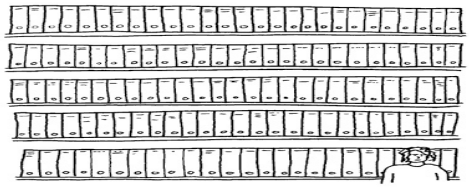
Databasteknik
Databaser och bioinformatik
Data structures and Indexing (I)

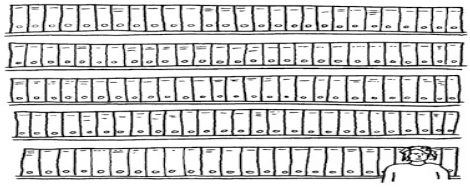
Fang Wei-Kleiner

Searching and Indexing



- There are 8 Million archives.
- Each archive consists of the personal information such as person number, name, address, age, …
- The task is given a person number, find the corresponding archive.

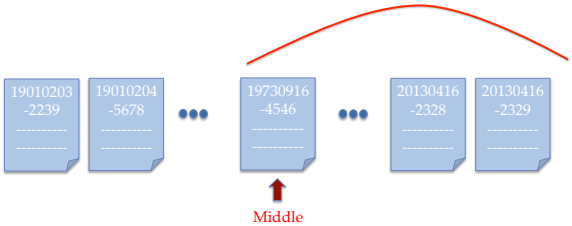
Searching and Indexing



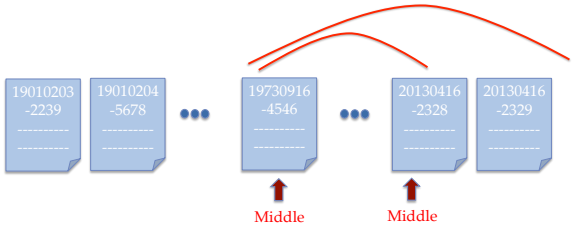
- Constraint: the archives are stored in some place, you can go and fetch the archive (each time only one piece), but you can only read the information after you are back in your place.
- How much time do we need to find the archive given a person number *n*?

Sort the archives

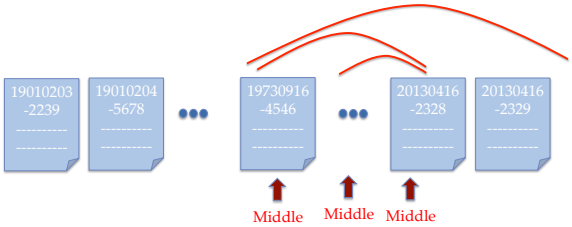
- Sort the archives according to the person number
- What is the time cost for finding the archive given a person number n?
- → Binary search!



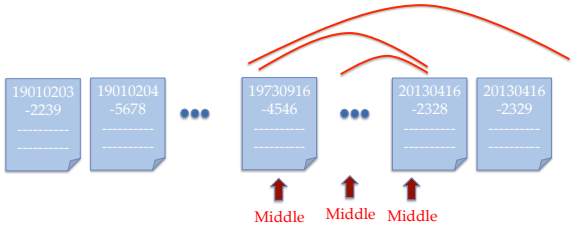
- Sort the archives according to the person number
- What is the time cost for finding the archive given a person number n?
- → Binary search!



- Sort the archives according to the person number
- What is the time cost for finding the archive given a person number n?
- → Binary search!

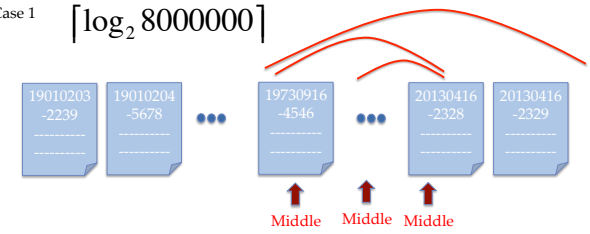


- Sort the archives according to the person number
- What is the time cost for finding the archive given a person number n?
- → Binary search!



- Sort the archives according to the person number
- What is the time cost for finding the archive given a person number n?
- → Binary search!
- Each time half of the documents are removed…

Case 1 $\lceil \log_2 8000000 \rceil$



- Sort the archives according to the person number
- What is the time cost for finding the archive given a person number n?
- → Binary search!
- Each time half of the documents are removed…

- Now consider that we can put every 100 sorted archives into one box. → order remains!
- The number of boxes: 8000000/100=80000
- Time cost for retrieving the document with a given person number?

