MyHeap Methods

```
private void swap(int index1,int index2){
    E temp=heap.get(index1);
    heap.set(index1, heap.get(index2));
    heap.set(index2, temp);
}
```

Θ(1)
Θ(1) Total: Θ(1)
Θ(1)

```
protected void sortHeap() {
    for (int i = heap.size()-1; i>=0; i--)
        if (heap.get(i).compareTo(heap.get((i-1)/2))>0)
            swap(i, (i-1)/2);
}
```

Θ(n)
Θ(1) Total:Θ(n)
Θ(1)

```
private E min(){
    E temp=heap.get(0);
    for (int i = 0; i < heap.size(); i++)
        if (temp.compareTo(heap.get(i))>0)
            temp=heap.get(i);

    return temp;
}
```

Θ(1)
Θ(n)
Θ(1) Total: Θ(n)
Θ(1)

```
public E poll() {
    if( heap.size()== 0)
        return null;
    E temp = heap.get(0);
    swap(0, heap.size()-1);
    heap.remove(heap.size()-1);
    sortHeap();
    return temp;
}
```

Θ(1)
Θ(1)
Θ(1)//Last element remove, no shifting
Θ(1)
Total: Θ(1)

```
public Boolean contains(E e) {
    for (int i = 0; i < heap.size(); i++)
        if(heap.get(i).compareTo(e)==0)
            return true;
    return false;
}
```

Θ(n)
Θ(1)
Max: O(n)
Min: Ω(1)//e == heap.get(0)
T(n)=Θ(n)

```
public boolean add(E e) {
    heap.add(e);
    sortHeap();
    return true;
}
```

Θ(1)
Θ(n) Total: Θ(n)

```
public E peek() {
    if( heap.size() == 0 )
        throw new NoSuchElementException();
    return heap.get(0);
}
```

Total: Θ(1)

```
public E findElement(E e) {
    for (int i = 0; i < heap.size(); i++)
        if(heap.get(i).compareTo(e)==0)
            return heap.get(i);
    return null;
}
```

Θ(n)
Θ(1)
Θ(1)

Total:Θ(n)

```
public E remove() {
    if( heap.size() == 0 )
        throw new NoSuchElementException();

    return poll();
}
```

Θ(1)

Total:Θ(1)

Θ(1)

```
public Boolean mergeHeap(MyHeap<E> other) {
    if (other== null || other.size()==0)
        return false;

    MyHeap<E> temp = other;
    while (temp.size()!=0)
        this.add(temp.poll());

    return true;
}
```

Θ(1)

Θ(n)
Θ(k)

Total:Θ(nk)

```
public String toString(){
    StringBuilder str = new StringBuilder();
    for (int i = 0; i < heap.size(); i++)
        str.append(heap.get(i)+" ");
    return str.toString();
}
```

Θ(n)
Θ(1)
Θ(1)

Total: Θ(n)

```
public E removeNthLargest(int n) {
    if (n<0 || n> heap.size())
        throw new IndexOutOfBoundsException();
    if (n==1)
        return heap.remove(0);

    E max=heap.get(0) ,temp=min();
    E max2= temp;
    int k=0;//location of element
    for (int i = 0; i < n-1; i++){
        max2=temp;
        for (int j = 0; j < heap.size(); j++)
            if(heap.get(j).compareTo(max)<0 && max2.compareTo(heap.get(j))<0){
                max2=heap.get(j); k=j;
            }
        max=max2;
    }
    swap(k, heap.size()-1);
    heap.remove(heap.size()-1);
    sortHeap();
    return max2;
}
```

Θ(1)
Θ(k)//shifting
Θ(k)
Θ(n)
Θ(k)
Θ(1)
Θ(1)
Θ(1)
Θ(k)
Total:Θ(nk)

HeapIter Methods

```
public boolean hasNext(){
    if (iterSize != location){
        return true;
    }else
        return false;
}
```

Total:Θ(1)

```
public E next(){
    if(!hasNext())
        throw new NoSuchElementException();

    last++;
    return  heap.get(location++);
}
```

Total: Θ(1)

```
public E set(E item) {
    if(last < 0 )
        throw new IndexOutOfBoundsException();

    E temp = heap.get(last);
    heap.set(last,item);
    sortHeap();
    return temp;
}
```

Θ(1)
Θ(1)
Θ(n)

Total:Θ(n)

Data Methods

```
public String toString() {
    StringBuilder str=new StringBuilder();
    str.append(data).append(",").append(count);
    return str.toString();
}
```

Θ(1)
Total:Θ(1)

*Others are get/set methods or increase/decrease methods so Θ(1)

BSTHeapTree Methods

```
private Helper<E> addToHeap(Data<E> data,
                            BinarySearchTree.Node<MyHeap<Data<E>>> root){
    Helper<E> helpMe=new Helper<E>(null,-1);
    int count=1;
    if (root==null){
        root=new BinarySearchTree.Node<>(new MyHeap<>(data));
        helpMe=new Helper<E>(root,count);
        return helpMe;
    }
    if (root.data.size() <MAX_SIZE ) {
        if(root.data.contains(data)){
            count=root.data.findElement(data).increaseCount();
        }else
            root.data.add(data);
        helpMe=new Helper<E>(root,count);
    }else if (root.data.contains(data)){
        count=root.data.findElement(data).increaseCount();
        helpMe=new Helper<E>(root,count);
    }else if (data.compareTo(root.data.peek())<0) {
        helpMe=addToHeap(data, root.left);
        root.left=helpMe.node;
        helpMe=new Helper<E>(root,helpMe.count);
    }else {
        helpMe=addToHeap(data, root.right);
        root.right=helpMe.node;
        helpMe=new Helper<E>(root,helpMe.count);
    }
    return helpMe;
}
```

