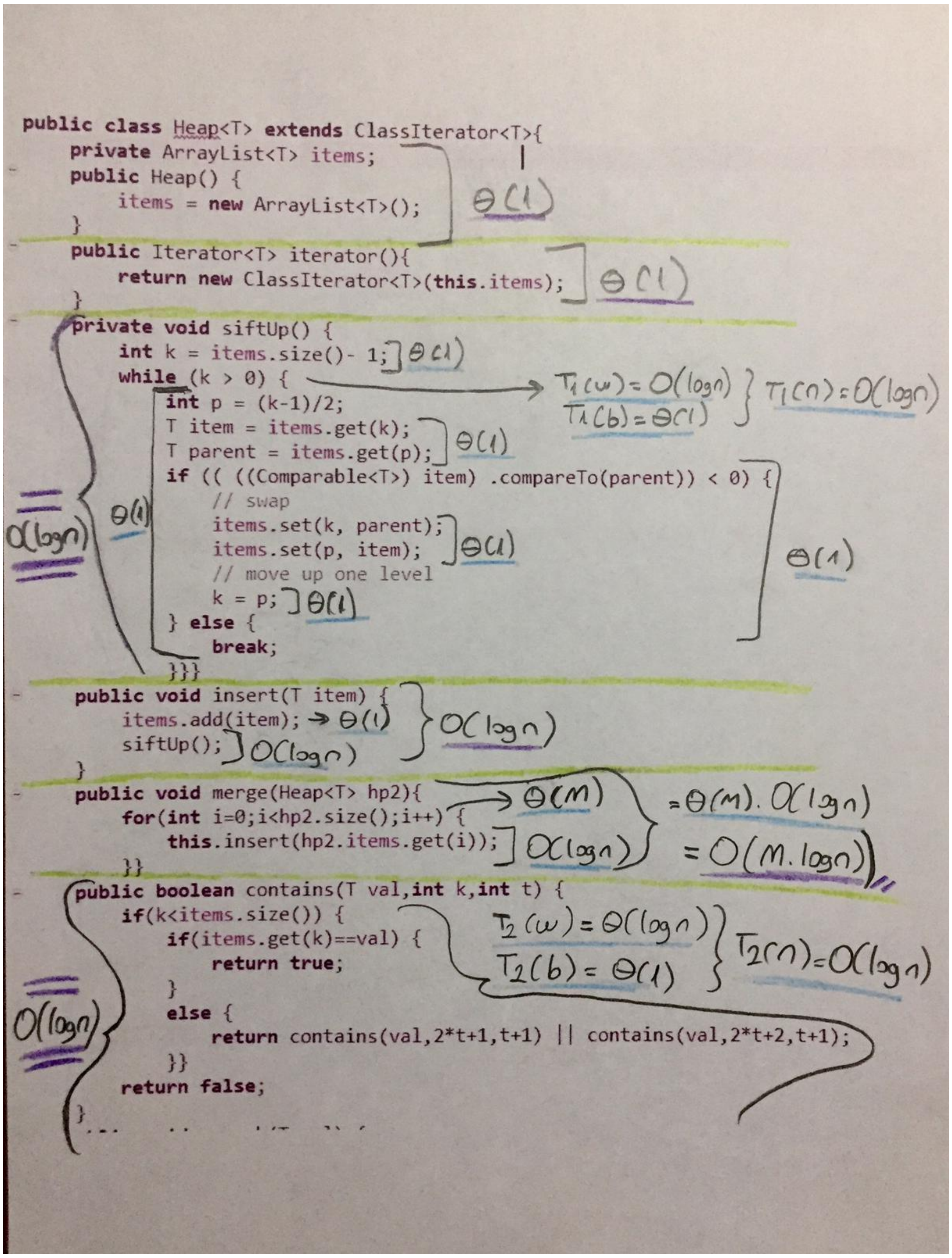
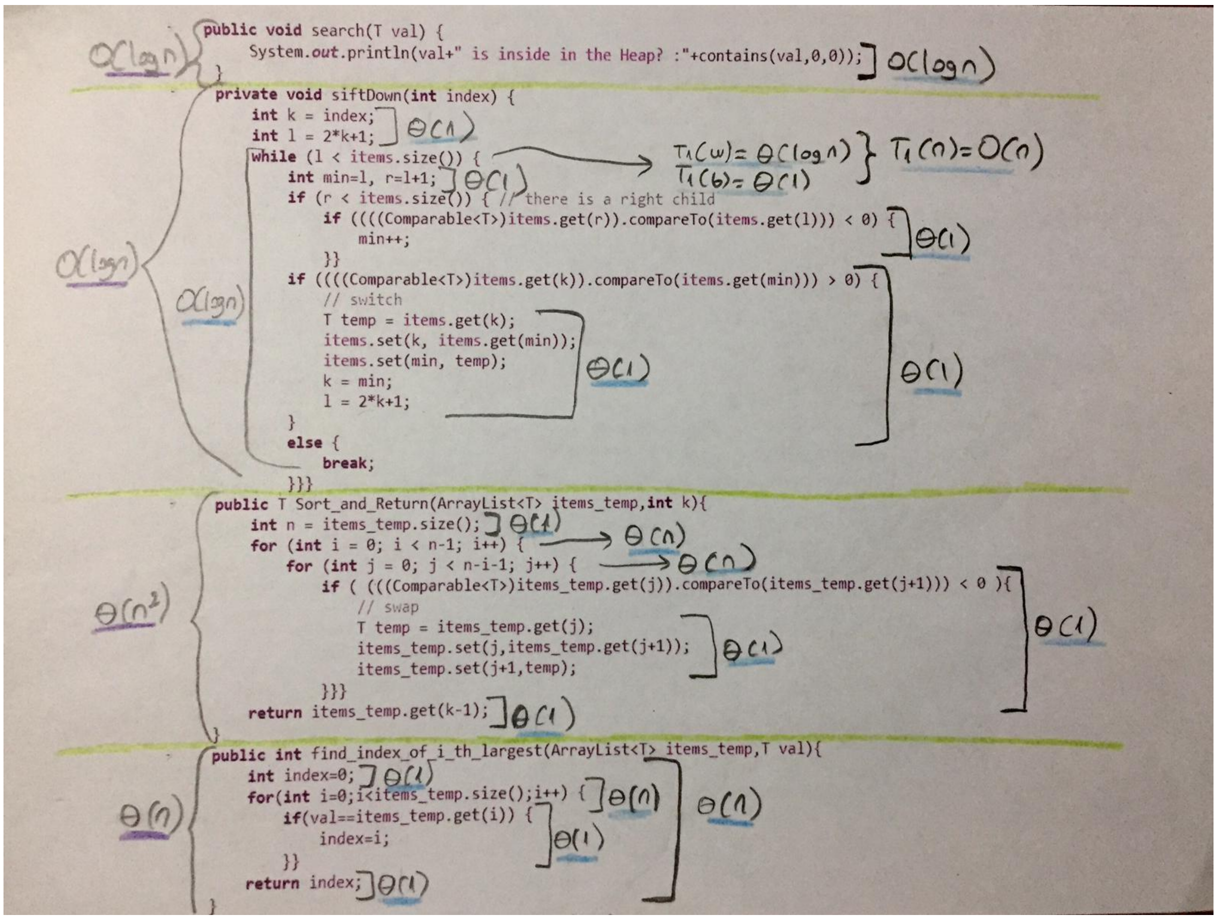
Time Complexity

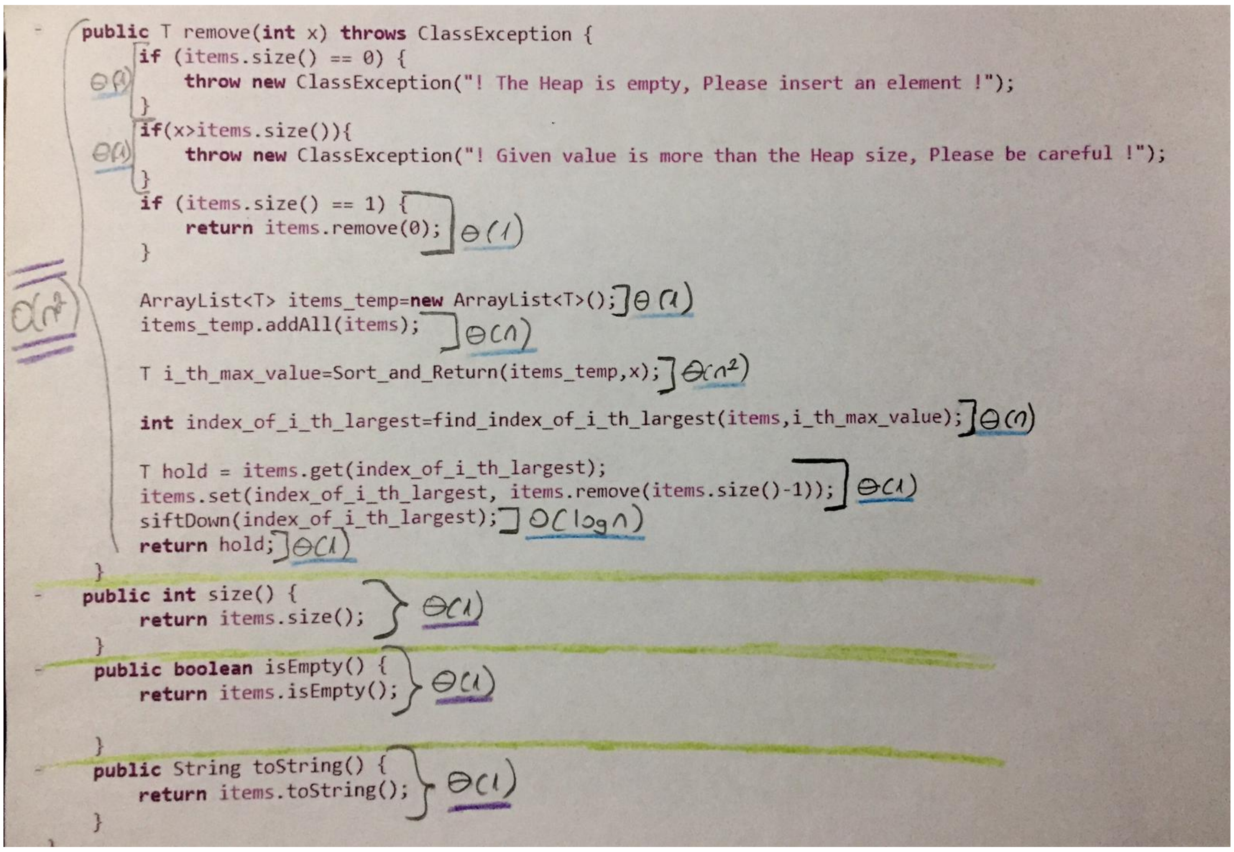
Min Heap Class



```
public class Heap<T> extends ClassIterator<T>{
    private ArrayList<T> items;
    public Heap() {
        items = new ArrayList<T>();
    }
    public Iterator<T> iterator(){
        return new ClassIterator<T>(this.items);
    }
    private void siftUp() {
        int k = items.size()- 1;
        while (k > 0) {
            int p = (k-1)/2;
            T item = items.get(k);
            T parent = items.get(p);
            if (( ((Comparable<T>) item) .compareTo(parent)) < 0) {
                // swap
                items.set(k, parent);
                items.set(p, item);
                // move up one level
                k = p;
            } else {
                break;
            }}}
    public void insert(T item) {
        items.add(item);
        siftUp();
    }
    public void merge(Heap<T> hp2){
        for(int i=0;i<hp2.size();i++) {
            this.insert(hp2.items.get(i));
        }}
    public boolean contains(T val,int k,int t) {
        if(k<items.size()) {
            if(items.get(k)==val) {
                return true;
            }
