a) En vez de pensar en número elementos en el arbol podemos pensar en el número de comparaciones que se deben hacer para alcanzar un valor k lo cual serían 1 + el número de nodos hasta el nodo que se busca (la profundidad). La profundidad del nodo final puede verse de manera recursiva sumando las profundidades de cada nodo que hay en el camino para alcanzarlo. En un caso promedio el porcentaje de comparaciones para alcanzar k será CN/N donde CN es la profundidad del nodo N sobre el número de nodos y establecemos nuestros casos base para un nodo vacío y un nodo único C0=C1=0. De esta forma la manera recursiva sería
   CN = (C0 CN1)/ N + (C1 CN2)/N . . . (CN1 C0)/N
   Esta forma recursiva tiene aproximación a 2NnN que se aproxima a 1.39lnN
b) En un BST se sabe que camino coger para alcanzar la respuesta ya que está ordenado de tal manera que no se recorren sub árboles innecesarios pues se sabe por comparación que la respuesta no puede estar ahí. En una tabla de hash se deben recorrer los elementos hasta obtener el resultado y el tiempo para esta operación depende de la carga de la tabla de hash, si la carga es muy alta se debe iterar sobre buckets muy seguido causando que incremente la velocidad. De esta manera si se responde a la misma consulta con tablas de hash el tiempo de respuesta aumenta.
c) Utilizan la función values(map, keylo, keyhi), que retorna todos los valores del árbol que se encuentren entre [keylo, keyhi] en un BST (binary search tree). En este caso map es analyzer['dateIndex'], el keylow es la fecha inicial y la keyhigh es la fecha en que termina el rango.