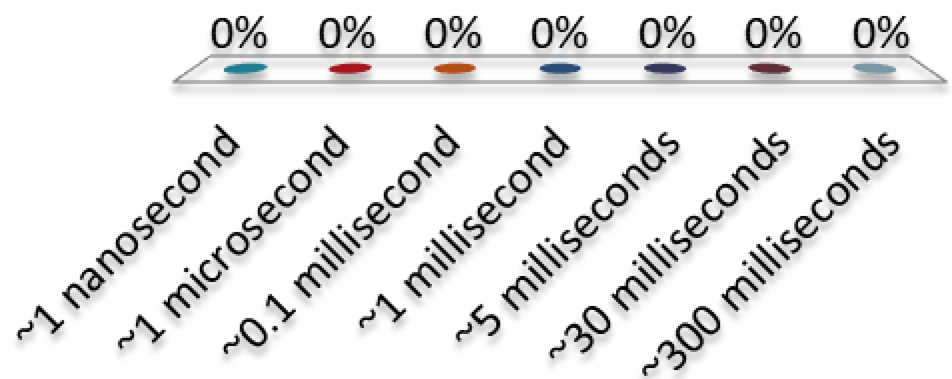
Sequential File Organization

- Keep sorted by some search key
- Insertion
  - Find the block in which the tuple should be
  - If there is free space, insert it
  - Otherwise, must create *overflow pages*
- Deletions
  - Delete and keep the free space
  - Databases tend to be insert heavy, so free space gets used fast
- Can become *fragmented*
  - Must reorganize once in a while

Find a record with ID = x in sequential file with 1,000,000,000 records, each record is 100 bytes?

A. ~1 nanosecond
B. ~1 microsecond
C. ~0.1 millisecond
D. ~1 millisecond
E. ~5 milliseconds
F. ~30 milliseconds
G. ~300 milliseconds



Sequential File Organization

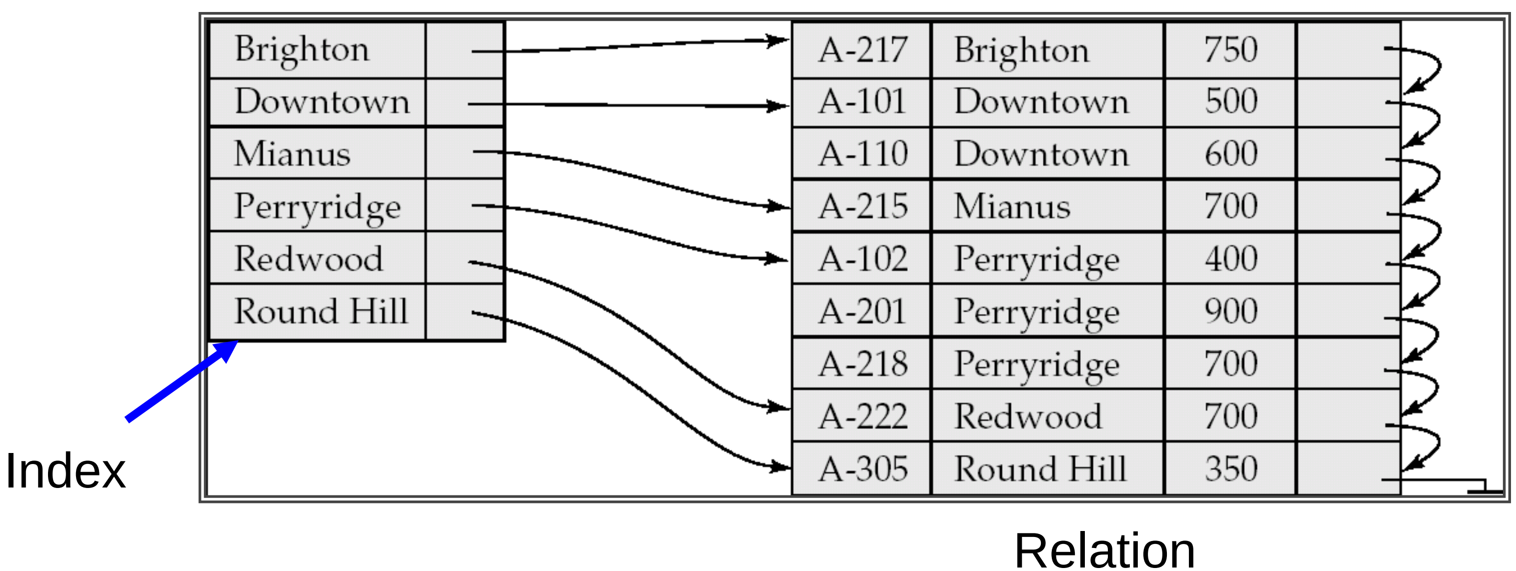
- What if I want to find a particular record by value ?
  - *Account info for ID = 123*
- Binary search
  - Takes *log(n)* number of disk accesses
    - Random accesses
  - Too much
    - n = 1,000,000,000 -- log(n) = 30
    - Recall each random access approx 10 ms
    - 300 ms to find just one account information
    - < 4 requests satisfied per second

