Task 1 (5 marks)
An objective of this task is to estimate the efficiency of indexing.

Assume that a relational table ORDERS contains information about the orders submitted by the customers.

```
ORDERS(order#, order_date, product, quantity, price_per_unit)
```

A relational table ORDERS has a primary key (order#).

Assume that:
(i) a relational table ORDERS occupies 1000 data blocks,
(ii) a blocking factor in a relational table ORDERS is 50 rows per block,
(iii) a relational table ORDERS contains information about 200 products,
(iv) a relational table ORDERS contains information about 100 prices per unit,
(v) a primary key is automatically indexed,
(vi) an attribute product is indexed,
(vii) all indexes are implemented as B*-trees with a fanout equal to 10,
(viii) a leaf level of an index on an attribute product consists of 30 data blocks,
(ix) a leaf level of an index on primary key consists of 200 data blocks,

For each one of the following queries briefly describe how the database system processes each query and estimate the total number of read block operations needed to compute each query. There is no need to perform the final computations. A correctly constructed formula filled with the appropriate constants is completely sufficient.

(1) 1 mark

```
SELECT product
FROM ORDERS
WHERE product = 'bolt' OR quantity = 100;
```

A relational table ORDERS is sequentially read block by block. A condition quantity = 100 is evaluated for each row in each data block read from a relational table.

1000

(2) 1 mark

```
SELECT count(*)
FROM ORDERS
WHERE product IN ('bolt', 'screw');
```

An index on an attribute product is vertically traversed twice and the sets of row identifiers associated with each key in ('bolt', 'screw') are union. Next, a set of row identifiers obtained after union is counted.

$2 * (\log_{10}(200)+1)$

(3) 1 mark

```
SELECT product, COUNT(*)
FROM ORDERS
GROUP BY product
HAVING count(*) > 5;
```

An index on an attribute `product` is horizontally traversed and the row identifiers associated with each index key are counted. If a result of counting is less or equal `5` then a key and a counter associated with it is ignored.

`30`

(4) 1 mark

```
SELECT order#, product, quality
FROM ORDERS
ORDER BY order#, product;
```

A relational table is sorted over `order#, product` and after sorting the values of `order#, product, quality` are listed. Persistent storage sort can be used for sorting. Persistent storage sort reads a table one time to partition it into a number of buckets. Each time a bucket is read into a transient memory it is sorted there and later on written to persistent storage. Then the buckets are simultaneously read from persistent storage and merged into the final results. The results can be communicated to a user. Such algorithm reads a table 2 times and it writes a table 1 time.

`read: 2*1000`
`write: 1000`

(5) 1 mark

```
SELECT *
FROM ORDERS
WHERE order# = 12345 AND product = 'bolt';
```

An index on the primary key `order#` is traversed vertically. When a key is found, one block from a relational table is read and a condition `product = 'bolt'` is evaluated in transient memory.

$(\log_{10}(50*1000)+1) +1$

(6) 1 mark

```
SELECT product
FROM ORDERS;
WHERE product = 'bolt' AND quantity = 100;
```

An index on an attribute `product` is vertically traversed and the sets of row identifiers are used to get the rows from a relational table. A condition `quantity = 100` is evaluated on each row in a transient memory.

$(\log_{10}(200)+1) + ((50*1000/200)+ (1000/200))/2$

(7) 1 mark

```
SELECT product
FROM ORDERS;
```

An index on an attribute `product` is horizontally traversed.

30

(8) 1 mark

```
SELECT order#, product, quality
FROM ORDERS;
```

A relational table `ORDERS` is sequentially read block by block.

1000

(9) 1 mark

```
SELECT order#, product
FROM ORDERS;
```

A relational table `ORDERS` is sequentially read block by block.

1000

(10) 1 mark

```
SELECT order#
FROM ORDERS;
```

An index on the primary key `order#` is horizontally traversed.

200

Deliverables

A file `solution1.pdf` with the comprehensive descriptions of query processing plans for each query and the estimations of the total number of read block operations needed to process each query.
