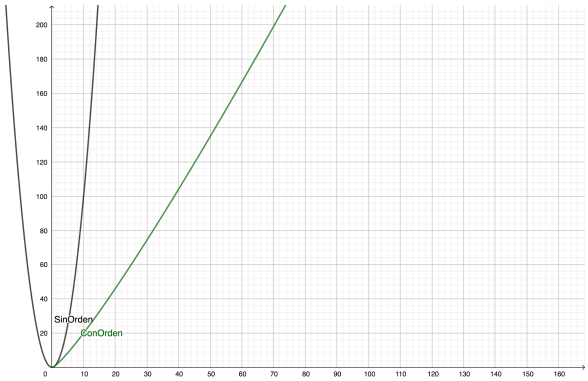
Juan Sebastian Pardo - 201923794 - j.pardor@uniandes.edu.co
Daniela Echeverry - 201822966 - d.ricaurte@uniandes.edu.co

En general, para los requerimientos del reto usamos principalmente los siguientes métodos.

1. **sublistaR1(Comparator<T> comparador, T elemento)**
    a. Descripción: El método genera una sublista del arreglo original con todos los elementos que tengan las características deseadas, que están incluidas en el elemento recibido por parámetro.
    b. Orden de crecimiento:$O(N)$debido a que realiza un recorrido por todos los elementos.
    c. Complejidad Espacial:$O(N)$
    d. Justificación: Escogimos usar este método porque facilita mucho todos los requerimientos. Además, teniendo en cuenta que el archivo de videos-all contiene más o menos 370,000 datos, lo más eficiente es que este sea el único método que se use sobre la lista original debido a su bajo orden de crecimiento.
2. **MergeSort(lLista<T> lista, Comparator<T> comparador, boolean ascendente)**
    a. Descripción: Este método organiza la lista dividiéndola y organizandola por mitades. Para nuestros métodos, este método nos permite organizar la lista recibida por parámetro, que siempre va a ser una sublista hecha por **sublistaR1()**, y dependiendo de parámetro podemos manipularla como queramos. Para obtener los videos con más likes el parámetro se compara mediante el número de likes. Para contar cuantos días en tendencia dura cada video, se comparan los elementos mediante nombre, con eso quedan agrupadas todas las ocurrencias de cada video facilitando su uso.
    b. Orden de Crecimiento: $O(Nlog(N))$
    c. Complejidad Espacial: $O(Nlog(N))$
    d. Justificación: La razón por la que escogimos este algoritmo es que como el objetivo es poner analizar datos de videos de Youtube, sin importar cual o de cuando es la muestra, en realidad no sabemos cuantos videos saldrán en cada sublista, ni que tan organizados estarán. Teniendo en cuenta esto, la consistencia de orden de crecimiento de MergeSort entre sus diferentes casos es favorable sobre otros algoritmos de ordenamiento.
3. **mayorContado(Comparator<T> comparador)**
    a. Descripción: El método recorre una lista ordenada para contar cuál es el elemento con más ocurrencias usando el comparador recibido por parámetro. Para el reto, este método se utiliza para contar cuantos días un video específico fue tendencia. Este método retorna tanto el elemento con mayores ocurrencias, como el número de ocurrencias de dicho elemento. Para hacer esto, se retorna el String() del elemento junto al número.
    b. Orden de crecimiento: $O(N)$ya que solo realiza un recorrido por todo el arreglo ordenado.
    c. Complejidad Espacial: $O(1)$
    d. Justificación: Este método es bastante beneficioso debido a su orden de crecimiento, aunque esto solo se debe a la organización que se hacía antes. De

lo contrario, tocaría hacer un doble recorrido para contar las repeticiones de cada elemento de la lista, lo que haría que su orden de crecimiento fuera $O(N^2)$. Para visualizar mejor esto, se puede observar la siguiente gráfica, que compara el orden de crecimiento con una lista previamente ordenada usando MergeSort ($NlogN + N$) con el de contar sin ordenar previamente la lista (N^2). Como se puede observar, organizar previamente la lista es considerablemente superior incluso en valores bajos de N.



