Selamat sore, Pak Alfan.

Saya sempat coba-coba mengerjakan algoritma WAND Top-K Retrieval dan hasilnya saya rasa sudah benar, naun ada beberapa pertanyaan yang ingin saya tanyakan terkait algoritma dan detail implementasi.

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
          DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
            one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
              to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
              one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

- Saya masih agak bingung pada baris 12-13 dan 21-22, apakah yang dilakukan next iterator hanya pada postingList yang setelah disort pada index pertama saja? Saya sempat melakukan implementasi, namun idenya saya setelah mendapatkan success, atau masuk ke baris 16, itu semua iterator yang document idnya sukses iteratornya dimajukan. Alasannya adalah karena apabila hanya terms yang paling atas saja yang dimajukan, akan dilakukan sorting pada baris 4 dengan kompleksitas waktu total Q^2 apabila digunakan insertion sort (memanfaatkan bahwa faktanya sudah ascending kecuali indeks paling bawah aja) dan $Q^2 \log Q$ apabila menggunakan algoritma sort lain, untuk Q merupakan banyaknya terms pada query. Saya rasa ini bisa dioptimisasi dengan langsung memajukan semua terms langsung setelah match pada baris 16, dan akan dilakukan quick sort sekali saja dengan kompleksitas waktu $Q \log Q$.

```
Function next(θ):
    repeat:
        // O(Q log Q)
        quickSort(terms, posting)
        // O(Q)
        computePrefixSum(terms, posting)
        // O(Q)
        pterm := getPivot(terms, posting, threshold)
        if(pivot == lastID) return (NoMoreDocs)
        // kasus pivot <= curDoc tidak mungkin terjadi
        if(pivot == posting[0].DID):
            //Assign jawaban
            curDoc := pivot
```

```
            // O(Q)
            evaluateFull(pivot)
            // O(Q)
            aterm := (posting if posting.iterator == pivot)
            // Amortized O(Σ len(posting))
            // Majuin semua yang sama dengan pivot ke >= curDoc + 1
            posting[aterm] = aterm.iterator.next(curDoc + 1)
        else:
            // Massnya tidak cukup
            // Disini mungkin bisa dioptimisasi lebih kencang
            aterm := (posting if posting.iterator < pivot)
            // Amortized O(Σ len(posting))
            // Majuin semua yang sama dengan pivot ke >= pivot
            posting[aterm] = aterm.iterator.next(pivot)
    end repeat
```

Implementasi saya kira kira berakhir dengan pseudo code yang serupa dengan yang di atas.

```
Processing
----------------
curDoc: 0
threshold: 0
Cetak Pointer:
Heap:
-1 0
-1 0
hujan -> 1.6: (1, 1.5)
turun -> 1.5: (1, 0.7)
deras -> 1.8: (1, 1.2)
Ketemu pivot: 0 hujan
Fully evaluating 1
----------------
curDoc: 1
threshold: 0
Cetak Pointer:
Heap:
-1 0
1 3.4
hujan -> 1.6: (2, 0.4)
turun -> 1.5: (3, 1)
deras -> 1.8: (6, 1)
Ketemu pivot: 0 hujan
Fully evaluating 2
----------------
curDoc: 2
threshold: 0.4
Cetak Pointer:
```

```
Heap:
2 0.4
1 3.4
hujan -> 1.6: (3, 0.6)
turun -> 1.5: (3, 1)
deras -> 1.8: (6, 1)
Ketemu pivot: 0 hujan
Fully evaluating 3
----------------
curDoc: 3
threshold: 1.6
Cetak Pointer:
Heap:
3 1.6
1 3.4
hujan -> 1.6: (6, 1)
turun -> 1.5: (6, 1.5)
deras -> 1.8: (6, 1)
Ketemu pivot: 0 hujan
Fully evaluating 6
----------------
curDoc: 6
threshold: 3.4
Cetak Pointer:
Heap:
1 3.4
6 3.5
deras -> 1.8: (7, 0.5)
hujan -> 1.6: (8, 1.5)
turun -> 1.5: (8, 1.5)
Ketemu pivot: 1 hujan
----------------
curDoc: 6
threshold: 3.4
Cetak Pointer:
Heap:
1 3.4
6 3.5
hujan -> 1.6: (8, 1.5)
turun -> 1.5: (8, 1.5)
deras -> 1.8: (10, 0.6)
Ketemu pivot: 2 deras
----------------
curDoc: 6
threshold: 3.4
Cetak Pointer:
Heap:
1 3.4
6 3.5
```

```
turun -> 1.5: (10, 0.3)
deras -> 1.8: (10, 0.6)
hujan -> 1.6: (11, 1.6)
Ketemu pivot: 2 hujan
----------------
curDoc: 6
threshold: 3.4
Cetak Pointer:
Heap:
1 3.4
6 3.5
deras -> 1.8: (11, 1.8)
hujan -> 1.6: (11, 1.6)
turun -> 1.5: (12, 1.1)
Ketemu pivot: 1 hujan
Fully evaluating 11
----------------
curDoc: 11
threshold: 3.4
Cetak Pointer:
Heap:
11 3.4
6 3.5
turun -> 1.5: (12, 1.1)
deras -> 1.8: (2147483647, 2.14748e+09)
hujan -> 1.6: (2147483647, 2.14748e+09)
Fully evaluated count: 5
11 3.4
6 3.5
```

Kira-kira seperti di atas jalannya untuk K = 2. Saya belum mencoba membandingkan dan melakukan eksperimen panjang untuk variasi implementasi dari algoritmanya yang saya juga belum terlalu yakin dengan kebenarannya.