# Lab02实验报告

班级：软外1903 姓名：秦政瀚 学号：201906150312

## Task 1

### Answer:

The relationship between child index with parent index in d-heaps is:

// i is the index of child element  
childIndexArray[i] = parentIndex \* numOfChild + i + 1;

So that in d-heap for d=4 is:

childIndexArray[i] = parentIndex \* 4 + i + 1;

## Task 2a

### Theory:

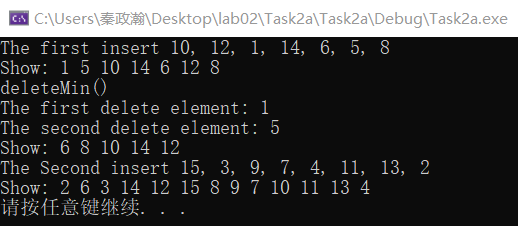
* Using a data structure of tree to implement the d-heaps for d=4.

### Function:

* void insert(int element) that insert the new element to the d-heaps for d=4.
  + If the size of the quadHeap equals to the capacity before inserting, throw error of "The quadHeap is full!".
  + If the size equals to zero before inserting, insert the new element directly.
  + Using while-loop to percolate up the new element until it finds the correct location and by return void to end the loop, it is help compare the child index and the parent index and exchange the position if necessary.
* int deleteMin() that delete the minimum element of the d-heaps for d=4.
  + If the size of the quadHeap equals to zero before deleting, throw error of "The quadHeap is empty!".
  + In oreder to maintain the character of the d-heap, I use for-loop to percolate down the elements which affected by the operation of deleting the minimum element of the d-heaps.
* bool isEmpty() that returns true if the h-heaps for d=4 is empty and returns false otherwise.
* void show() that output the quadHeap completely.

## Task 2b

### The screenshot of output:



## Task 3

### Theory:

* The implement of Task 3 is similar to Task 2a, just change the "4" in childIndex=parentIndex\*4+1 and parentIndex=(curIndex-1)/4 to "n" which represents arbitrary number.

### Comception:

* For the basic operation of the binary heap, the insert operationneeds to locate the parent node, which requires a division operation, the number of operations related to the height of the tree. The deleteMin operation needs to find the son with the smallest weight of all the sons, while the search for the son node requires a multiplication operation, the complexity of which is related to the number of sons (the larger the number of sons, the more sons the node has, the slower the lookup).
* Assuming that there are a lot of insert operations and only a small amount of deleteMin, the d-heap theoretically has a performance advantage. Because d is much larger than 2, the height of the tree is small, but when d is not a multiple of 2, division operation cannot be achieved by shifting, there may be a certain performance loss. And if there's a lot of deleteMin operations, the d-heap might be less than low-performance because: the larger the d, the more sons the node has, and the more comparisons it takes to find the son with the least weight.

### Function:

* The same to the Task 2a.