Universidad de Costa Rica
CI-0116: Análisis de Algoritmos y Estructuras de Datos
Laboratorio #2

Objetivo: Familiarizar al estudiante en la implementación de los métodos estudiados: *heap-sort* y colas de prioridad

Enunciado

Resuelva los siguientes problemas. Escriba un código en C++ que resuelva los siguientes problemas.

1. Implemente el algoritmo de ordenamiento *heapsort* visto en clase. Utilícelo para ordenar al menos 3 arreglos de prueba diferentes.

2. Implemente una cola de prioridad haciendo uso de un montículo (*heap*). Puede utilizar una implementación utilizando arreglos o punteros, queda a su decisión.
    a. La cola de prioridad debe usar plantillas (*templates*) para permitir al programador usar una cola de prioridad para cualquier tipo de objeto.
    b. Esto significa: cada nodo posee una prioridad (int) y un objeto T que es lo que el usuario podría ordenar dentro de dicha cola.

Puntos de honor +10

(Los puntos de honor ***no*** son puntos extra, tampoco son evaluados, simplemente son un desafío para los que quieran perfeccionar sus habilidades)

3. Construya un iterador para su cola de prioridad.
    a. Un iterador es una clase anidada que se suele incluir en las estructuras de datos (Estructura::iterator) que permite iterar (recorrer de manera ordenada) los elementos de la clase principal.
    b. La clase principal posee al menos 2 métodos especiales para permitir la utilización de su iterador:
        i. iterator begin(): Este método retorna un objeto de la clase iterador inicializado en el primer elemento de la estructura de datos. Idealmente tiene duración O(1)
        ii. iterator end(): Este método retorna un objeto de la clase iterador con unos valores inválidos que se obtendría si se intentara accesar un elemento después del rango.
    c. La clase anidada debe sobrecargar al menos 2 operadores para su funcionamiento:
        i. El operador !=: Este operador permite comparar un iterador para ver si se encuentra en una posición finalizada
        ii. El operador ++var (diferente a var++): Este operador nos debe permitir avanzar la posición del iterador al siguiente elemento.

De esta manera el código:

```
for( PriorityQueue::iterator it = pq.begin(); it != pq.end(); ++it){
    cout << it->data << endl;
}
```

Imprimiría los diferentes elementos de la cola de prioridad en orden.

Para comprobar funcionamiento correcto.