**Especificación**

En este problema, se implementará una estructura de datos llamada ‘Heap’, la cual es la implementación de un árbol binario de búsqueda en forma de lista, esta implementación se hará en dos versiones:

Max Heap, en donde cada elemento es mayor a sus dos hijos.

Min Heap, en donde cada elemento es menor a sus dos hijos.

Entrada

Una lista en cualquier orden la cual será convertida en un Heap, por medio del método llamado heapify de la clase Heap.

Salida

Una lista en forma de Heap, en la cual se pueden hacer búsquedas en tiempo logarítmico.

**Estrategia**

La clase Heap se basa en una lista, la cual tiene un orden específico para ser interpretada como un árbol de búsqueda, además de una política en donde se decide si es un Max Heap o un Min Heap, para construir el Heap se intercambian elementos en la lista, verificando que cada uno cumpla las propiedades del Heap con respecto a su padre y a sus hijos.

**Componentes**

La clase contiene herramientas para: construir un heap a partir de una lista, agregar elementos al heap, eliminar elementos del heap, actualizar elementos del heap, obtener el heap.

**Pruebas**

La función main contiene dos pruebas, en donde a partir de una lista se genera un Min Heap y un Max Heap, allí se puede verificar que la lista es convertida correctamente en las estructuras de datos mencionadas anteriormente.

**Complejidad temporal**

La complejidad temporal del algoritmo heapify es O(n), es el algoritmo más costoso, por tanto, la complejidad temporal del programa es O(n).

**Complejidad espacial**

El heap requiere una lista de tamaño n, en donde n es el número de elementos, por tanto tenemos una complejidad espacial lineal.