Act 3.2 - Árbol Heap: Implementando una fila priorizada

Un **HEAP** es un árbol binario que cumple con las siguientes reglas:

- Todo nodo padre del árbol contiene un elemento que tiene un valor de mayor prioridad que los valores de sus nodos hijos. La prioridad está determinada por el programa.
- El árbol debe de estar completamente balanceado, todos los niveles del árbol tienen el máximo de dos nodos posible, excepto el nivel inferior que puede estar incompleto.
- Los nodos hijo del nivel inferior están lo mas a la izquierda posible.

Tomando en cuenta esto, se desarrollo una ADT que fuera capaz de interactuar con este tipo de estructura de datos, las funciones y métodos fueron especificados por el profesor en clase, los cuales son:

1. push

Esta función inserta un numero en su orden de prioridad, buscando elemento por elemento si su lugar es ese.

Test case 1

```
Ingrese un numero: 8
La cola era:
10 9 7 5 2 1

La cola es:
10 9 8 7 5 2 1
Press any key to continue . . .
```

Test case 2

Cola vacía

```
Ingrese un numero: 6
La cola era:
La cola esta vacia

La cola es:
6
Press any key to continue . . .
```

Test case 3

```
Ingrese un numero: 10
La cola era:
10 9 8 7 6 5 4 3 2 1

La cola es:
10 10 9 8 7 6 5 4 3 2 1
Press any key to continue . . .
```

Test case 4

```
Ingrese un numero: 20
La cola era:
28 21 17 19 5 8

La cola es:
28 21 20 19 5 8 17
Press any key to continue . . .
```

2. pop

Esta función elimina el elemento de mayor prioridad en la lista.

Test case 1

```
La cola era:
10 9 8 7 5 2 1
El elemento eliminado es: 10

La cola es:
9 8 7 5 2 1
Press any key to continue . . .
```

Test case 2

Cola vacía

```
La cola era:
La cola esta vacia
El elemento eliminado es: La cola esta vacia
-1

La cola es:
La cola esta vacia
Press any key to continue . . .
```

Test case 3

```
La cola era:
10 10 9 8 7 6 5 4 3 2 1
El elemento eliminado es: 10

La cola es:
10 9 8 7 6 5 4 3 2 1
Press any key to continue . . .
```

Test case 4

```
La cola era:
28 21 20 19 5 8 17
El elemento eliminado es: 28

La cola es:
21 20 19 5 8 17
Press any key to continue . . .
```

3. top

Esta función regresa el elemento de mayor prioridad en la lista.

Test case 1

```
La cola es:
9 8 7 5 2 1
El elemento del frente es: 9
Press any key to continue . . .
```

Test case 2

Cola vacía

```
La cola es:
La cola esta vacia
El elemento del frente es: La cola esta vacia
-1
Press any key to continue . . .
```

Test case 3

```
La cola es:
10 9 8 7 6 5 4 3 2 1
El elemento del frente es: 10
Press any key to continue . . .
```

Test case 4

```
La cola es:
21 20 19 5 8 17
El elemento del frente es: 21
Press any key to continue . . .
```

4. empty

Esta función devuelve un booleano indicando si la lista está vacía o no.

Test case 1

Fila: 9 8 7 5 2 1

```
La fila no esta vacia
Press any key to continue . . .
```

Test case 2

Fila:

```
La fila esta vacia
Press any key to continue . . .
```

5. size

Esta función devuelve el tamaño de la lista.

Test case 1

```
La cola es:
9 8 7 5 2 1
El tamano de la fila es: 6
Press any key to continue . . .
```

Test case 2

Cola vacía

```
La cola es:
La cola esta vacia
El tamano de la cola es: 0
Press any key to continue . . .
```

Test case 3

```
La cola es:
10 9 8 7 6 5 4 3 2 1
El tamano de la cola es: 10
Press any key to continue . . .
```

Test case 4

```
La cola es:
21 20 19 5 8 17
El tamano de la fila es: 6
Press any key to continue . . .
```

Complejidad.

Push: Tiene una complejidad de O(n) ya que compara los elementos de la lista.

Pop: Elimina el elemento de mayor prioridad, tiene complejidad de O(1)

Top: Muestra el elemento top, por lo que tiene complejidad de O(1)

isEmpty: Devuelve un booleano, tiene una complejidad de O(1)

Size: Complejidad de O(1) ya que devuelve el tamaño sin recorrer la lista de prioridad.