

## My Project

Generated by Doxygen 1.8.11



# Contents



# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

|                                       |       |    |
|---------------------------------------|-------|----|
| <a href="#">BinaryHeap&lt; T &gt;</a> | ..... | ?? |
|---------------------------------------|-------|----|



## Chapter 2

# Class Documentation

### 2.1 BinaryHeap< T > Class Template Reference

#### Public Member Functions

- [BinaryHeap](#) (const vector< T > &items)
- bool [IsEmpty](#) () const
- const T & [FindMin](#) () const
- void [Insert](#) (const T &newItem)
- void [DeleteMin](#) ()
- void [DisplayLinear](#) (ofstream &output)
- void [RemoveInOrder](#) (ofstream &output)

#### Private Member Functions

- void [BuildHeap](#) ()
- void [PercolateDown](#) (int holeIndex)

#### Private Attributes

- unsigned int **m\_size**
- vector< T > **m\_array**

#### 2.1.1 Constructor & Destructor Documentation

**2.1.1.1** `template<typename T> BinaryHeap< T >::BinaryHeap ( const vector< T > & items )`  
`[inline]`

=1mm

spread Opt [!]|X[-1,!]|X[-1,!]|Parameters

---

Parameters

---

*items* <const vector<T>&> the initial items to put in the heap.

---

First, copy the items from the items vector into the m\_arrays vector. However, we do not put anything at position 0 in m\_array, so make sure you're storing items in i+1 for m\_array, when it is item i from the items vector. Then, call the BuildHeap function.

Returns

Parameters

Parameters

---

*newItem* <const T&> The new item to insert

---

If *m\_size* is one less than the size of the *m\_array*, then tell *m\_array* to resize to double its current size (use the vector's *resize()* function.)

Afterward:

1. Create an integer called *hole*, and assign it to the current value of *m\_size*.
2. Increment *m\_size* by one
3. Create a loop. It will continue looping WHILE *hole* > 1 && *newItem* < *m\_array*[ *hole* / 2 ] And at the end of each cycle, it should EXECUTE *hole* /= 2
  - 3a. Within the loop, set the element of *m\_array* at position *hole* to the value of the element of *m\_array* at position *hole*/2.
4. After the loop, set the element of *m\_array* at position *hole* equal to the *newItem*.



### 2.1.2.5 `template<typename T> bool BinaryHeap< T >::IsEmpty ( ) const [inline]`

#### Returns

bool true if m\_array is empty, or false otherwise.

### 2.1.2.6 `template<typename T> void BinaryHeap< T >::PercolateDown ( int holeIndex ) [inline], [private]`

=1mm

spread Opt [I]|X[-1,I]|X[-1,I]]Parameters

---

#### Parameters

---

*holeIndex* <int> The position in the heap to percolate down.

---

1. Create an integer called child
2. Create a temp T item, and assign it to the value of the element of m\_array at the holeIndex.
3. Loop while holeIndex \* 2 <= m\_size:
  - 3a. Set child to holeIndex \* 2.
  - 3b. If the child is not m\_size, and the element of m\_array at child+1 is less than the element of m\_array at child: Increment child by 1.
  - 3c. If the element of m\_array at child is less than the temp value: Set the element of m\_array at position holeIndex to the element of m\_array at position child. OTHERWISE break.
  - 3d. At the end of the loop cycle, set holeIndex to the child.
4. Set the element of m\_array at position holeIndex to the temp value.

The documentation for this class was generated from the following file:

- BinaryHeap.hpp