Index

- A data structure for efficient search through large databaess
- Two key ideas:
  - The records are mapped to the disk blocks in specific ways
    - Sorted, or hash-based
  - Auxiliary data structures are maintained that allow quick search
- Think library index/catalogue
- Search key:
  - Attribute or set of attributes used to look up records
  - E.g. ID for a person table
- Two types of indexes
  - *Ordered* indexes
  - *Hash-based* indexes

Ordered Indexes

- Primary index
  - The relation is sorted on the search key of the index
- Secondary index
  - It is not
- Can have only one primary index on a relation



Primary *Sparse* Index

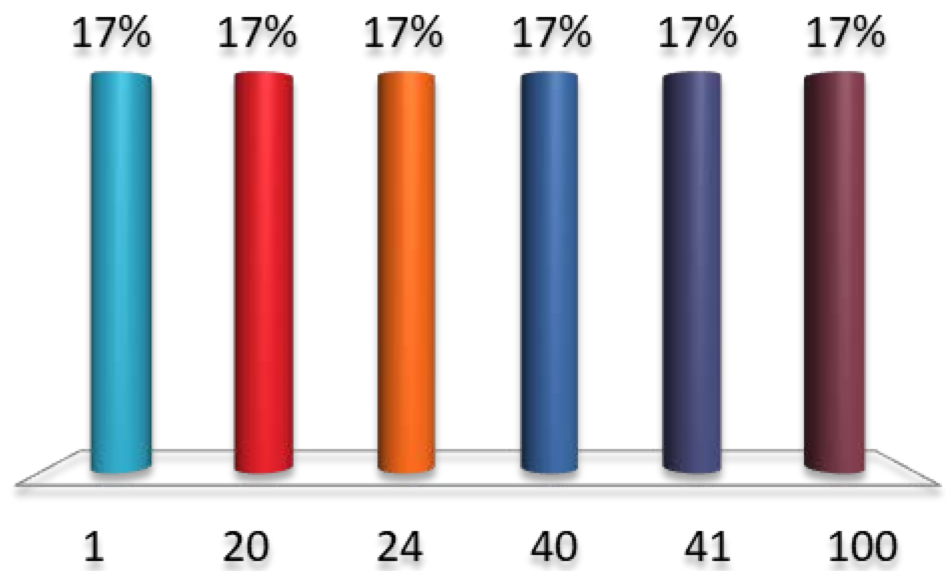
- Every key doesn’t have to appear in the index
- Allows for very small indexes
  - Tradeoff: Must access the relation file even if the record is not present
  - Tradeoff: Must scan through multiple records to find desired record
    - But at least this scan is sequential

