Instituto Tecnológico de Costa Rica, Análisis de Algoritmos
Prof. Rodrigo Núñez, Caso #1 – 5%, individual
Objetivo: Replantear soluciones de algoritmos tomando en cuenta cantidad de iteraciones, operaciones o comparaciones buscando reducirlas todo lo posible.

Proceda a escribir en C++ dos algoritmos para los primeros tres ejercicios, buscando que uno de ellos sea por mucho mejor que el otro. En los comentarios de cada implementación de cada ejercicio haga una explicación donde deje claro la diferencia entre las dos implementaciones que hace a una mejor que la otra.

1. Compare the Triplets | HackerRank
2. Time Conversion | HackerRank
3. Subarray Division | HackerRank

Escriba en C++ un algoritmo lo más optimizado posible para los siguientes tres problemas de hackerrank.

4. The Minion Game | HackerRank
5. Cipher | HackerRank
6. Pairs | HackerRank

Deberá entregar el ejercicio al asistente, al correo kevinqj.2002@gmail.com el día 28 de febrero a las 6pm. Incluyendo:

- Link al repositorio de git
- Un solo archivo .cpp que tenga todas las implementaciones de los ejercicios y un main que haga las llamadas de prueba, haciendo hasta 2 pruebas por ejercicio
- Un Makefile para el compilado del caso
- Si hay algún detalle que el asistente deba saber anotarlo en el readme.md
- Deberá usar gcc o c++ para linux, en caso de usar Windows use un c++ de ansi, eso excluye al compilador default que trae c-lion, visual studio, c++ buider

Instituto Tecnológico de Costa Rica, Análisis de Algoritmos
Prof. Rodrigo Núñez, Caso #2 – 10%, individual
Objetivo: Aprender y calcular el O grande de diversos algoritmos para que funcionen como referencia al comparar los rendimientos de algoritmos en el futuro.

El algoritmo de Quicksort podría ser en algunos casos O(n log(n)) y en otros es O(n^2), mismos casos que podrían verse cambiados cuando se trata de un pivote fijo a uno seleccionado aleatoriamente.

En el mismo sentido, el insertion sort podría ser en algunos casos O(n) y en otros casos O(n^2).

Otro problema importante para analizar es lo que resuelven muchos editores de texto cuando se hacen búsquedas del tipo "free text". Es decir, dado un texto de larga extensión, si se hace la búsqueda de "cal" por ejemplo, puede decir la cantidad de apariciones de "cal" y dichas apariciones, por ejemplo: cal, cala, calor, calentura, caléndula, calzado… Este problema fácilmente puede programarse en O^2 o incluso un caso peor.

Proceda a programar los 3 algoritmos anteriores demostrando programáticamente, con los resultados y análisis obtenidos de experimentos, debidamente documentados en el código; todos los casos anteriores:

- Quicksort logarítmico
- Quicksort cuadrático
- Influencia del pivote fijo y aleatorio
- Insertion sort lineal
- Insertion sort cuadrático
- Lograr un freetext search en tiempo logarítmico o lineal. Se recomienda usar un texto de mínimo 10MB.

Deberá entregar al asistente al correo kevinqj.2002@gmail.com los siguientes entregables:

1. Link al repositorio de git de la solución
2. Un archivo cpp separado para cada uno de los 3 algoritmos
3. Un archivo Makefile para compilar
4. Deberá usar c++, g++ de linux, en el caso de usar Windows no utilice compiladores que no sean ANSI, es decir, evite los que trae por default clion, visual studio y similares
5. En el repositorio de git debe tener un readme.md que explique debidamente los experimentos y resultados obtenidos, la forma en que justifica que logró demostrar que los algoritmos programados y las pruebas hechas cumplen los tiempos algorítmicos que se indican en los supuestos
6. Entrega será el día 24 de marzo a las 9pm

Instituto Tecnológico de Costa Rica, Análisis de Algoritmos
Prof. Rodrigo Núñez, Caso #3 – 25%, grupos de 2
Objetivo: Practicar la destreza de seleccionar una estrategia algorítmica para resolver problemas específicos que unidos logran una única misión en un programa.

El siguiente caso para resolver le plantea al estudiante una problemática desde varios puntos de vista:

a) Desde la orientación a objetos que busca un diseño estratégico de las clases
b) El uso de patrones orientados a objetos los cuales tienen una aplicación efectiva para lograr crear programas más claros y escalables
c) Un conjunto de sub-retos y una serie de posibilidades algorítmicas para resolver el reto
d) La optimización de los algoritmos para obtener mejores tiempos de respuesta

El programa que se quiere resolver se basa en un archivo SVG semilla con una imagen cualquiera. El usuario indica una lista de puntos absolutos [(X, Y)…(X,Y)] de la imagen, una lista de colores de enfoque [#7DA367, #57ABE7,…,#4BFBFD], un ángulo en radianes y un entero que indica la cantidad de frames a producir.

Ahora el programa debe lograr lo siguiente:

selección: Encontrar todos los elementos del SVG cercanos o intersecados por la lista de puntos y la lista de colores, aproximando con ello los elementos del SVG que van a ser seleccionados.

enrutamiento: calcular para todos los elementos su posición inicial y su destino, determinado este por un cálculo de distancia según el tamaño del SVG, el ángulo deseado, y el tipo de ruta (recta o curva) dado la cantidad de frames y optimizado al menor costo. La cantidad de frames determina los pasos que se van a hacer en los diversos elementos del SVG al punto destino; manteniendo en la medida de lo posible los colores de enfoque unidos por color.

generación: con todos esos cálculos ya hechos, proceda a generar múltiples archivos SVG con el mismo nombre del original, pero numerados 1,2,3… realizando dentro del SVG los movimientos de desplazamiento según los cálculos hechos. En el proceso de generación, se verifica que efectivamente cada elemento se desplazó con respecto al paso anterior, pero si dicho desplazamiento no es notorio, entonces se descarta el movimiento del elemento.

Los archivos anteriores producen una pseudo animación, la cuál será verificada con el programa de svganimation proporcionado por el profesor.

Usted cuenta con las siguientes estrategias: divide y vencerás, programación dinámica, algoritmo voraz y backtracking para resolver los 3 problemas: selección, enrutamiento y generación; y solo podrá usar una estrategia por problema, usted decidirá cuál le conviene más en cada caso. Además, cualquier solución que implemente debe ser máximo O(n log n), llamadas anidadas son equivalentes a for anidados. Deje en los comentarios del código el análisis respectivo.

Elementos prácticos

- Todo se debe hacer orientado a objetos en C++
- La escritura de los archivos SVG en disco se deben hacer con patrón productor consumidor, con un single thread consumer
- La activación de cada uno de los procesos se debe hacer siguiendo el observer pattern
- Deberá seguir el estándar de código proporcionado por el profesor
- Utilice alguna librería de XML para procesar el SVG
- Se evaluará calidad de las clases, encapsulamiento, uso de polimorfismo,

abstracción, separación clara de funciones en classes y namespaces

- Cualquier sospecha de copia anulará el trabajo
- El control de versiones deberá ser con github, solo ser revisará lo que se encuentre en el branch (main), no borre ningún branch, debe ser evidente branches personales y branches usados para hacer merge entre los integrantes
- El caso se revisará con cita con el profesor entre los días 20 a 25 de abril