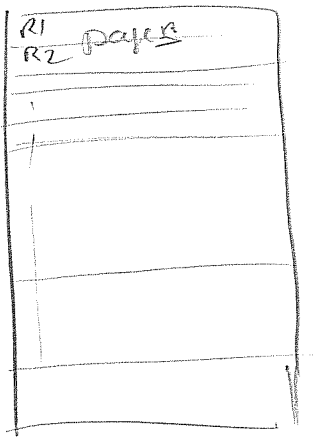
Heap

HEAP (1/2)

Scan

Fetch all records in a file

~~Assume each record in a separate page~~



Each page has R records

⇒ for every record: D + C

⇒ for every page: ~~R(D+C)~~ D + RC

⇒ for all pages: ~~BR(D+C)~~ B(D + RC)

Search with Equality

On average I have to search 1/2 the pages & search through all records in each to find a match.

⇒ for each record D+C

⇒ for each page: D+RC

⇒ for 1/2 pages: $\frac{1}{2}B(D+RC)$ Average

Worse on: B(D+RC)

Search with range

similar to scan whole file for a heap

$\Rightarrow B(D+RC)$

Insert

find the page : $\frac{1}{2} B(D+RC)$

modify RC

put the page back : D

* Assume record inserted at end of file

$\Rightarrow$ get page : D

insert record : C

$\Rightarrow$ place page back D

$2D+C$

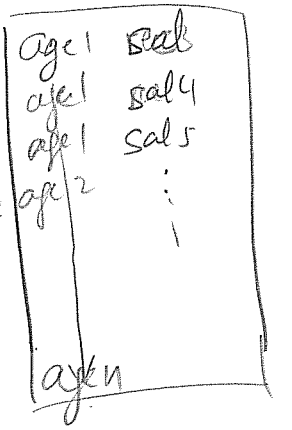
Delete

Find the page with the record $\Rightarrow \frac{1}{2} B(D+RC)$

delete record $\Rightarrow C$

put page back $\Rightarrow D$

SORTED FILE



Scan

one page : $D + RC$

all pages : $B(D+RC)$

Search with equality

Binary search on pages => ~~$\lceil \log_2 B \rceil (\quad)$~~

First get mid-page get <u>two</u> extreme records & compare to desired key

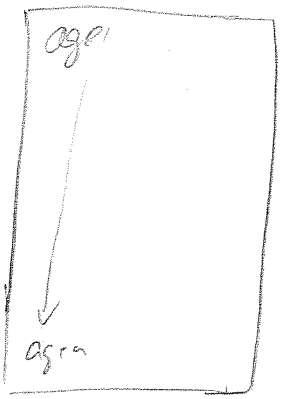
If within then found within page

$\Rightarrow \log_2 R$ to find record.

repeat $\boxed{(\log_2 B)(\cancel{D+2C}) + (\log_2 R)C}$

$\sim (\log_2 B)(D) + (\log_2 R)(C)$

Search with range



$\Rightarrow$ find 1st record

$(\log_2 B)(D+2C) + (\log_2 R)C$

+ sequential read

read pages with range

every time check if within range $\Rightarrow 2C$

until not within range ~~$\Rightarrow (\log_2 R)C$~~

$\frac{1}{2} RC$

Insert

$\Rightarrow$ first find the record ~~$\log_2 B(D+2C) + (\log_2 R)C$~~ $\Rightarrow$ Assume $\frac{1}{2}$ way in file

