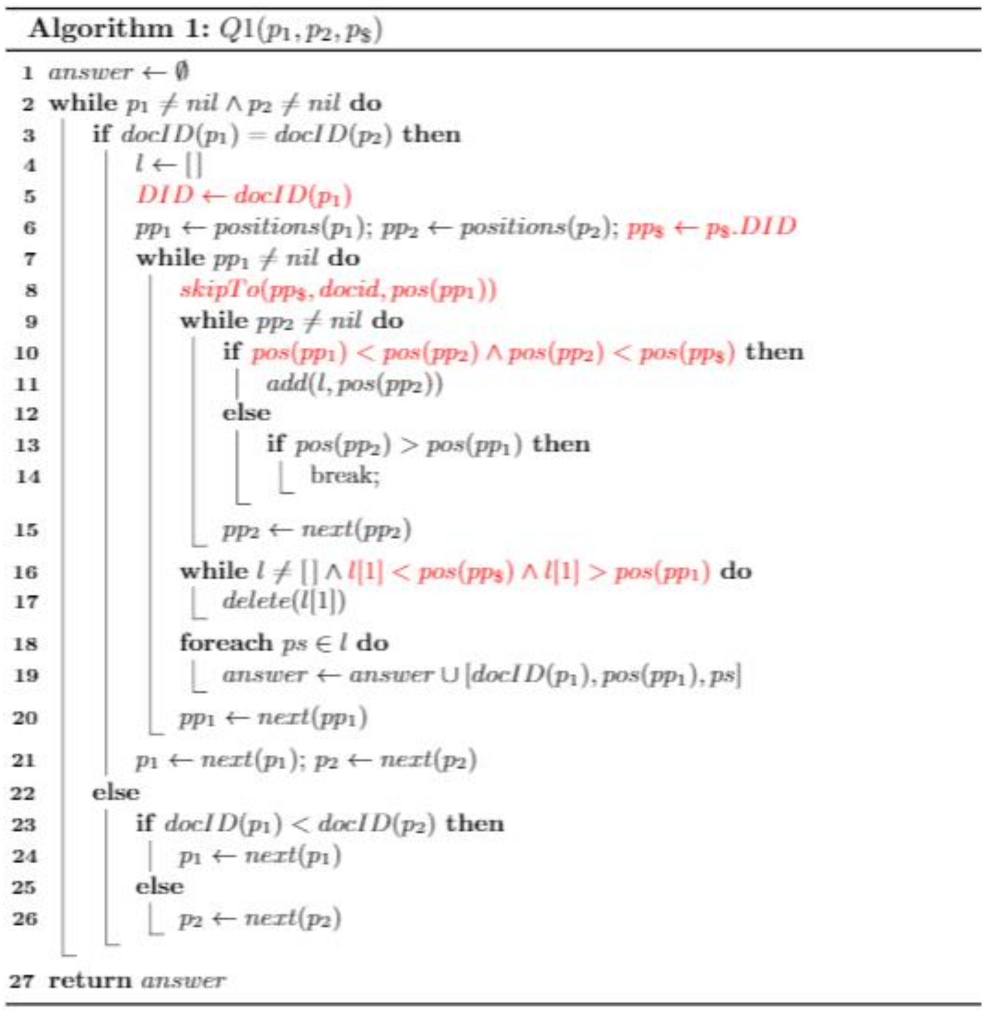
COMP6714 ASSIGNMENT 1

MINGLANG XIE z5228006

Q1



Algorithm 1: $Q1(p_1, p_2, p_\$)$

```
answer ← ∅
while $p_1 \neq nil \wedge p_2 \neq nil$ do
    if $docID(p_1) = docID(p_2)$ then
        $l \leftarrow []$
        $DID \leftarrow docID(p_1)$
        $pp_1 \leftarrow positions(p_1)$; $pp_2 \leftarrow positions(p_2)$; $pp_\$ \leftarrow p_\$.DID$
        while $pp_1 \neq nil$ do
            $skipTo(pp_\$, docid, pos(pp_1))$
            while $pp_2 \neq nil$ do
                if $pos(pp_1) < pos(pp_2) \wedge pos(pp_2) < pos(pp_\$)$ then
                    $add(l, pos(pp_2))$
                else
                    if $pos(pp_2) > pos(pp_1)$ then
                        break;
                $pp_2 \leftarrow next(pp_2)$
            while $l \neq [] \wedge l[1] < pos(pp_\$) \wedge l[1] > pos(pp_1)$ do
                $delete(l[1])$
            foreach $ps \in l$ do
                $answer \leftarrow answer \cup [docID(p_1), pos(pp_1), ps]$
            $pp_1 \leftarrow next(pp_1)$
        $p_1 \leftarrow next(p_1)$; $p_2 \leftarrow next(p_2)$
    else
        if $docID(p_1) < docID(p_2)$ then
            $p_1 \leftarrow next(p_1)$
        else
            $p_2 \leftarrow next(p_2)$
return answer
```