Brighton	
Mianus	
Redwood	

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

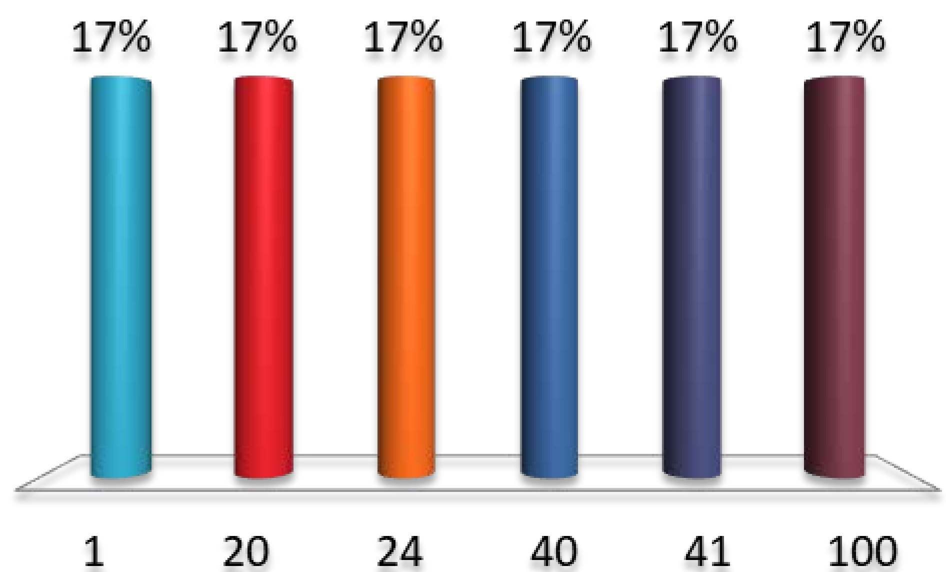
How many index keys / records will be accessed to find X which happens to be the 100^{th} record of a table, when a sparse index contains an entry for every 40 records (i.e. it has pointers to the 1^{st}, 41^{st}, 81^{st}, 121^{st}, etc. records)

A. 1
B. 20
C. 24
D. 40
E. 41
F. 100



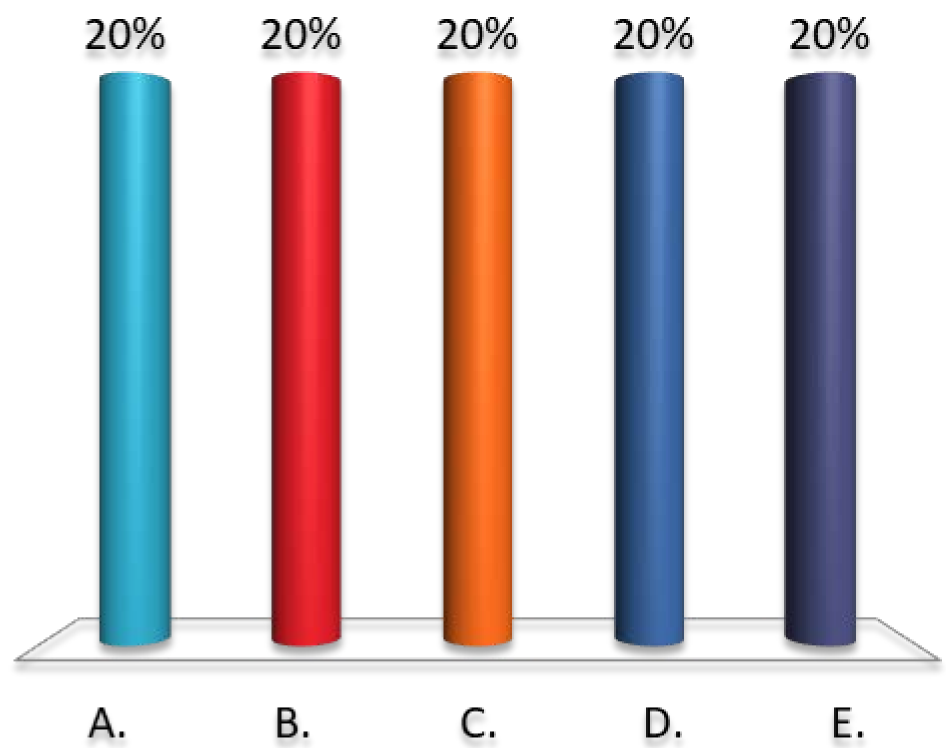
How many index keys / records will be accessed to find X which happens to be the 100th record of a table, when a dense index is used instead of a sparse index?