            else {
                return contains(val,2*t+1,t+1) || contains(val,2*t+2,t+1);
            }}
        return false;
    }
```

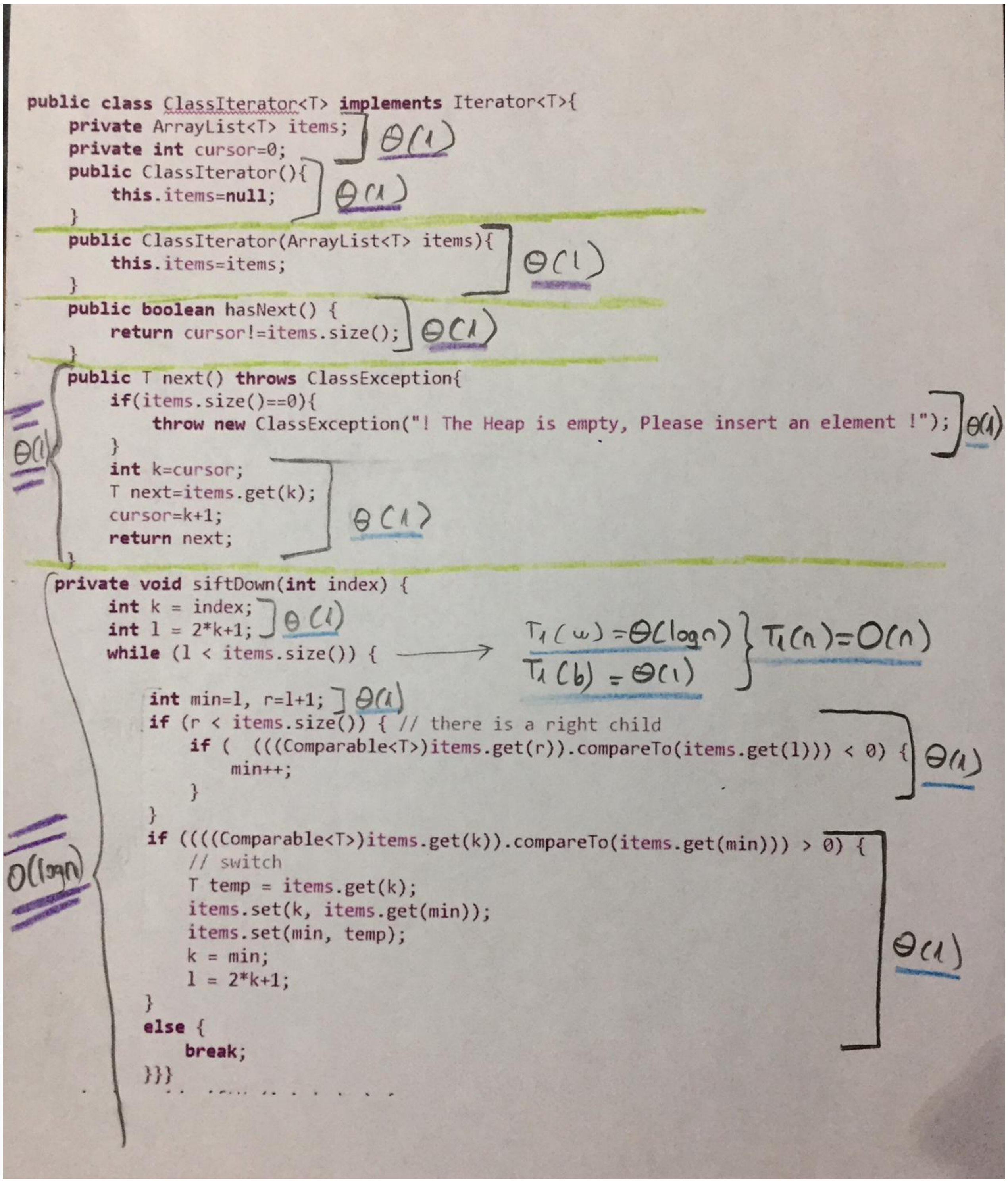


```
O(logn)  public void search(T val) {
             System.out.println(val+" is inside in the Heap? :"+contains(val,0,0)); ] O(logn)
         }
         private void siftDown(int index) {
             int k = index;   ] Θ(1)
             int l = 2*k+1;   ]
             while (l < items.size()) {   ----> T1(w)= Θ(logn) } T1(n)=O(n)
                                                T1(b)= Θ(1)
                 int min=l, r=l+1; ] Θ(1)
                 if (r < items.size()) { // there is a right child
                     if ((((Comparable<T>)items.get(r)).compareTo(items.get(l))) < 0) { ] Θ(1)
                         min++;
                     }}
O(logn)   O(logn) if ((((Comparable<T>)items.get(k)).compareTo(items.get(min))) > 0) {  ] Θ(1)
                     // switch
                     T temp = items.get(k);              ]
                     items.set(k, items.get(min));       ] Θ(1)
                     items.set(min, temp);               ]
                     k = min;                            ]
