Ass1

Chenqu Zhao z5242692

Q1.(1) The following code is implemented in Python and contains Python operations about list.

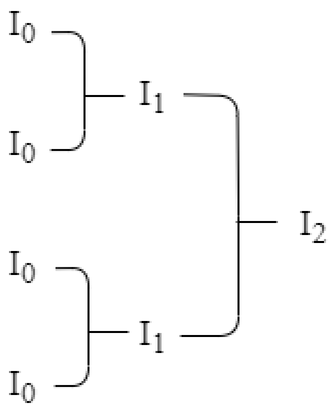
```
result ← < >

Intersect(A, B):
if A.len = 0 or B.len = 0:
    return result
else:
    index ← A[0]
    A_left, A_right, B_left, B_right ← < >, < >, < >, < >
    for i in range(A.len):
        if A[i] < index:
            A_left.append(A[i])
        elif A[i] > index:
            A_right.append(A[i])

    for j in range(B.len):
        if B[j] < index:
            B_left.append(B[j])
        elif B[j] > index:
            B_right.append(B[j])
        else:
            result.append(index)

    Intersect(A_left, B_left)
    Intersect(A_right, B_right)
    return result
```

(2) The boundary case does not change, and the else operations are explained below. When requiring k sub-lists, k-1 indexes should be picked up. As the same logic of question(1), I pick the first k-1 values as indexes in list A (if there are enough, otherwise, just acquire as many as it can) and sort them from small to large. After that, I traverse list A and divide it into k sub-lists using these k-1 indexes. Then, I traverse list B and divide it into k sub-lists using same indexes(append the same values as indexes to result list). Name the k sub_lists of A and B as $A_0, A_1, A_2 \ldots A_{n-1}$ and $B_0, B_1, B_2 \ldots B_{n-1}$.

To use same logic as question (1), do a for loop for m which is an integer from 0 to k -1. invoke Intersect(A_m, B_m) for each sub-list pair during each iteration.

Q2.

(1) The logarithmic merge is used with t sub-indexes(I_0) initially, the diagram below indicates partial algorithm.



…

One kind of sub-index could only appear once during each generation, otherwise, it will be merged with sub-index having same subscript to produce a new sub-index on higher level of the binary tree. Therefore, the result of number of sub-indexes is at most ceil($\log_2 t$).

(2) There are M pages in total. During each page processing, according to logarithmic merge, there is only 1 occurrence of sub-index with generation number k.

Therefore, the following recursive calculations can be defined:
Merge k-1 sub-indexes once
Merge k-2 sub-indexes twice
Merge k-3 sub-indexes four times
…
Merge 0 sub-indexes 2^{k-1} times

Regarding to question(1), k = $\log_2 t$

Reading collection: t * M
Merge: $\sum_{i=0}^{k-1} 2^i * (2 * 2^{k-i-1} + 2^{k-i}) * M = k * 2^{k+1} * M = 2 * t * M * \log_2 t$

The total I/O cost is $t * M + 2 * t * M * \log_2 t$
Regarding to Big-O notation, the total I/O cost of the logarithmic merge is $O(t * M * \log_2 t)$.

Q3.

(1) Precision = tp/(tp + fp) = (6/6+14) = $\frac{3}{10}$

(2) Recall = tp/(tp + fn) = (6/6+2) = $\frac{3}{4}$

F1 = [(β^2 + 1)PR] / [β^2P + R] = $\frac{(1+1)*\frac{3}{10}*\frac{3}{4}}{1*\frac{3}{10}+\frac{3}{4}}$ = 0.4286

(3)

doc_id	1	2	3	4	5	6	7	8	9	10
Precision(%)	100	100	66.67	50	40	33.33	28.57	25	33.33	30
Recall(%)	12.5	25	25	25	25	25	25	25	37.5	37.5

doc_id	11	12	13	14	15	16	17	18	19	20
Precision(%)	36.36	33.33	30.77	28.57	33.33	31.25	29.41	27.78	26.32	30
Recall(%)	50	50	50	50	62.5	62.5	62.5	62.5	62.5	75

According to the table above, the uninterpolated precisions of the system at 25% recall are 100%, 66.67%, 50%, 40%, 33.33%, 28.57%, 25%.

(4) The highest precisions found for any recall higher than 33%(doc_id >= 9) is 4/11 = 0.3636

(5) MAP = $\frac{1}{8}*(\frac{1}{1}+\frac{2}{2}+\frac{3}{9}+\frac{4}{11}+\frac{5}{15}+\frac{6}{20}+0+0)$ = 0.4163

(6) The largest possible MAP is $\frac{1}{8}*(\frac{1}{1}+\frac{2}{2}+\frac{3}{9}+\frac{4}{11}+\frac{5}{15}+\frac{6}{20}+\frac{7}{21}+\frac{8}{22})$ = 0.5034

(7) The smallest possible MAP is $\frac{1}{8}*(\frac{1}{1}+\frac{2}{2}+\frac{3}{9}+\frac{4}{11}+\frac{5}{15}+\frac{6}{20}+\frac{7}{9999}+\frac{8}{10000})$ = 0.4165

(8) 0.5034 – 0.4163 = 0.0871
0.4165 – 0.4163 = 0.0002
So, the error is in [0.0002, 0.086]

Q4.

(1) P(Q|d1) = $\frac{2}{10}*\frac{3}{10}*\frac{1}{10}*\frac{2}{10}*\frac{2}{10}*\frac{0}{10}$ = 0

P(Q|d2) = $\frac{7}{10}*\frac{1}{10}*\frac{1}{10}*\frac{1}{10}*\frac{0}{10}*\frac{0}{10}$ = 0

It is not able to rank these two documents.

(2) With smoothing,

P(Q|d1) = $(0.8*\frac{2}{10}+0.2*0.8)*(0.8*\frac{3}{10}+0.2*0.1)*(0.8*\frac{1}{10}+0.2*0.025)*(0.8*\frac{2}{10}+0.2*0.025)*(0.8*\frac{2}{10}+0.2*0.025)(0.8*\frac{0}{10}+0.2*0.025)$ = 9.63 * 10 $^{-7}$

$$P(Q|d_2) = (0.8 * \frac{7}{10} + 0.2 * 0.8) * (0.8 * \frac{1}{10} + 0.2 * 0.1) * (0.8 * \frac{1}{10} + 0.2 * 0.025) * (0.8 * \frac{1}{10} + 0.2 * 0.025) * (0.8 * \frac{0}{10} + 0.2 * 0.025)(0.8 * \frac{0}{10} + 0.2 * 0.025) = 1.30 * 10^{-8}$$

Document d_1 would be ranked higher.