COMP 6714 Assignment 1
Yu Han Z5219071

Q1

(1) Intersect (A, B):

```
If A.len == B.len == 1 then
    Do If A == B then Return A;
    Else Return [ ];
Else:
    List A_less, A_greater, B_less, B_greater;
    V = sum(A) / A.len;
    For i in list A:
        Do If i < V then A_less.append(i);
        Else A_greater.append(i);
    For j in list B:
        Do If j < V then B_less.append(j);
        Else: B_greater.append(j);
    Return Intersect (A_less, B_less) + Intersect (A_greater, B_greater);
End;
```

(2) For k sub-lists, we only need to change the List A_less, A_greater and B_less, B_greater to k lists. For example, A_1 … A_k, and B_1 … B_k. And let V be the largest number of A, and V1 = V/(k-1), V2 = V1+V/(k-1) … V(k-1) = V(k-2) + V(k-1). Then traversal list A and B and allocate each value to the suitable range (V1 to V2 to V3 to V(k-1)). Finally, return Intersect (A_1, B_1) + Intersect (A_2, B_2) + Intersect (A_k, B_k).

Q2

(1) The logarithmic merge strategy is similar like the gallop search algorithm. Let's assume the indexes are $I_0,\ I_1, I_2$ ... and the size of the indexes are $2^0 * M,\ 2^1 * M,\ 2^2 * M\ldots$ Postings percolate up this sequence of indexes and are processed only once on each level. As before, up to M pages of postings are accumulated in an in-memory auxiliary index, which we call Z_0. When the limit M is reached, the $2^0 * M$ pages of postings in Z_0 are transferred to a new index I_0 that is created on disk. The next time Z_0 is full, it is merged with I_0 to create an index Z_1 of size $2^{-1} * M$. Then Z_1 is either stored as I_1 (if there isn't already an I_1) or merged with I_1 into Z_2 (if I_1 exists) and so on. We service search requests by querying in-memory Z_0 and all currently valid indexes I_i on disk and merging the result.
For example, we have 7 sub-indexes without merge. In the logarithmic merge algorithm, we'll combine four old sub-indexes as a new one and combine another two old sub-indexes as a new one, and the rest one as another new one. Therefore, we used $\log_2 t$.

(2) Because each page is processed only once on each of the $\log_2 t$ levels, and the total number of pages is t * M, the total I/O cost for the logarithmic merge is O (t * M * $\log_2 t$).

Q3

(1) Precision = 6 / 20 = 0.3

(2) Recall = 6 / 8 = 0.75

F1 = 2PR / (P + R) = 3 / 7 = 0.43

Precision & Recall Table

K	1	2	3	4	5	6	7	8	9	10
Precision (%)	1/1	2/2	2/3	2/4	2/5	2/6	2/7	2/8	3/9	3/10
Recall (%)	1/8	2/8	2/8	2/8	2/8	2/8	2/8	2/8	3/8	3/8

K	11	12	13	14	15	16	17	18	19	20
Precision (%)	4/11	4/12	4/13	4/14	5/15	5/16	5/17	5/18	5/19	6/20
Recall (%)	4/8	4/8	4/8	4/8	5/8	5/8	5/8	5/8	5/8	6/8

(3) 8 * 0.25 = 2

Therefore, the uninterpolated precisions could be 1, 2/3, 2/4, 2/5, 2/6, 2/7, 1/4

(4) The interpolated precision is 4 / 11 = 0.364 at 33% recall.

(5) MAP = 1/8 * (1 + 1 + 3/9 + 4/11 + 5/15 + 6/20) = 0.4163

(6) $MAP_{largest}$= 1/8 * (1 + 1 + 3/9 + 4/11 + 5/15 + 6/20 + 7/21 + 8/22) = 0.5034

(7) $MAP_{smallest}$= 1/8 * (1 + 1 + 3/9 + 4/11 + 5/15 + 6/20 + 7/9999 + 8/10000) = 0.4165

(8) 0.5034 – 0.4163 = 0.0871

Q4

(1) P(Q|d1) = 2/10 * 3/10 * 1/10 * 2/10 * 2/10 * 0/10 = 0

P(Q|d2) = 7/10 * 1/10 * 1/10 * 1/10 * 0/10 * 0/10 = 0

Therefore, the two documents have the same score 0.

(2) P(Q|d1) = (0.8*0.2+0.2*0.8) * (0.8*0.3+0.2*0.1) * (0.8*0.1+0.2*0.025) * (0.8*0.2+0.2*0.025) * (0.8*0.2+0.2*0.025) * (0.8*0+0.2*0.025) = $9.63 * 10^{-7}$

P(Q|d2) = (0.8*0.7+0.2*0.8) * (0.8*0.1+0.2*0.1) * (0.8*0.1+0.2*0.025) * (0.8*0.1+0.2*0.025) * (0.8*0+0.2*0.025) * (0.8*0+0.2*0.025) = $1.3 * 10^{-8}$

P(Q|d1) > P(Q|d2), so document 1 would be ranked higher.