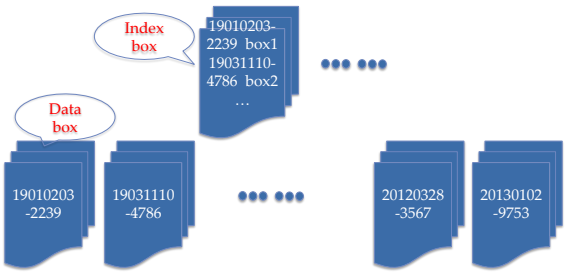
Case 2 $\lceil \log_2 80000 \rceil$

- Now consider that we can put every 100 sorted archives into one box. → order remains!
- The number of boxes: 8000000/100=80000
- Time cost for retrieving the document with a given person number?

How about building Index for the boxes?

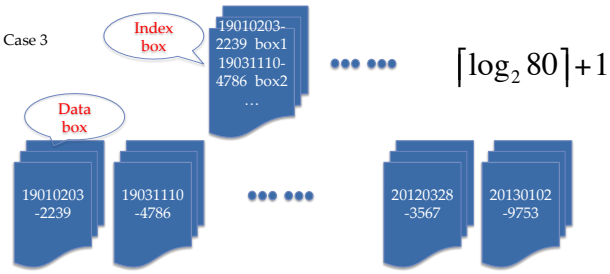
- The number of boxes: 80000
- Index entry contains: the smallest person number of the box, box number

- The number of boxes: 80000
- Index entry contains: the smallest person number of the box, box number (i.e. the address of the box)
- Note that the person number entries are sorted.
- We store the index entries in boxes!
- Assume we can store 1000 index entries in each box.
- We need 80000/1000=80 boxes to store indexes.



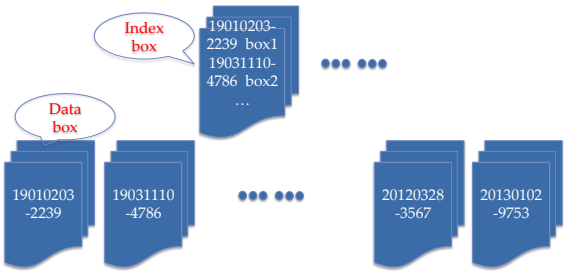
- So we have 80000 data boxes + 80 index boxes.
- Searching?
- We search first the index boxes. Binary search, since they are ordered.

Case 3

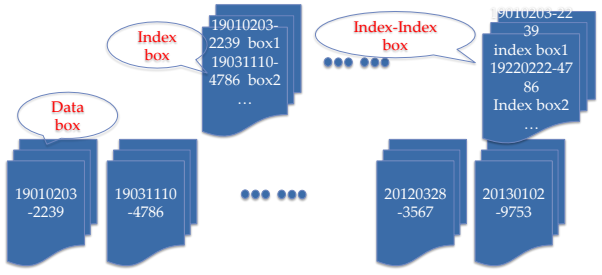


$$\lceil \log_2 80 \rceil + 1$$

- So we have 80000 data boxes + 80 index boxes.
- Searching?
- We search first the index boxes. Binary search, since they are ordered.
- +1 means as soon as we find the entry in the index, we need to fetch the document.

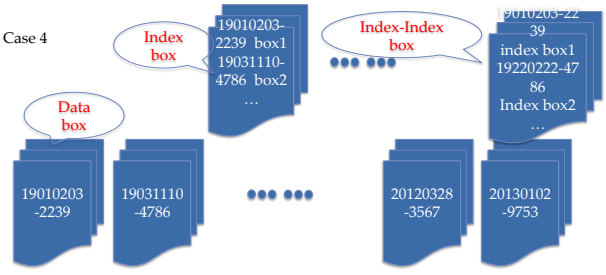


- So we have 80000 data boxes + 80 index boxes.
- We can build index over index (second level index).
- Each index box → one entry
- So we need 80 index entries → that fits in one box.



- So we have 80000 data boxes + 80 index boxes.
- We can build index over index (second level index).
- Each index box → one entry
- So we need 80 index entries → that fits in one box.
- So we have 80000 data boxes + 80 index boxes + 1 index-index box.

Case 4

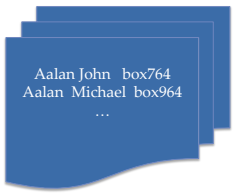


- So we have 80000 data boxes + 80 index boxes.
- We can build index over index (second level index).
- Each index box → one entry
- So we need 80 index entries → that fits in one box.
- So we have 80000 data boxes + 80 index boxes + 1 index-index box.
- Search? 1+1+1

- The number of boxes: 80000.
- Now we search for a document given only the name value → no person number.
- Time cost??
- We have to go through all the boxes → worst case 80000, average 80000/2=40000.