A. 1
B. 20
C. 24
D. 40
E. 41
F. 100



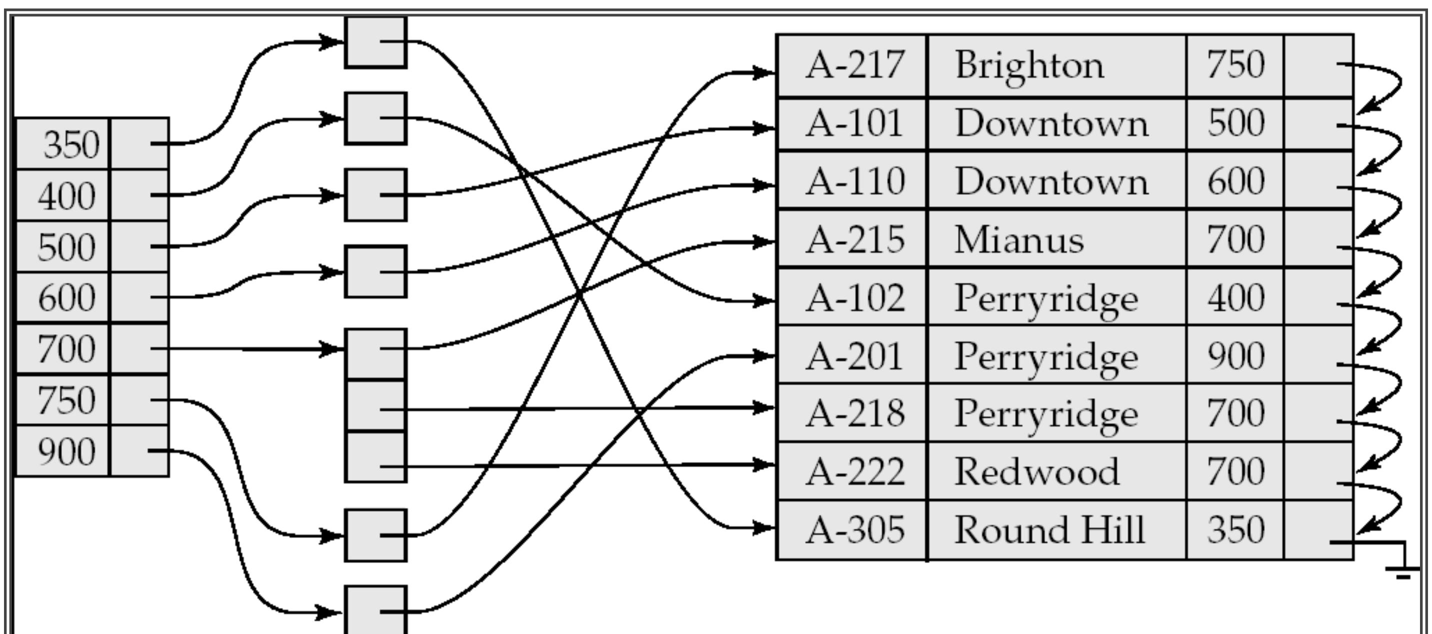
When should a dense index be used instead of a sparse index (assume data is sorted by indexed attribute)

A. Never
B. When we want to find one of the first few tuples in a table
C. When we want to find one of the last few tuples in a table
D. When the entire dense index fits in memory
E. When random IO is just as slow as sequential IO



