```java
public Heaps() {
    System.out.println("Minumum Heap is creating ");
    heap = new ArrayList<Integer>();
}

/**
 * methods for start with one element
 * @param data is the object
 */
public Heaps(Integer data) {
    System.out.println("Minumum Heap is creating ");
    System.out.println("element " + data + " is added");
    heap = new ArrayList<Integer>();
    heap.add(data);
}

/**
 * if came full list
 * @param new_array is new list
 */
public Heaps(ArrayList<Integer> new_array) { heap = new_array; }
```

$T(n) = \Theta(1)$

$T(n) = \Theta(1)$

$T(n) = \Theta(1)$

```java
/**
 * Method for searching element
 * @param element is search element
 * @return location of element
 */
public int Search(E element) {
    if (heap.isEmpty()) {
        System.out.println("Element " + element + " is not in tree");
        return -1;
    }
    return heap.indexOf(element);
}
```

$T(n) = \Theta(1)$

```java
public void delete() {
    if ( heap.isEmpty() ) {
        throw new NoSuchElementException();
    }
    else {
        System.out.println("Element " + heap.get(0) + " is deleted");
        int parent = heap.get(heap.size()-1);
        heap.remove( index: heap.size()-1);
        heap.set(0,parent);
        recursive_delete( parent: 0);
    }
}

/**
 * if list is not empty , need recursive deleted
 * @param parent is check with child
 */
private void recursive_delete(int parent) {
    try {
        if ( (parent*2)+1 <= heap.size()-1 ) {
            int left = heap.get( (parent*2)+1 );
            int right = left + 1;
            if ((parent*2)+2 <= heap.size()-1 ) {
                right = heap.get( (parent*2)+2 );
            }
            if (heap.get(parent) > left && left <= right) {
                int temp = heap.get(parent);
                heap.set(parent , heap.get((2*parent)+1));
                heap.set((2*parent)+1 , temp);
                recursive_delete( parent: (2*parent)+1);
            }
            else if (heap.get(parent) > right && left > right) {
                int temp = heap.get(parent);
                heap.set(parent , heap.get((2*parent)+2));
                heap.set((2*parent)+2 , temp);
                recursive_delete( parent: (2*parent)+2);
            }
        }
    }catch (Exception e) {
        /* System null */
    }
}
```