                     l = 2*k+1;                          ]
                 }
                 else {
                     break;
                 }}}
         public T Sort_and_Return(ArrayList<T> items_temp,int k){
             int n = items_temp.size(); ] Θ(1)
             for (int i = 0; i < n-1; i++) {  ----> Θ(n)
                 for (int j = 0; j < n-i-1; j++) {  ----> Θ(n)
Θ(n²)                if ( (((Comparable<T>)items_temp.get(j)).compareTo(items_temp.get(j+1))) < 0 ){ ] Θ(1)
                         // swap
                         T temp = items_temp.get(j);                   ]
                         items_temp.set(j,items_temp.get(j+1));        ] Θ(1)
                         items_temp.set(j+1,temp);                     ]
                     }}}
             return items_temp.get(k-1); ] Θ(1)
         }
         public int find_index_of_i_th_largest(ArrayList<T> items_temp,T val){
             int index=0; ] Θ(1)
             for(int i=0;i<items_temp.size();i++) { ] Θ(n)   ] Θ(n)
Θ(n)             if(val==items_temp.get(i)) {  ] Θ(1)
                     index=i;
                 }}
             return index; ] Θ(1)
         }
```

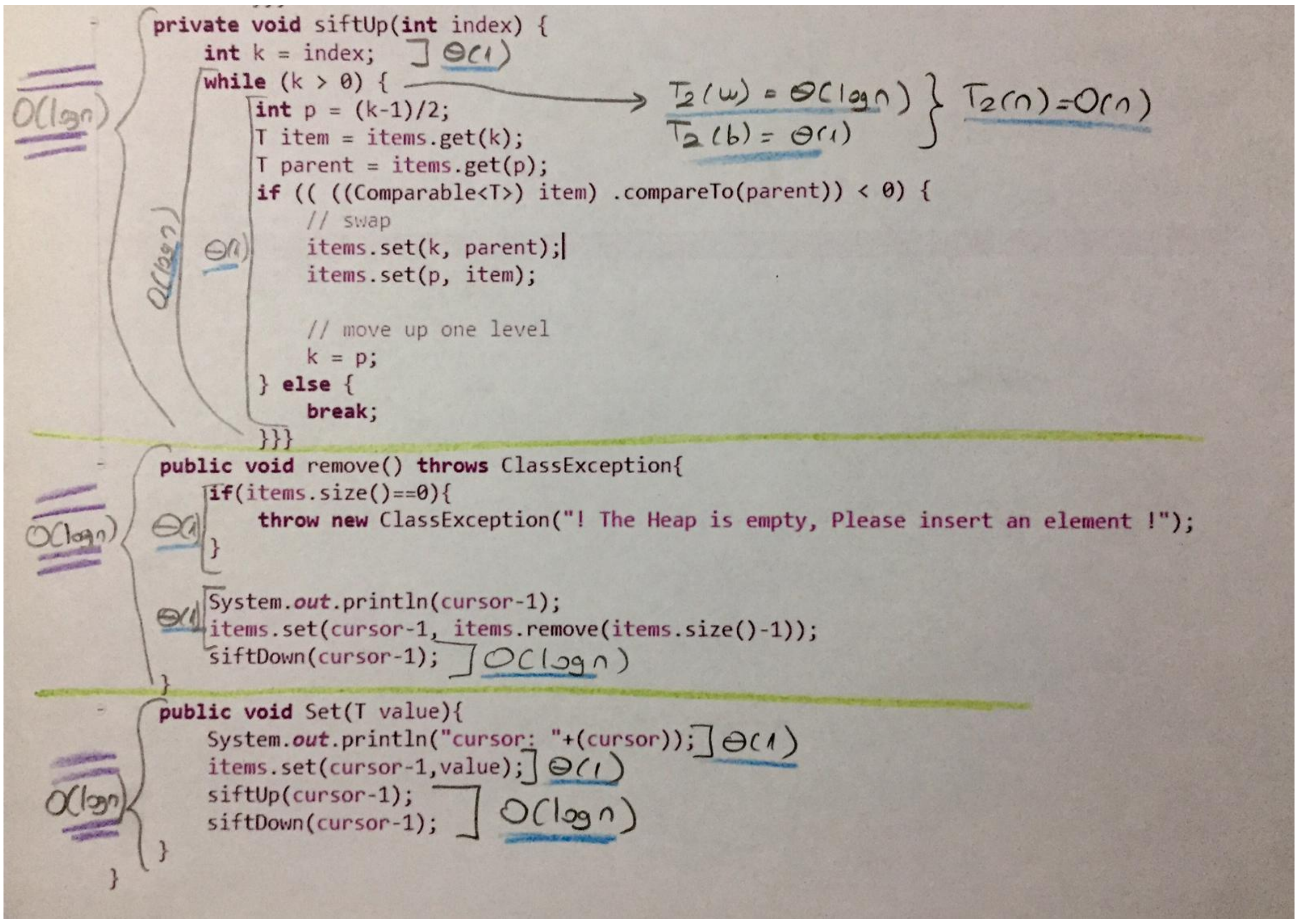


```
       public T remove(int x) throws ClassException {