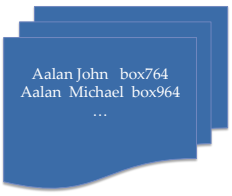
How about building index for the names?

- The documents are not sorted after name → we need for each document one entry in the index → 8000000!
- Index entry contains: name, box number
- But we sort the names in the index of course!



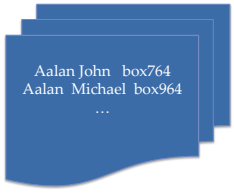
- Assume again we can store 1000 index entries in each box.
- We need 8000000/1000=8000 boxes to store indexes for the names → 100 times more than the person number.
- So now we have 80000 data boxes + 8000 index boxes for names.
- Search according to name?

Case 5



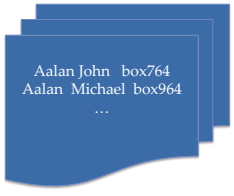
$$\lceil \log_2 8000 \rceil + 1$$

- Assume again we can store 1000 index entries in each box.
- We need 8000000/1000=8000 boxes to store indexes for the names → 100 times more than the person number.
- So now we have 80000 data boxes + 8000 index boxes for names.
- Search according to name?



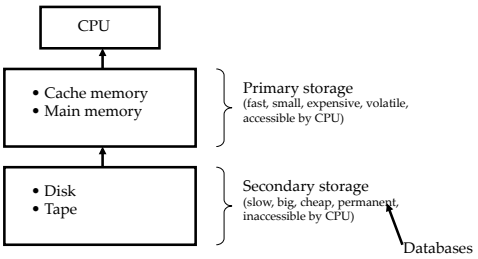
- So now we have 80000 data boxes + 8000 index boxes for names.
- We can build a second level index → names are sorted in the indexes, so each index gets one entry in the second level index.
- Since each box fits 1000 entries, we need 8000/1000=8 second level index boxes →
- 80000 data boxes + 8000 index boxes + 8 second level index boxes.

Case 6



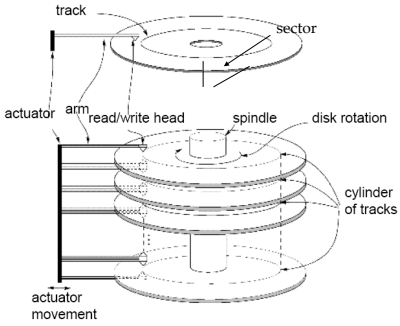
- Since each box fits 1000 entries, we need 8000/1000=8 second level index boxes →
- 80000 data boxes + 8000 index boxes + 8 second level index boxes.
- Third level index → one more box.
- So searching on the 3 level indexes takes 3+1 times of fetching the boxes (3 times index box and 1 time document).

Storage hierarchy



- Important because it effects query efficiency.

Disk



Disk

- Formatting divides the hard-coded sectors into equal-sized **blocks**.
- Block is the **unit of transfer of data** between disk and main memory, e.g.
  - Read = copy block from disk to buffer in main memory.
  - Write = the opposite way.
  - R/w time = seek time + rotational delay + block transfer time.

search track + search block + 1-2 msec. = 12-60 msec.

Inside a harddisk

Files and records

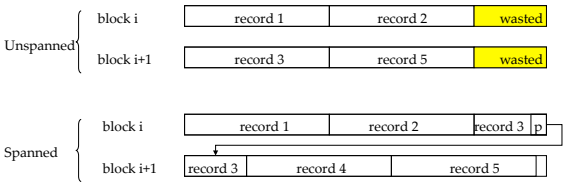
- Data stored in **files**.
- File is a **sequence** of **records**.
- Record is a set of field values.
- For instance, file = relation, record = entity, and field = attribute.
- Records are allocated to file **blocks**.

Files and records

- Let us assume
  - B is the size in bytes of the block.
  - R is the size in bytes of the record.
  - r is the number of records in the file.
- **Blocking factor**: $bfr = \left\lfloor \frac{B}{R} \right\rfloor$
- Blocks needed: $b = \left\lceil \frac{r}{bfr} \right\rceil$
- What is the space wasted per block ?

Files and records

- Wasted space per block = $B - bfr * R$.
- Solution: **Spanned** records.



From file blocks to disk blocks

- **Contiguous** allocation: cheap sequential access but expensive record addition. Why ?
- **Linked** allocation: expensive sequential access but cheap record addition. Why ?
- **Linked clusters** allocation.
- **Indexed** allocation.

File organization

- Heap files.
- Sorted files.
- Hash files.

- File organization != access method, though related in terms of efficiency.