Tarea 2
INFO053 - Estructuras de Datos y Algoritmos.

Académico: Héctor Ferrada.
Instituto de Informática, Universidad Austral de Chile.

Julio 13, 2018

Entrega. En un archivo tarea2Alumno1Alumno2.zip (máximo 2 integrantes), con su implementación (que incluya el Makefile) y un informe, por correo con asunto INFO053 TAREA 2 a **jorge.delgado01@alumnos.uach** a más tardar el Lunes 6 de Agosto.

Informe. Detallado, claro y bien elaborado, donde explique: su metodología de trabajo, la implementación y entregue las conclusiones de su trabajo.

Implementación. La eficiencia de su código es un factor importante en la evaluación. Codifique ordenadamente, incluyendo comentarios en las cabeceras de las funciones/procedimientos explicando que reciben, que hacen y que devuelven. Será inaceptable que se entregue un código con errores de compilación, o que no incluya el Makefile y un *readme* explicando como se ejecuta.

1. Algoritmos de Ordenamiento

Dado n leído desde los argumentos del programa, cree un arreglo de enteros aleatorio $A[0..n-1]$, con $A[i] \leq$MAX; MAX definido en la cabecera del programa. Dentro de un método, sortA(int $*A$, int n), debe comparar los tiempo que tardan en ordenar el arreglo A para 4 alternativas:

1. Para los algoritmos quickSort, heapSort y bucketSort (implementados por ud.), donde debe considerar lo siquiente:

   a) Para quickSort, debe decidir como escoger el pivote y la estrategia en su algorítmica; p.ej, si usar recursividad o no.

   b) Para bucketSort, implemente una variante a la vista en clases, donde los casilleros contadores no son para elementos individuales sino para intervalos o rangos de enteros. Entonces el ordenamiento en la primera etapa es parcial, y debe continuar ordenando enteros restringuidos a intervalos específicos indicados por los contadores. Debe seleccionar que intervalos usar así como el método de ordenamiento de cada uno de ellos; justifique debidamente sus elecciones.

2. Como cuarta opción de ordenamiento usará, desde la std del sistema (std c++11), el método que implementa quickSort.

Debe entregar estadísticas de comparación y buenas conclusiones empíricas de todas las opciones de ordenamiento.

2. Búsqueda

Dado n leído desde los argumentos del programa, genere n enteros aleatorios (nuevamente $\leq$MAX) y almacenelos en 3 estructuras diferentes. Luego ud. medirá el performance en búsqueda de enteros usando cada estructura:

1. Un Arreglo Ordenado. almacenará los n enteros en un arraglo ascendente $A[0..n-1]$.
2. Un Árbol AVL. Implementará la búsqueda y la inserción con los balanceos vistos en clases.
3. Una Tabla Hash. Almacenará los enteros en una tabla hash con una buena función de hashing definida por ud.

Entonces, una vez construida las estructuras, en una etapa experimental ud. medirá los tiempos de búsquedas de cada alternativa. Para el arreglo ordenado implemente un buen método que determine cuantas veces esta un entero en el arreglo.
Los métodos de búsquedas en cada estructura, deberán retornar el número de veces que el entero buscado esta en la estructura; por tanto, las tres esctructuras deben soportar elementos duplicados en ellas.
Entregue estadísticas que promedien varios miles de búsquedas de enteros sobre las estructuras. Informando el tiempo y el promedio de ocurrencias encontradas en sus búsquedas. Experimente con $n = \{10^5, 10^6, 10^7\}$ y con MAX=10^6.