+ then

~~$(\log_2 B)(D+2C) + (\log_2 R)C$~~ $\Rightarrow$ read $\frac{1}{2} B(D+RC)$

+

~~move $\frac{1}{2}$ page~~ $\Rightarrow \frac{1}{2} RC$ ~~+C~~

now write back all other pages $\Rightarrow \frac{1}{2} B(D+RC)$

put page back

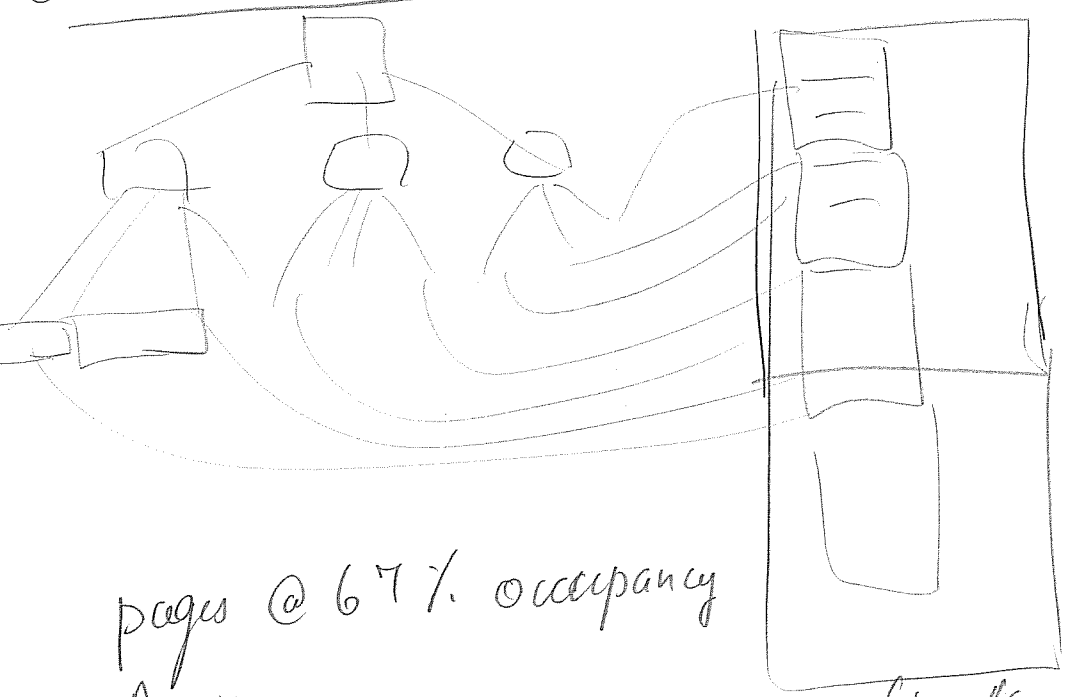
Total $\log_2 B(D+2C) + (\log_2 R)C + B(D+RC)$

Delete

find record : $(\log_2 B)(\frac{1}{2}D + 2C) + (\lg_2 R)C$

for compact
- + read $\frac{1}{2}B(D+RC)$
- + remove : $\frac{1}{2}C$
- + compact $\Rightarrow$ $\frac{1}{2}B(D+RC)$
- + ~~spark page back~~ : D

Clustered B+ Tree



$\frac{B}{0.67} = 1.5$ pages.

pages @ 67% occupancy

Assume
$\Rightarrow$ # of pages ~ 1.5B (to ~~keep room for tree~~ avoid growth

Scan

1.5B (D+RC)

search with equality

$\log_F(1.5B)$ to locate ~~record~~ page

~ height of tree

Each step, two comparisons for range

then binary search on page to determine

$\log_2(F \cdot R) \cdot C$

$2C$

One found binary search

$\log_F(1.5B)\,[D + 2C] + (\log_2(R))\,C$

Insert

$$\log_F(1.5B)\,[D+2C]$$

$$+$$

$$\log_2(R)\,C$$

$$+$$

write page back D

Find the page location to insert

Find the record location

Delete

same as above.

ap Fil

join out of
tree

index
on <age>

au =>

Assu

⇒

Assu

⇒

to fe

f

Now
data

Search with equality

search on age, given index.

need ① $\underbrace{\log_F (0.15B)}_{\text{\# of steps}} D$ to locate index page for desired age

↑ each time get a page of data entries.

then fetch actual data page

$$+ \underbrace{[D]}_{③} + \underbrace{\log_2 (6.7R) C}_{②}$$

② find the index within leaf.