Que se hace antes de los requerimientos:

1. Cargar las categorías a un arregloDinamico desde el archivo csv
2. Cargar los datos a un arregloDinamico desde el archivo csv
3. Hacer 2 copias del arreglo con los datos
    a. **Lista1**
    b. **Lista2**
4. Ordenar **Lista1** por views de manera descendente usando mergeSort. $O(Nlog(N)) - Espacial\ O(Nlog(N))$
5. Ordenar **Lista2** por likes de manera descendente usando mergeSort. $O(Nlog(N)) - Espacial\ O(Nlog(N))$
6.

Se hacen diferentes copias de las listas con diferentes criterios de ordenamiento, debido a que al momento de ejecución esto nos permite que la ejecución de los requerimientos sea mucho más rápida, ya que facilita saber qué datos se necesitan, gracias a que ya están parcialmente ordenados para cada uno de los requerimientos del programa.

Requerimientos: Descripción general y su respectivo orden de crecimiento y complejidad espacial (solo para métodos con orden distinto a $O(1)$.Vale aclarar que todos los métodos a partir de **sublistaR1()** en realidad no son del mismo orden N que **sublistaR1()** sino alguna fracción de N, que son los datos restantes después de la filtración:

1. Descripción: Obtener los n videos con mas vistas para un dado país y categoría.
    a. Orden de ejecución:
        i. Obtener el país y la categoría
        ii. Crear un nuevo YoutubeVideo con esos dos parámetros

iii. Crear un comparador de tipo YoutubeVideo que revise si los dos elementos tienen los mismos atributos deseados, esto evita que se hagan dos sublistas una para país y otra para categoría, haciendo más eficiente el proceso.
iv. Usar el método **sublistaR1()** sobre la **Lista1** con los dos elementos creados anteriormente. $O(N) - Espacial: O(N)$
v. Organizar descendentemente por views la lista obtenida en el paso anterior usando **mergeSort().** $O(NLog(N)) - Espacial: O(NLog(N))$
vi. Mostrar en consola los n videos con mas vistas usando un for y getElement()

b. Orden de Crecimiento total: $O(Nlog(N))$

2. Descripción: Obtener el video con más días en tendencia para un país específico
   a. Orden de ejecución:
      i. Obtener el país deseado
      ii. Crear un nuevo YoutubeVideo con ese país.
      iii. Crear un comparador de tipo YoutubeVideo que revise si los dos elementos son del mismo país.
      iv. Usar el método **sublistaR1()** sobre la **Lista0** con los dos elementos anteriores. $O(N)$
      v. Contar cual es el video con mas repeticiones en la lista ordenada, usando un comprador por nombre. $O(N) - O(1)$. Este retorna la información del video y el número de ocurrencias.
      vi. Mostrar en consola cual es el video con más días en tendencia en consola, usando un String[] para separar el resultado del método anterior.
   b. Orden de Crecimiento total: $O(N)$
   c. Quien lo implementó: Daniela Echeverry
3. Descripción: Obtener el video con más días en tendencia para una categoria especifica
   a. Orden de ejecución:
      i. Obtener la categoría deseada por el usuario
      ii. Crear un nuevo YoutubeVideo con esa categoría
      iii. Crear un comparador de tipo YoutubeVideo que revise la categoría
      iv. Usar el método **sublistaR1()** sobre la **Lista0** con los dos elementos anteriores. $O(N) - O(N)$
      v. Contar cual es el video con mas repeticiones en la lista ordenada, usando un comprador por nombre. $O(N) - O(1)$
      vi. Mostrar en consola cual es el video con más días en tendencia en consola, usando un String[] para separar el resultado del método anterior.
   b. Orden de Crecimiento total: $O(N)$
   c. Quien lo implementó: Juan Sebastian Pardo
4. Descripción: Obtener los n videos de un país y con una etiqueta específica, con mas likes.
   a. Orden de ejecución:
      i. Obtener el país y tag deseados por el usuario
      ii. Crear un nuevo YoutubeVideo con ese país y tag

iii. Crear un comparador de tipo YoutubeVideo que revise que el video a comparar es de ese país, y que contenga el tag deseado.
iv. Usar el método **sublistaR1()** sobre la **Lista2** con los dos elementos anteriores. $O(N)$
v. Ordenar la lista obtenida anteriormente de manera descendente por likes usando **mergeSort().** $O(Nlog(N)) - Espacial: O(Nlog(N))$
vi. Mostrar los n videos con mas vistas en consola usando un for y getElement()

b. Orden de Crecimiento total: $O(Nlog(N))$