Tecnológico de Monterrey, Campus Monterrey

Departamento de Ciencias Computacionales

**ANALISIS Y DISEÑO DE ALGORITMOS** - Ing. Román Martínez M.

**PRIMER EXAMEN DE PROGRAMACIÓN**  - 4 de Marzo 2015

# Nombre: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Matrícula: \_\_\_\_\_\_\_\_\_\_\_\_

**Instrucciones:**

* Para resolver este examen puedes utilizar el compilador de C++ de tu preferencia para programar tus respuestas.
* Se te provee de un archivo llamado ex1em15.cpp en donde se encuentra cierto código que utilizarás y en ese mismo archivo programarás tus respuestas.
* Identifica tu archivo con un comentario inicial con tu nombre y matrícula, y documenta el código con los comentarios que consideres importantes y necesarios para explicar lo que programas.
* El archivo final de entrega como respuesta de tu examen, deberá subirse a la página del curso en el espacio correspondiente.
* Está estrictamente PROHIBIDO el acceso a códigos previamente realizados y cualquier tipo de comunicación electrónica o presencial. Cualquier situación deshonesta será penalizada con una calificación de DA.

Revisa el código fuente que se encuentra en el archivo ex1em15.cpp. Podrás observar que se encuentran definidos algunos arreglos que utilizarás y que ya está la implementación del algoritmo del Quicksort y el de Floyd. Usa este código para resolver los problemas II y III que se presentan a continuación. El problema I no requiere de ningún código adicional. Para probar el programa, deberás seguir las instrucciones que se indican y responder las preguntas asociadas a cada problema. El ALGORITMARIO contiene únicamente el algoritmo de la búsqueda binaria por si te fuera útil.

***PROBLEMA 1.*** *(30 puntos – Tiempo sugerido para solucionar el problema: 20 min).*

Utiliza la técnica de Divide y Vencerás para implementar una función que sirva para obtener el valor mayor de un arreglo de números enteros.

(Sugerencia: Este caso se resuelve con la misma estrategia del problema de los puntos más cercanos, con la ventaja de que en este caso ya no se tienen que hacer cálculos y los datos a comparar son directos).

Responde en base a los resultados de tu programa:

* ¿Cuál es el valor mayor del arreglo de datos? \_\_\_\_\_\_
* ¿Cuál es el orden de complejidad del algoritmo implementado? \_\_\_\_\_\_\_\_\_
* ¿Representa alguna ventaja esta implementación con la técnica de divide y vencerás con respecto a una solución tradicional?

***PROBLEMA 2.*** *(35 puntos – Tiempo sugerido para solucionar el problema: 30 min).*

En el archivo ex1em15.cpp contiene la implementación del algoritmo del Quicksort pero utiliza una versión diferente para la función de la partición que es un poco más intuitiva que la que se vio en clase. Analiza esta implementación para poder realizar los cambios que se describen a continuación.

Como ya conoces, este algoritmo tiene la desventaja de que cuando la partición es desbalanceada en la cantidad de datos en los subarreglos, el ordenamiento se comporta en el peor caso con un orden cuadrático, a diferencia del orden O(n log n) que ocurre cuando la partición genera subarreglos de un tamaño balanceado.

Modifica la implementación del Quicksort para que considere la siguiente mejora que trate de evitar el comportamiento cuadrático en el peor caso:

1. Calcula el tamaño de cada subarreglo después de hacer la partición.
2. Mientras el tamaño de un subarreglo sea menor a 1/3 del tamaño total, deberá hacerse una nueva partición utilizando otro dato como pivote. El nuevo dato pivote será el dato que le sigue al pivote anterior secuencialmente en la lista de datos. Este ciclo asegurará que la partición a ordenar tiene al menos un balance de 1/3 y 2/3 de los datos en cada subarreglo, y eso impactará en la eficiencia.

Para probar tu implementación, utiliza el arreglo de datos que ya contiene el archivo y agrega instrucciones que desplieguen el tamaño de los subarreglos cada vez que se hace una partición.

Responde lo siguiente:

* ¿De qué tamaño son los subarreglos en la primer partición normal del algoritmo con los datos dados?
* ¿De qué tamaño son los subarreglos en la primer partición balanceada del algoritmo con los datos dados?
* ¿Cómo afecta en el orden de complejidad este cambio en el algoritmo? ¿lo mejora o lo empeora? ¿resuelve realmente el problema de que el peor caso del Quicksort es O(n2)? Explica y justifica brevemente.

***PROBLEMA 3.*** *(35 puntos – Tiempo sugerido para solucionar el problema: 30 min).*

En el archivo ex1em15.cpp podrás encontrar la definición de una matriz de datos que representa los costos de trasportación por carretera de un nodo (ciudad) a otro. La matriz guarda el dato 9999 para representar el valor infinito. Los nodos están identificados del 0 al 6. Adicionalmente, podrás observar que ya se tiene implementado el algoritmo de Floyd.

Modifica el algoritmo de Floyd para que además de calcular los costos mínimos para ir de una ciudad a otra, obtenga la cantidad de ciudades en las que se hace escala en la trayectoria del camino más corto.

Responde las siguientes preguntas para los datos que se dan en el archivo:

* ¿Cuántas trayectorias de ciudad a ciudad no implican hacer escalas? \_\_8\_\_\_
* ¿Cuál es la cantidad máxima de escalas que se realizan las trayectorias óptimas entre las ciudades? \_5\_\_\_
* ¿Cuántas escalas se realizarían en la trayectoria óptima de la ciudad 3 a la ciudad 5? \_\_4\_\_\_

**Una vez que termines el examen y lo hayas subido a la página del curso, entra a expresatec.mty.itesm.mx y escribe cuál es tu opinión del examen y tu resultado esperado.**

**ALGORITMARIO**

**Algoritmo recursivo en pseudocódigo de la búsqueda binaria**

**function busca (inicio, fin: index) : index**

**if (inicio > fin) then return 0;**

**else**

**mitad = (inicio + fin) div 2;**

**if (x == arreglo[mitad]) then return mitad;**

**else if (x < arreglo[mitad]) then**

**return(busca(inicio, mitad-1));**

**else return(busca(mitad+1,fin));**