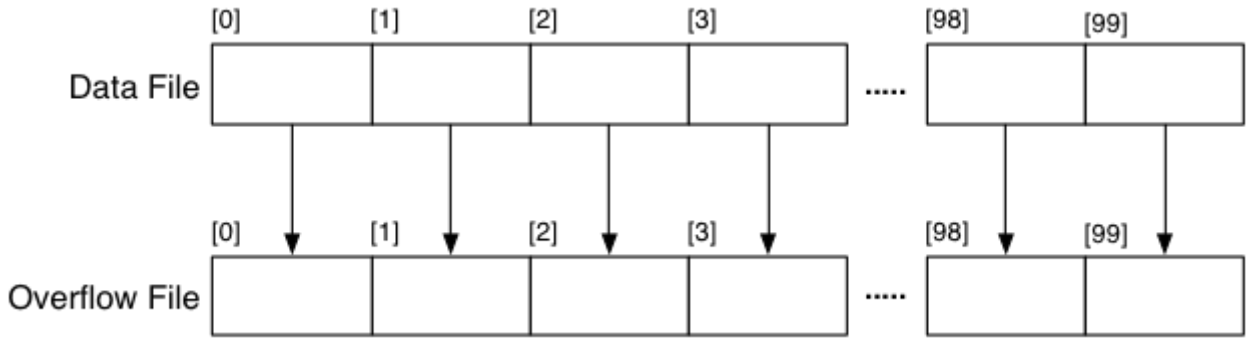
Quiz 3

Deadline	Friday, 27 March 2020 at 11:59PM
Latest Submission	Thursday, 26 March 2020 at 8:59PM
Raw Mark	4.00/4.00 (100.00%)
Late Penalty	N/A
Final Mark	4.00/4.00 (100.00%)

Question 1 (1 mark)

Consider a **sorted** file that uses overflow pages. The "file" actually consists of two files: a main data file and an overflow file. Each page in the main data file is potentially associated with a chain of overflow pages in the overflow file; a data page and its overflow page(s) form a data "bucket". The following diagram shows an example of such a file, which has *b = 100* data pages and, by an amazing coincidence, exactly one overflow page for each data page.



When a tuple is inserted into the file, we first use binary search to locate the appropriate bucket into which the tuple should be placed. During the search, we need to examine both the data page and its associated overflow pages to determine whether the new tuple should be placed in that bucket.

The binary search algorithm used is roughly as follows:

```
lo = 0; hi = b;
while (1) {
    mid = (lo+hi)/2);
    (min, max) = findMinMaxKeys(mid);
    if (key < min)      hi = mid-1;
    else if (key > max) lo = mid+1;
    else /* this bucket */      break;
}
```

where findMinMax`Keys()` determines the minimum and maximum key values in a bucket by reading all of the data in the bucket.

If there is room in the main data page, we can place the tuple into it; otherwise the tuple is placed in an overflow page. If all of the overflow pages associated with the data page are full, then we add a new overflow page and link it to the overflow chain.

If the file is in the state above, what is the maximum number of pages read and written to insert a new tuple?

(a) ○	12 reads + 1 write
(b) ○	12 reads + 2 writes
(c) ○	14 reads + 1 write
(d) ◉	14 reads + 2 writes
(e) ○	16 reads + 1 write
(f) ○	16 reads + 2 writes
(g) ○	None of the other options is correct

✔ Your response was correct.
Mark: 1.00

The binary search takes at most ceil($log_2 b$) pages, where b=100 in this case => reads at most 7 data pages. For each data page, we also read 1 overflow page. Maximum number of page reads is 14.

We write either 1 or 2 pages. 1 page if there is room in the data page or overflow page for the new tuple, i.e. insert the tuple and writ the page. 2 pages if there is no room in the data page or overflow page i.e. create a new (empty) overflow page, insert the tuples and write the new page, then update what was previously the last overflow page to point to the new overflow page, and write that page.

Question 2 (1 mark)

Given the following table definition:

```
create table Students (
    id     integer check (id between 300000 and 400000),
    name   varchar(100) not null,
    phone  varchar(20) unique,
    born   date,
    degree varchar(10),
    primary key (id)
);
```

Assume that id values are in the range 300000 to 400000 inclusive, multiple students have the same birthday, multiple students have the same name, and many people study for a given degree.

