Tarea Programada
Algoritmos de Ordenamiento

Descripción:

Los algoritmos de ordenamiento forman parte del conjunto de algoritmos más estudiados en el campo de la computación debido a su relevancia y simplicidad. Son utilizados diariamente por millones de programas y, aunque hoy en día se encuentran a un "*sort()*" de distancia, es importante comprender su funcionamiento, la diferencias entre los diferentes tipos de ordenamiento que hay, sus ventajas y desventajas.

Enunciado:

En esta tarea el objetivo consiste en implementar los diversos algoritmos de ordenamiento vistos en clase y realizar experimentos de medición con los mismos. Para ello deberán implementar los siguientes algoritmos de ordenamiento: *selection sort*, *insertion sort*, *merge sort*, *heap sort*, *quick sort* y *radix sort*. (+2 puntos extra si incorporan algún otro algoritmo de ordenamiento con motivos de comparación, e.g: *bubble sort*, *gnome sort, etc.*[1])

Además de dichas implementaciones el objetivo será comparar el rendimiento de los algoritmos de ordenamiento en tres diferentes casos: un arreglo aleatorio, un arreglo ordenado ascendentemente y un arreglo ordenado descendentemente. Para cada uno de esos casos será necesario repetir el experimento con arreglos de diversos tamaños: 16384 (2^{14}), 32768 (2^{15}), 65536 (2^{16}), 131072 (2^{17}) y 262144 (2^{18}). (Opcional: puede realizar el experimento con 2^{19} y 2^{20} para tener más datos).

El objetivo de estos arreglos y tamaños será ejecutar los algoritmos de ordenamiento y medir el tiempo que les toma finalizar la ejecución de su ordenamiento. Las mediciones de tiempo se realizarán en segundos, pero su precisión debe incluir milisegundos. Para ello puede utilizar la biblioteca <chrono> que forma parte de las bibliotecas estándar de C++.

Para cada combinación de arreglo y tamaño, deberá repetir el experimento tres veces y guardar los resultados, sin embargo con fines de graficación se utilizará el promedio de las 3 mediciones.

Por lo tanto se tiene un total 6 algoritmos de ordenamiento, 3 tipos de arreglos a ordenar, 5 tamaños de arreglos y 3 mediciones a realizar para cada combinación, para un gran total de: 270 ejecuciones.

Debido a la cantidad de pruebas, se le recomienda automatizar las pruebas: automatizar la generación de los arreglos, automatizar los llamados al ordenamiento de arreglos, automatizar la toma de mediciones (recuerde inicializar el contador de tiempo inmediatamente antes del llamado al algoritmo de ordenamiento y finalizar dicho contador inmediatamente después) y automatizar la verificación de los resultados.

Otra recomendación es que al correr las pruebas no tenga ningún otro programa ejecutándose en la computadora y siempre realizar las pruebas en la misma máquina, esto para evitar "ruido" en los datos.

---

[1] Con excepción de *bogosort, slowsort,* o algún otro de esos "amigos": esos restan puntos más bien.

Con los datos obtenidos debe redactar un entregable escrito en el que presente los resultados de sus experimentos: Tablas con los valores obtenidos, gráficos de comparación entre los resultados de los 6 algoritmos (un gráfico para cada uno de los 3 tipos de arreglo) y debe explicar los resultados observados, así como si el comportamiento fue el que se esperaba.

Evaluación:

- *Insertion Sort, Selection Sort, Merge Sort* (15 pts, 5 pts c/u)
- *Heap Sort, Quick Sort, Radix Sort* (30 pts, 10 pts c/u)
- Documentación interna (5 pts)
- Automatización de pruebas (10 pts)
- Código main (10 pts)
- Documento escrito (30 pts)
    - Debe incluir los gráficos y tablas con los datos obtenidos.
        - Utilice los estándares para numeración de tablas e imágenes.
    - Puede redactarlo en Word o Latex pero debe entregarse como PDF.
    - Las gráficas puede hacerlas con su herramienta favorita (e.g: Excel, Python, Drive, etc.)
    - Si menciona algo proveniente de algún libro o artículo, recuerde citar.
    - No hay tamaño mínimo de entregable, pero póngale amor :)

Se puede trabajar de manera individual, en parejas o en tríos. Pero solo **una** persona por grupo debe enviar el trabajo, de lo contrario perderán 5 puntos por envío duplicado. (Los asistentes siempre agradecen cuando evitan enviar duplicados)

P.D: Recuerde que las funciones son, a fin de cuentas, direcciones de memoria donde se encuentra almacenado código ejecutable. Por lo tanto es posible enviar una función como un parámetro de otra función. Esto puede resultar de utilidad a la hora de automatizar las pruebas. E.g:

```
void f(int* array, int (*g)(int x,int y));
```

Describe una función `f` que recibe por parámetros un puntero de enteros `array` y una función `g` que recibe por parámetros 2 números enteros y retorna un entero. Dentro de `f` podremos utilizar la función `g` de manera normal: `g(2,3)`, sin preocuparnos de cómo está implementada dicha función.