Θ(1)
Θ(n)
Θ(n)
Θ(n)
Θ(n)
Θ(n)
Θ(1)
T(h-1)
T(h-1)

T(h,n)=T(h-1,n)+Θ(n) ⇒ h.Θ(n)⇒T(h,n)= O(hn)

```
public int add(E data) {
    Helper<E> helpMe=new Helper<E>(null,-1);
    helpMe=addToHeap(new Data<E>(data),tree.root);
    tree.root=helpMe.node;
    return helpMe.count;
}
```

Total: Θ(hn)

```java
private Data<E> getLastPoll(
        BinarySearchTree.Node<MyHeap<Data<E>>> root) {
    if (root.left != null)
        return getLastPoll(root.left);

    if (root.right != null)
        return getLastPoll(root.right);

    Data<E> temp;
    if (root.data.size()==1) {
        temp =root.data.peek();
        tree.remove(root.data);
        return temp;
    }
    temp =root.data.poll();
    return temp;
}
```

T(h-1)

T(h-1)

T(h,n)=T(h-1)+Θ(n)+O(h)
⇒T(h,n)=O(h.log(n))

O(h)

Θ(n)

```java
private Helper<E> removeFromHeap(Data<E> data,
                        BinarySearchTree.Node<MyHeap<Data<E>>> root){
    Helper<E> helpMe=new Helper<E>(null,-1);
    int count=-1;
    if (root==null){
        helpMe=new Helper<E>(root,count);
        return helpMe;
    }
    if (root.data.contains(data)){
        Data<E> temp=root.data.findElement(data);
        count=temp.decreaseCount();
        if (count==0) {
            if (root.data.size()==1) {
                tree.remove(root.data);
                return new Helper<E>(null,count);
            }else
                temp.changeData(getLastPoll(root));
            root.data.sortHeap();
        }
        helpMe=new Helper<E>(root,count);
    }else if (data.compareTo(root.data.peek())<0) {
        helpMe=removeFromHeap(data, root.left);
        root.left=helpMe.node;
        helpMe=new Helper<E>(root,helpMe.count);
    }else {
        helpMe=removeFromHeap(data, root.right);
        root.right=helpMe.node;
        helpMe=new Helper<E>(root,helpMe.count);
    }
    root.data.sortHeap();
    return helpMe;
}
```

Θ(1)

Θ(n)

Θ(n)

Θ(1)

O(h)

O(h.log(n))

Θ(n)

Θ(1)

T(h-1,n)

T(h-1,n)

Θ(n)

```java
public int remove(E data) {
    Helper<E> helpMe=new Helper<E>(null,-1);
    helpMe=removeFromHeap(new Data<E>(data),tree.root);
    tree.root=helpMe.node;
    return helpMe.count;
}
```

T(h,n)=T(h-1,n)+O(h.log(n))
⇒T(h,n)=O($h^2$ .log(n))

```
private Helper<E> findData(Data<E> data,
                           BinarySearchTree.Node<MyHeap<Data<E>>> root){
    Helper<E> helpMe=new Helper<E>(null,-1);
    int count=-1;
    if (root==null){
        helpMe=new Helper<E>(root,count);
        return helpMe;
    }
    if (root.data.contains(data)){
        helpMe=new Helper<E>(root, root.data.findElement(data).getCount());
    }else if (data.compareTo(root.data.peek())<0) {
        helpMe=findData(data, root.left);
        root.left=helpMe.node;
        helpMe=new Helper<E>(root,helpMe.count);
    }else {
        helpMe=findData(data, root.right);
        root.right=helpMe.node;
        helpMe=new Helper<E>(root,helpMe.count);
    }
    return helpMe;
}
```

Θ(1)
Θ(n)
Θ(n)
Θ(1)
T(h-1)
T(h-1)

```
public int find(E target) {
    Helper<E> helpMe=new Helper<E>(null,-1);
    helpMe=findData(new Data<E>(target),tree.root);
    tree.root=helpMe.node;
    return helpMe.count;
}
```

$T(h,n)=T(h-1,n)+\Theta(n) \Rightarrow h.\Theta(n)$
$\Rightarrow T(h,n)= O(hn)$

```
private Data<E> findMode(BinarySearchTree.Node<MyHeap<Data<E>>> root){
    Data<E> modeLeft=new Data<E>(null);
    modeLeft.setCount(0);
    Data<E> modeRight=new Data<E>(null);
    modeRight.setCount(0);
    if (root==null)
        return null;
    if (root.left!=null)
        modeLeft=findMode(root.left);
    if (root.right!=null)
        modeRight=findMode(root.right);

    MyHeap<Data<E>>.HeapIter itr=root.data.heapIterator();
    while (itr.hasNext()) {
        Data<E> temp= itr.next();
        if (modeLeft.getCount()<=temp.getCount())
            modeLeft=temp;
    }
    return modeLeft.getCount()>=modeRight.getCount() ? modeLeft : modeRight;
}
```

T(h-1,n)
T(h-1,n)
Θ(1)
Θ(n)
Θ(1)
Θ(1)

```
public E find_mode() {
    Data<E> temp = findMode(tree.root);
    return temp.getData();
}
```

Total: $2*T(h-1,n)+\Theta(n)$
$T(h,n)=2^h+\Theta(hn)$

Baran Solmaz 1801042601