Which of the following is an example of a *one*-type query? (where a *one* type query is a query that is guaranteed to return at most one result in a given database instance)

(a) ○	select * from Students where born between '1990-01-01' and '1999-12-31';

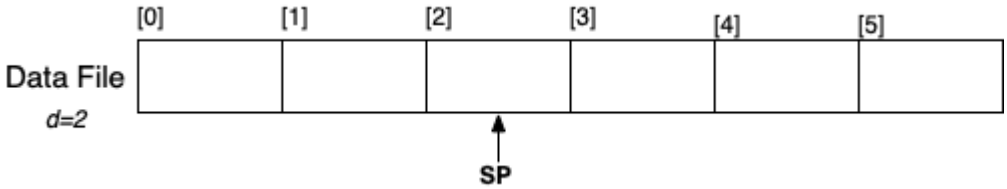
(b) ○	select * from Students where born = '1990-01-01';
(c) ○	select * from Students where phone like '9385%';
(d) ○	select * from Students where degree = 'PhD';
(e) ◉	select * from Students where id > 399999;
(f) ○	None of the other options is correct

✔ Your response was correct.
Mark: 1.00

Slightly trick question. The constraint enforces that the maximum id value is 400000, and the id attribute is unique (primary key). The query asking whether the id > 3999999 can have at most one possible answer, which is the definition of a *one* query. All of the other queries can potentially have many answers.

Question 3 (1 mark)

Consider the following linear-hashed file with depth *d=2* and the split pointer at page 2. Assume that it has no overflow pages.



If a query with hash value `...01110101` is run on this table, which page(s) will need to be read to answer the query?

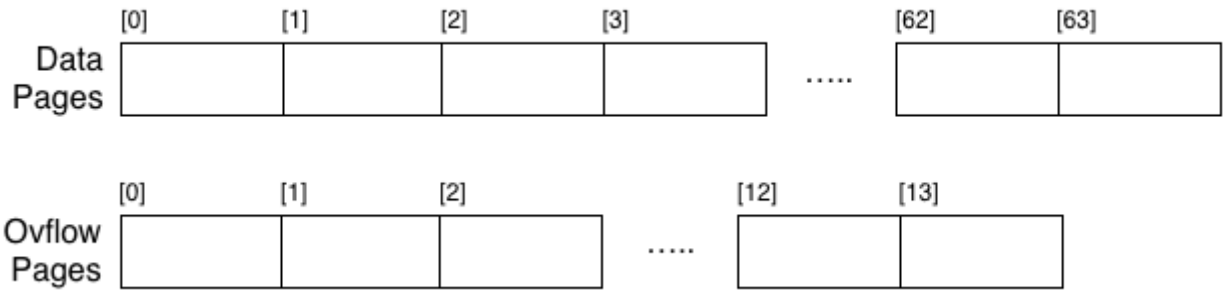
(a) ○	page 1 only
(b) ○	page 2 only
(c) ◉	page 5 only
(d) ○	pages 1 and 2
(e) ○	pages 1 and 5
(f) ○	None of the other options is correct

✔ Your response was correct.
Mark: 1.00

Since the last 2 bits of the hash value are `....01` , and this gives a page address which is less than *sp* , we need to consider 3 bits of the hash value, giving a hash of `...101` , which is page 5. So we read page 5 only.

Question 4 (1 mark)

Consider the following statically-hashed file, with b = 64 data pages and b_{Ov} = 14 overflow pages



The file stores data for a relation *R(id,a,b,c)*. The table is hashed on the *id* field, using 6-bits of the hash value, where *id* values are in the range 1000..2000.

How many data and overflow pages would be read if we asked the following query:

```
select * from R where id > 1234;
```

(a) ○	1 data page and 0 or 1 overflow pages
(b) ○	1 data page and between 0 and 14 overflow pages
(c) ○	6 data pages and 0 or more overflow pages
(d) ○	64 data pages and 0 overflow pages
(e) ◉	64 data pages and 14 overflow pages
(f) ○	None of the other options is correct

✔ Your response was correct.
Mark: 1.00

Hashing gives no help for queries not involving equality on the hash attribute. This is the case here, so we need to consider all pages (64+14).

Note also that hashing places the tuples effectively in random order on the key value, so we *can't* use the trick of finding the lower bound and scanning.