Binary search on age within index page

$$\text{total} = \left[\log_F (0.15B)\right] D + \left[D + C\log_2 (6.7R)\right]$$

search with range

⇒ search with equality then use linked index list to get remaining page for each record.

Insert

in the data file (at end) $= 2D + C$ ①

update index file: locate position $+$

$[D \log_F (0.15B) + C \log_2 (6.7R)] +$ ④

Delete

locate index: $D \log_F (0.15B)$ ①

$+ C \log_2 (0.7R)$ ②

+ get page D ③

+ delete C ④

+ update data file D ⑤

+ update data index D ⑥ (leave index location empty)

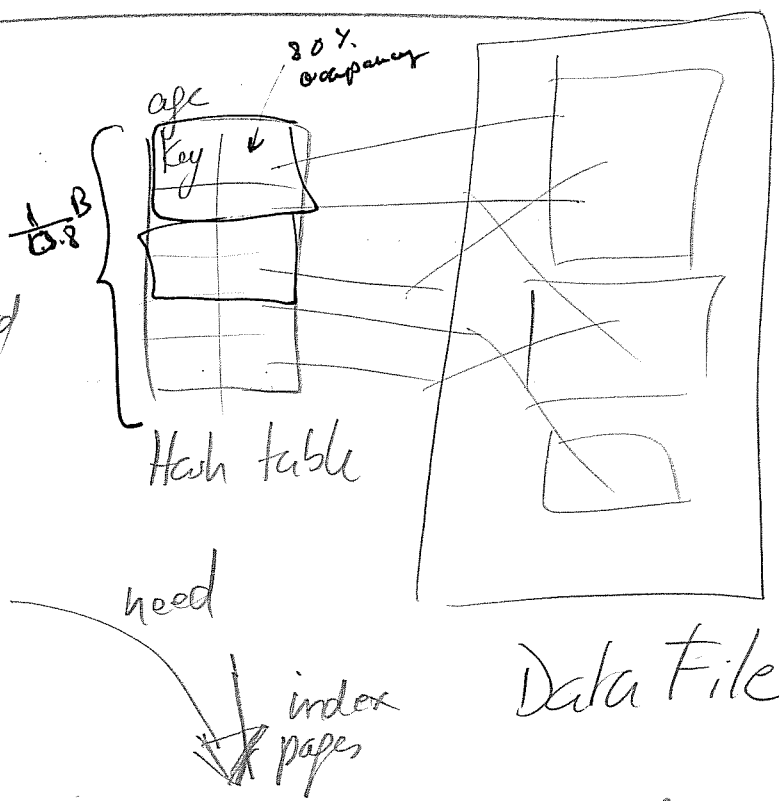
---

total = sum

Heap File with Unclustered Hash Index

Assume static hashing

data entry = 0.1 record data



pages at 80% capacity

need index pages

scan

Hash Table = $0.1 \cancel{B}\,(1.25B) = 0.125B$

80% occupancy

$\Rightarrow$ Reading Index File = $0.125B\left[D + \frac{10}{0.8}RC\right]$

For data file: +

For each record, get page:

$$RB\left[D + C\right]$$

---

total = sum

Search with Equality

find index page : H

get index page + : D

find a record in index + $\frac{1}{2}[R8] = 4RC$

get data page + : D

find index + : $\frac{1}{2}RC$

---

total $2D + \frac{3}{2}RC + H$

search with range

⇒ Scan entire data file

$B(D + RC)$

Insert

Insert in heap file (at end) $= 2D + C$

Insert hash entry : $H + [2D + C]$ just place it assume enough room

---

total = sum

Comparing

~~Sorted~~ Heap => no overhead & fast scanning
& insertions, but slow for search / delete

Sorted file => insertion & deletion slow (moving things)

Clustered file => inserts / deletes ($\log_F$ vs $\log_2$). (better than sorted ~~overhead~~) No need to move on Insert / delete

~~Unclustered~~ Tree file & Unclustered Tree & hash => fast search