Secondary Index

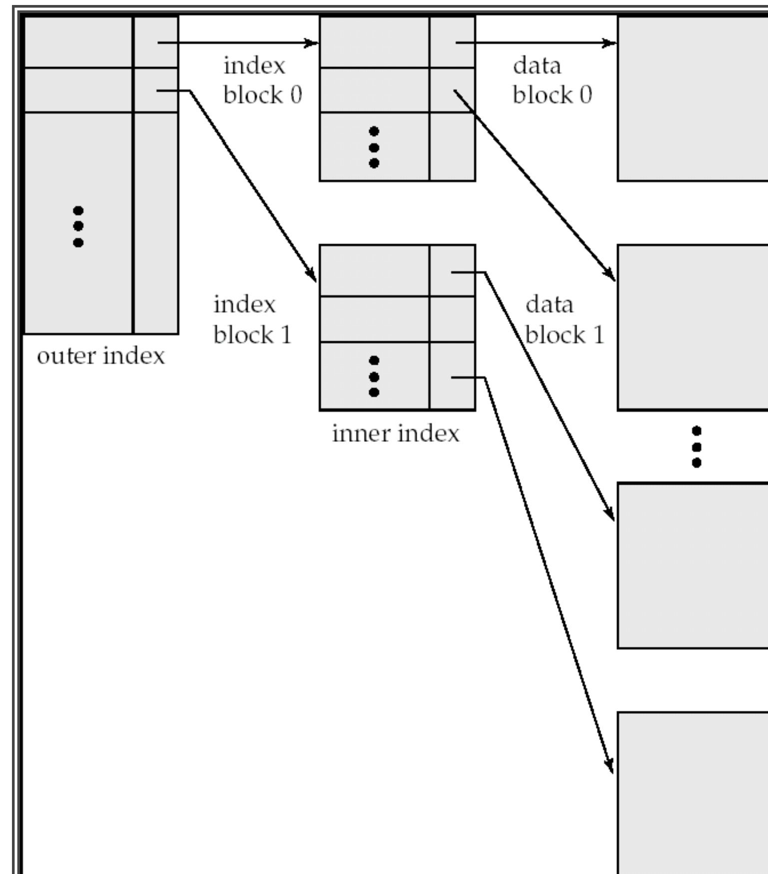
- Relation sorted on *branch*
- But we want an index on *balance*
- Must be dense
  - Every search key must appear in the index



Multi-level Indexes

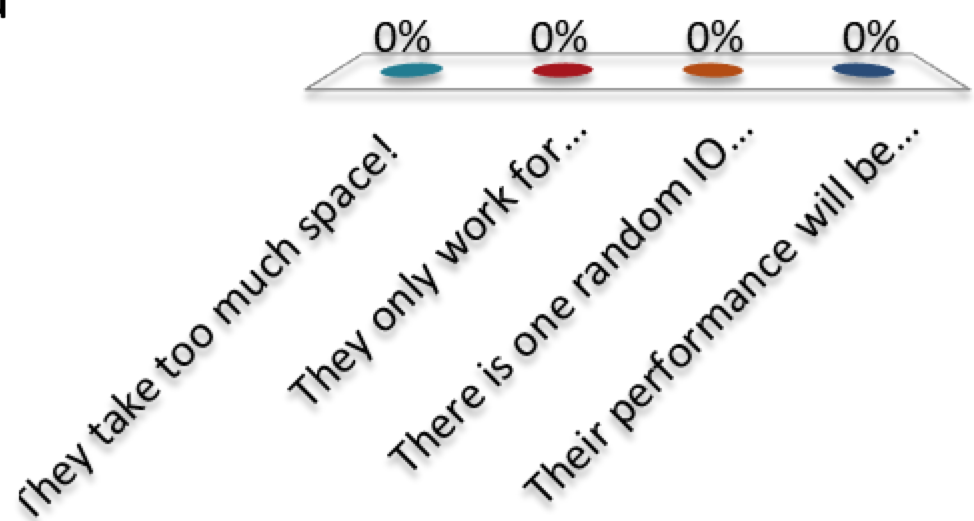
- What if the index itself is too big for memory ?
- Relation size = n = 1,000,000,000
- Block size = 100 tuples per block
- So, number of pages = 10,000,000

- Solution
  - Build a sparse outer index on the index itself



What do you think is the biggest downside of multi-level indexes from the previous slide?

A. They take too much space!
B. They only work for primary indexes (when data is sorted by the indexed attribute)!
C. There is one random IO per level in the hierarchy!
D. Their performance will be bad in the face of many inserts!



Multi-level Indexes

- What about keeping the index up-to-date ?
  - Tuple insertions and deletions
    - This is a static structure
    - Need overflow pages to deal with insertions
  - Works well if no inserts/deletes
  - Not so good when inserts and deletes are common
- This is the main motivation for B-trees