Q2

(1)

Assume that t sub-indexes (each of M pages) will be created if one chooses the no-merge strategy. Thus, there are tM pages in total, if the logarithmic merge strategy is used, we are following two rules:

1. only create I_k and I_k has size $2^k M$
2. whenever we have 2 I_k, we need to merge it to form a I_{k+1}

Therefore, when the generation grow to g, we will get $[M, 2M, 4M, \dots, 2^k M]$

$$M, 2M, 4M, \dots, 2^k M = tM$$

$$M(1+2+4+\cdots+2^k)=tM$$
$$\frac{(2^{k+1})-1}{2-1}=t$$
$$(2^{k+1})=t+1$$
$$k=\log_2(t+1)-1$$
when t is sufficient large, $k=\lceil\log_2(t)\rceil$. Therefore, if the logarithmic merge strategy is used, it will result in at most $\lceil\log_2(t)\rceil$ sub-indexes.

(2)
First, there are tM pages in total.
Second, we know the logarithmic merge strategy will result in at most $k=\lceil\log_2(t)\rceil$, which means after k generations, only the last generation in disk. Simply progress is:

Merge I_0 2^{k-1} times
Merge I_1 2^{k-2} times
…
Merge I_k one time

Hence, the total cost of the logarithmic merge is: $(2^0*2^{k-1}+2^1*2^{k-2}+\cdots+2^{k-1}*2^0)M=k*2^{k-1}*M\approx\log_2 t*t*M$
The total I/O cost of the logarithmic merge is $O(\log_2 t*t*M)$

Q3

After the δ encoding, the compressed non-positional inverted list is
$$01000101\ 11110001\ 01110000\ 00110000\ 11110110\ 11011$$
Since this compressed list started with a 0, and 1 is the only number to get a 0 after the δ encoding. So, we can divide the compressed list as:
$$0\ 1000101\ 11110001\ 01110000\ 00110000\ 11110110\ 11011$$
Then, we get the first document ID which is 1, and remian:
$$1000101\ 11110001\ 01110000\ 00110000\ 11110110\ 11011$$
The next gap will be $\mathrm{decode}(1000)$, which is 2, because the remain compressed list started with 10, and so $k_{dd}=1$, and 0 after 10, which give $k_{dr}=0$, use $k_{dr}=(k_d+1)-2^{\lfloor\log_2(k_d+1)\rfloor}$, we have $0=(k_d+1)-2^1$, given $k_d=1$, next, 0 after 100 given $k_r=0$. Therefore, $k_r=k-2^{k_d}\Rightarrow 0=k-2\Rightarrow k=2$. Then remain:
$$101\ 11110001\ 01110000\ 00110000\ 11110110\ 11011$$

Continue this progress, we can divide the compressed list as below:
$$(10\ 1\ 11)\ (110\ 00\ 101)\ (110\ 00\ 000)\ (110\ 00\ 011)\ (110\ 11\ 011011)$$

$$\mathrm{decode}(10\ 1\ 11)=7$$
$$\mathrm{decode}(110\ 00\ 101)=13$$

$$\text{decode}(110\ 00\ 000) = 8$$
$$\text{decode}(110\ 00\ 011) = 11$$
$$\text{decode}(110\ 11\ 011011) = 91$$

Therefore, the gap in the document list is $[1, 2, 7, 13, 8, 11, 91]$, the document ID is $[1, 3, 10, 23, 31, 42, 133]$.

Q4

The line 21 in Figure 2 that causes the bug, loop happens when pivot is unchanged, because *pickTerm* selects the term with the maximal idf, if we have terms in order A, B, C. C is pTerm return by *findPivotTerm(terms, θ)*, B have the same DID as pivot with the largest idf, and A have the smallest DID. Thus, *pickTerm(terms[0..pTerm-1])* will always return term B. Since B have the same DID as pivot, it will not change the posting. Therefore, the posting remains the same, resulted in the pivot is unchanged.

For example:
Threshold = 8

	A	B	C
UB	3	4	4
List	$\langle 1,3\rangle$	$\langle 1,4\rangle$	$\langle 2,4\rangle$
	$\langle 2,3\rangle$	$\langle 4,2\rangle$	$\langle 4,3\rangle$

Assume A have the smallest idf
First time:
pivot is C.DID, and pickTerm return B. Thus, B skip to $\langle 4,2\rangle$, the order after sort is A, C, B.

Second time:
pivot is B.DID, and pickTerm return C. Thus, C skip to $\langle 4,3\rangle$, the order after sorting is A, C, B.

Infinity loop start:
the pivot is always B.DID, and pickTerm always return C, and try to skip to B.DID, which is this case is 4. Since C.DID is also 4, the posting have nothing change.