$\Theta(1)$

$T(n) = \Theta(\log n)$

$\Theta(1)$

$\Theta(\log n)$

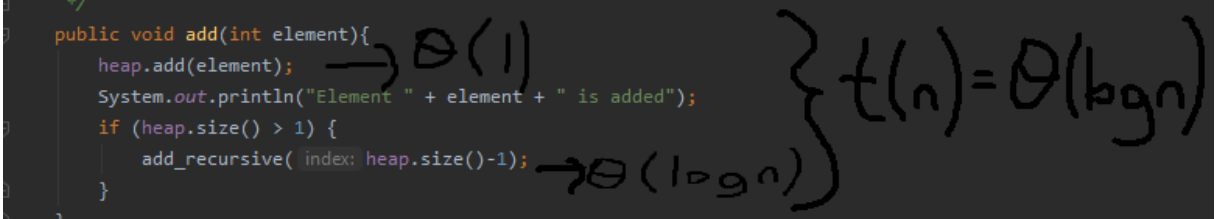
$T(n) = \Theta(\log n)$

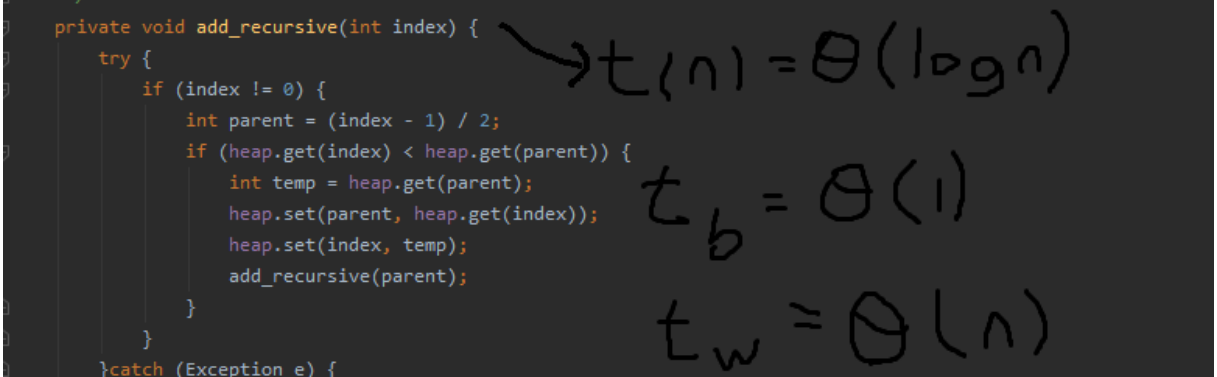
$t_b = \Theta(1)$

$t_w = \Theta(n)$

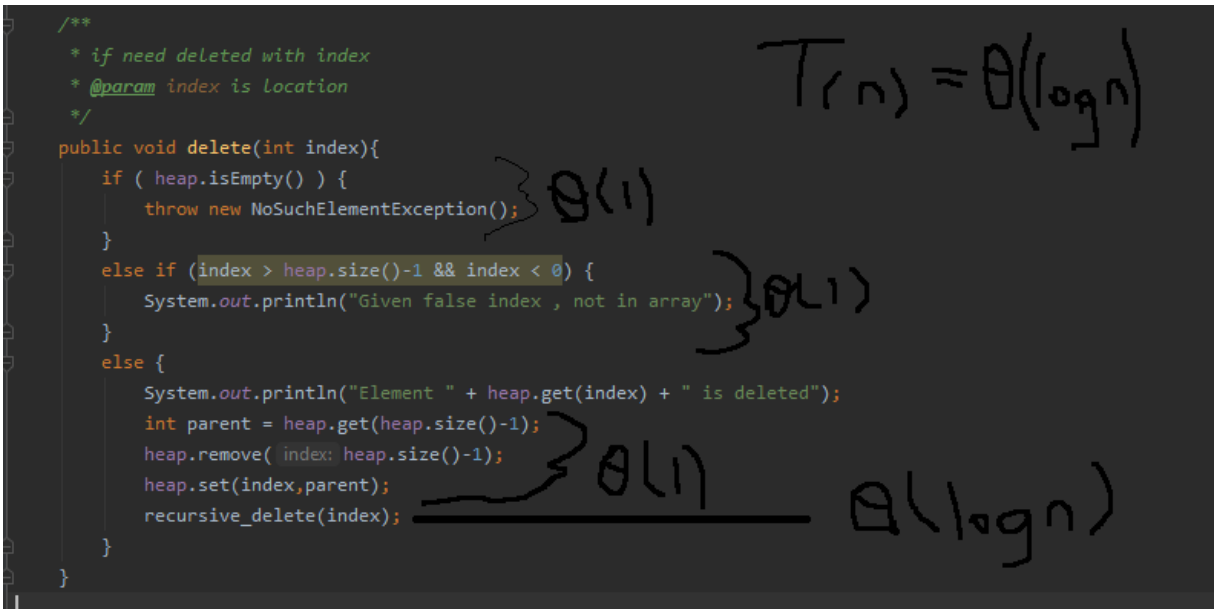
```java
/**
 * method add
 * @param element is added data
 */
public void add(int element){
    heap.add(element);
    System.out.println("Element " + element + " is added");
    if (heap.size() > 1) {
        add_recursive( index: heap.size()-1);
    }
}

/**
 * if list is not empty , need recursive added
 * @param index is check for minumum heaps
 */
private void add_recursive(int index) {
    try {
        if (index != 0) {
            int parent = (index - 1) / 2;
            if (heap.get(index) < heap.get(parent)) {
                int temp = heap.get(parent);
                heap.set(parent, heap.get(index));
                heap.set(index, temp);
                add_recursive(parent);
            }
        }
    }catch (Exception e) {
        /* System null */
    }
}
```






```java
/**
 * if need deleted with index
 * @param index is location
 */
public void delete(int index){
    if ( heap.isEmpty() ) {
        throw new NoSuchElementException();
    }
    else if (index > heap.size()-1 && index < 0) {
        System.out.println("Given false index , not in array");
    }
    else {
        System.out.println("Element " + heap.get(index) + " is deleted");
        int parent = heap.get(heap.size()-1);
        heap.remove( index: heap.size()-1);
        heap.set(index,parent);
        recursive_delete(index);
    }
}
```



```java
/**
 * check for new heap
 * @return the boolean
 */
private boolean empty() {
    if (heap.size() > 0) {
        return true;
    }
    return false;
}

/**
 * method for remove arraylist
 * @return removed element
 */
private int element_remove() { return heap.remove( index: heap.size()-1); }

/**
 * Merge with other Heap
 * @param heaps is other heap
 */
public void Merge_heap (Heaps<E> heaps) {
    while (heaps.empty() ){
        int element = heaps.element_remove();
        this.heap.add(element);
    }
}

/**
 * is show all element index by index
 */
public void to_String() {
    if (heap.isEmpty()) {
        throw new NoSuchElementException();
    }
    int i;
    System.out.print("All element : ");
    for (i = 0; i < heap.size() ; i++ ) {
        System.out.print(heap.get(i) + " ");
    }
    System.out.print("\n");
}
```