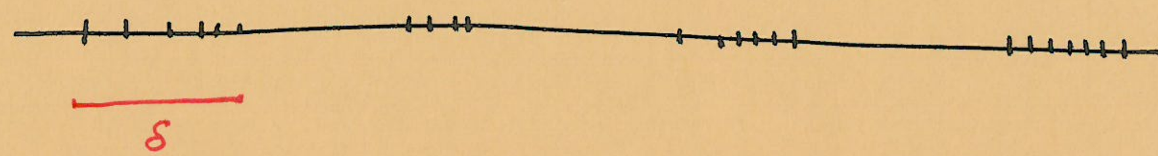
Solutions for the δ-interval covering problem:

Input:



- Assume that OPT number of δ-intervals are used in the optimal solution; moreover, let these intervals cover $k_1, k_2, \cdots, k_{OPT-1}, k_{OPT}$ points in A.
- Clearly, we have $k_1 + k_2 + \cdots + k_{OPT-1} + k_{OPT} = n$

---

① We use a greedy algorithm to compute the points covered by the first interval as follows:

1. Start from $i = 2, 3, \cdots$ (until k_1)
   If $d(A[1], A[i]) \leq \delta$ then $i{+}{+}$;
   else return $k_1 \leftarrow i-1$.
2. Repeat the above procedure to compute $k_2, k_3, \cdots, k_{OPT}$. // adjust indices

Analysis: It takes $O(n)$ time to compute k_1, and OPT might be $O(n)$.

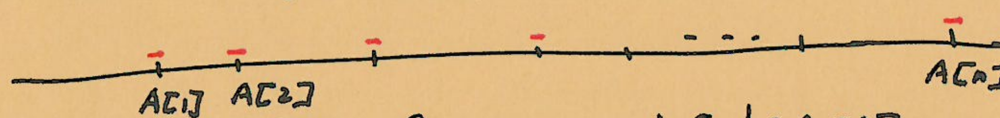
$\therefore$ The running time is $O(n^2)$.

// correct, but the analysis is too coarse!

New analysis: It takes $O(k_1)$ time to compute k_1, so the total running time would be

$$O(k_1) + O(k_2) + \cdots + O(k_{OPT}) = O(n).$$

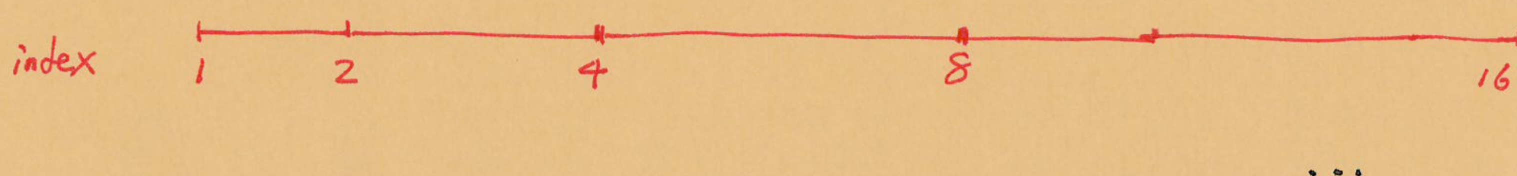
This is in fact optimal in the worst case — just let δ be very small (like ε), then you must use n δ-intervals.



For instance let $\delta = \varepsilon < \min_{i=1..n-1}\{d(A[i], A[i+1])\}$.