           if (items.size() == 0) {
Θ(1)           throw new ClassException("! The Heap is empty, Please insert an element !");
           }
           if(x>items.size()){
Θ(1)           throw new ClassException("! Given value is more than the Heap size, Please be careful !");
           }
           if (items.size() == 1) {  ]
               return items.remove(0); ] Θ(1)
           }

O(n²)      ArrayList<T> items_temp=new ArrayList<T>(); ] Θ(1)
           items_temp.addAll(items); ] Θ(n)

           T i_th_max_value=Sort_and_Return(items_temp,x); ] Θ(n²)

           int index_of_i_th_largest=find_index_of_i_th_largest(items,i_th_max_value); ] Θ(n)

           T hold = items.get(index_of_i_th_largest);                          ]
           items.set(index_of_i_th_largest, items.remove(items.size()-1));    ] Θ(1)
           siftDown(index_of_i_th_largest); ] O(logn)
           return hold; ] Θ(1)
       }
       public int size() {
           return items.size();      } Θ(1)
       }
       public boolean isEmpty() {
           return items.isEmpty();   } Θ(1)

       }
       public String toString() {
           return items.toString();  } Θ(1)
       }
```

Iterator Class



```
public class ClassIterator<T> implements Iterator<T>{
    private ArrayList<T> items;
    private int cursor=0;
    public ClassIterator(){
        this.items=null;
    }
    public ClassIterator(ArrayList<T> items){
        this.items=items;
    }
    public boolean hasNext() {
        return cursor!=items.size();
    }
    public T next() throws ClassException{
        if(items.size()==0){
            throw new ClassException("! The Heap is empty, Please insert an element !");
        }
        int k=cursor;
        T next=items.get(k);
        cursor=k+1;
        return next;
    }
   private void siftDown(int index) {
        int k = index;
        int l = 2*k+1;
        while (l < items.size()) {

          int min=l, r=l+1;
          if (r < items.size()) { // there is a right child
              if (   (((Comparable<T>)items.get(r)).compareTo(items.get(l))) < 0) {
                 min++;
              }
          }
          if ((((Comparable<T>)items.get(k)).compareTo(items.get(min))) > 0) {
              // switch
              T temp = items.get(k);
              items.set(k, items.get(min));
              items.set(min, temp);
              k = min;
              l = 2*k+1;
          }
          else {
              break;
          }}}
```



```
        private void siftUp(int index) {
            int k = index;  ] Θ(1)
            while (k > 0) {  ——→  T2(w) = Θ(log n) } T2(n)=O(n)
                                  T2(b) = Θ(1)
                int p = (k-1)/2;
                T item = items.get(k);
                T parent = items.get(p);
O(log n)        if (( ((Comparable<T>) item) .compareTo(parent)) < 0) {
                    // swap
      O(log n) Θ(1) items.set(k, parent);
                    items.set(p, item);

                    // move up one level
                    k = p;
                } else {
                    break;
                }}}
        public void remove() throws ClassException{
            if(items.size()==0){
O(log n)   Θ(1)     throw new ClassException("! The Heap is empty, Please insert an element !");
            }

            System.out.println(cursor-1);
       Θ(1) items.set(cursor-1, items.remove(items.size()-1));
            siftDown(cursor-1); ] O(log n)
        }
        public void Set(T value){
            System.out.println("cursor: "+(cursor));] Θ(1)
            items.set(cursor-1,value);] Θ(1)
O(log n)    siftUp(cursor-1);
            siftDown(cursor-1);  ] O(log n)
        }
    }
```