COMP6714 ASSIGNMENT 1 SAMPLE SOLUTION

Q1.

1. Idea: we reduce this case to the PositionalIntersect algorithm with varying k values. Lines 11 and 17 are updated to reflect the new ordering requirement.

See Algorithm 1.

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

```
1  answer ← ∅;
2  while p1 ≠ nil ∧ p2 ≠ nil do
3      if docID(p1) = docID(p2) then
4          l ← [ ];
5          pp1 ← positions(p1); pp2 ← positions(p2);
6          while pp1 ≠ nil do
7              skipTo(p$, docID(p1), pp1);
8              s ← pos(p$);
9              k ← s − pos(pp1) − 1;
10             while pp2 ≠ nil do
11                 if pos(pp2) − pos(pp1) ∈ [1, k] then
12                     add(l, pos(pp2));
13                 else
14                     if pos(pp2) > pos(pp1) then
15                         break;
16                 pp2 ← next(pp2);
17             while l ≠ [ ] ∧ l[1] ≤ pos(pp1) do
18                 delete(l[0]);
19             for each ps ∈ l do
20                 answer ← answer ∪ [docID(p1), pos(pp1), ps];
21             pp1 ← next(pp1);
22         p1 ← next(p1); p2 ← next(p2);
23     else
24         if docID(p1) < docID(p2) then
25             p1 ← next(p1);
26         else
27             p2 ← next(p2);
28 return answer;
```

Q2.

1. The i-th generation of sub-index will have a size of $2^i \cdot M$. Therefore, the largest generation will be $i_{max} = \lceil \log_2 t \rceil$. Therefore, there will be at most $i_{max} + 1 = O(\log_2 t)$ sub-indexes.

2. Consider any entry in a sub-index. It will **only** move to a higher generation sub-index (but **never** downgrade to a lower generation sub-index). Every time such a "promotion" is performed, this entry will be read and written once each. Therefore, the maximum number of times an entry will be involved in an I/O operation during the *whole* process is at most $2 \cdot \log_2 t$ times. Therefore, the entire I/O cost is bounded by $(tM) \cdot (2 \log_2 t)$.

Q3.

- After decoding, the gaps are: [1, 2, 7, 13, 8, 11, 91]
- The docIDs are: [1, 3, 10, 23, 31, 42, 133]

Q4.

- It is in Line 21. It cannot pick *any* term preceding the Pivot Term, as the only information we know is that the first term (in sorted order) does not point to the Pivot Doc. The detailed description in Sec 2.4 (last paragraph in the section) describes a heuristic of always choosing the one with the maximum idf. If this terms's current pointer is already on the Pivot Doc, then the algorithm runs in an infinite-loop.
- See one example below. All terms have the same idf. Hence it is possible that "pickTerm()" always chooose term C.

	A	B	C
UB	5	7	3
List	$\langle 1,3\rangle$ $\langle 2,3\rangle$ $\langle 8,5\rangle$	$\langle 1,4\rangle$ $\langle 7,7\rangle$	$\langle 1,4\rangle$ $\langle 7,3\rangle$