In a lot of cases, <u>worst</u> case might not happen, so it would be good to analyze (& design) an algorithm to handle this:

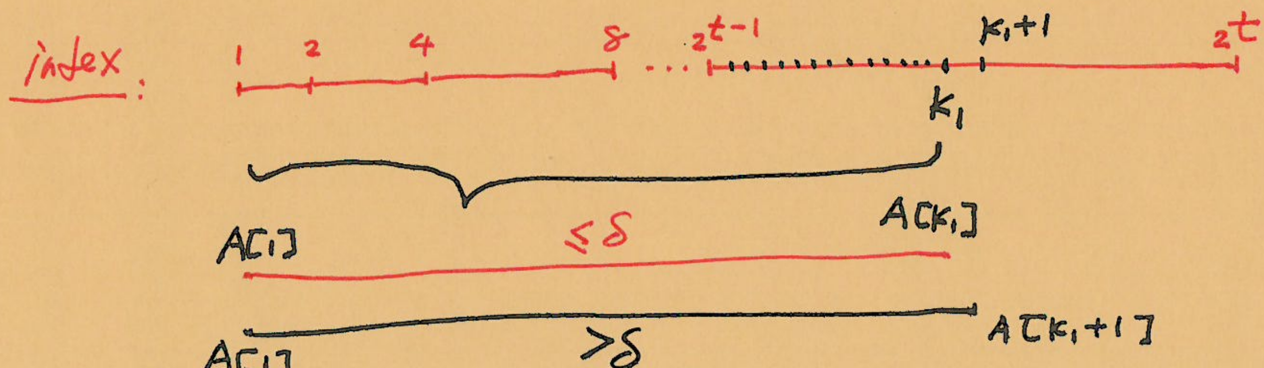
Exponential search:



② **Algorithm**:

(1) search with $t = 1, 2, 3, \cdots$ the first t such that $d(A[1], A[2^{t-1}]) \leq \delta$, but $d(A[1], A[2^t]) > \delta$.
Then, find the breakpoint k_1 in $A[2^{t-1} .. 2^t]$ such that $d(A[1], A[k_1]) \leq \delta$ but
$d(A[1], A[k_1+1]) > \delta$,
using binary search.

(2) Repeat the above process on $A[k_1+1 .. n]$ to find the remaining breakpoints (or, segments in A with length $k_2, k_3, \cdots, k_{OPT}$)



Claim: It takes $O(\log k_1)$ to compute k_1.

Reason: * $t-1, t \in O(\log k_1)$.

* binary search in the interval $[2^{t-1}, 2^t]$ would also take $O(\log k_1)$ time — as $2^t - 2^{t-1} = 2^{t-1}$, which is of size $O(k_1)$.

Claim: The total running time is

$$O(\log k_1) + O(\log k_2) + \cdots + O(\log k_{OPT})$$

$$= O(\log k_1 \times k_2 \times \cdots \times k_{OPT})$$ // $\log a + \log b = \log a * b$

$$\leq O\left(\log\left(\frac{k_1 + k_2 + \cdots + k_{OPT}}{OPT}\right)^{OPT}\right)$$ // $a_1 \cdot a_2 \cdots a_y \leq \left(\frac{a_1 + a_2 + \cdots + a_y}{y}\right)^y$, the inequality of arithmetic and geometric means

$$= O\left(\log\left(\frac{n}{OPT}\right)^{OPT}\right)$$

$$= O\left(OPT \cdot \log\frac{n}{OPT}\right)$$

$$= \begin{cases} O(n), & \text{if } OPT = c_1 \cdot n,\ c_1 \leq 1. \\ O(\sqrt{n}\log n), & \text{if } OPT = c_2\sqrt{n}, \text{ for some } c_2 > 0. \\ O(\log n), & \text{if } OPT = O(1). \end{cases}$$

- This is just an example of analysis of algorithms, which should be covered (& probably tested) in 432. So no need to worry about this.

Review for Test 2:

1. CFL basics: Chomsky Normal Form,
   design CFG's for different CFL's,
   ambiguity,
   CFL = PDA.
2. The pumping lemma for CFL.
3. Countability, diagonalization method, uncountable.
4. TM basics: Church-Turing thesis,
   decidable languages,
   undecidable languages
   (A_{TM}, E_{TM}, EQ_{TM}, E_{LBA}, ALL_{CFG}).
5. Reducibility.