CSE 222/CSE505 SPRING 2022

HOMEWORK 6 REPORT

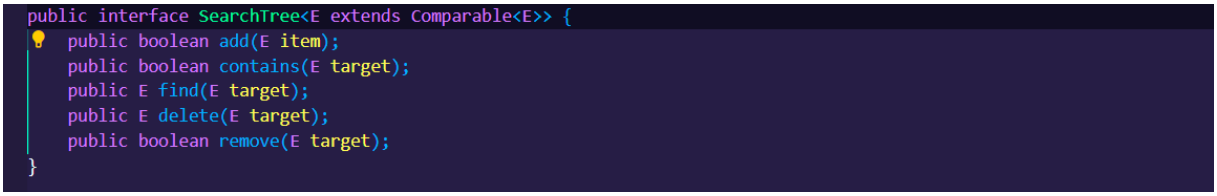
Burcu Sultan Orhan

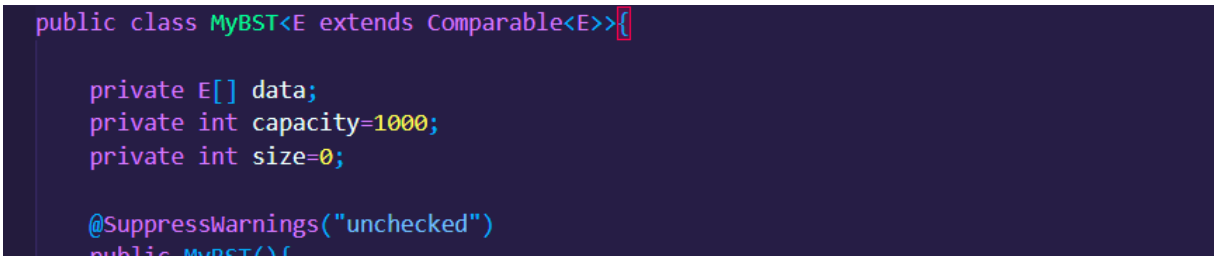
1901042667

1- Detailed System Requirements

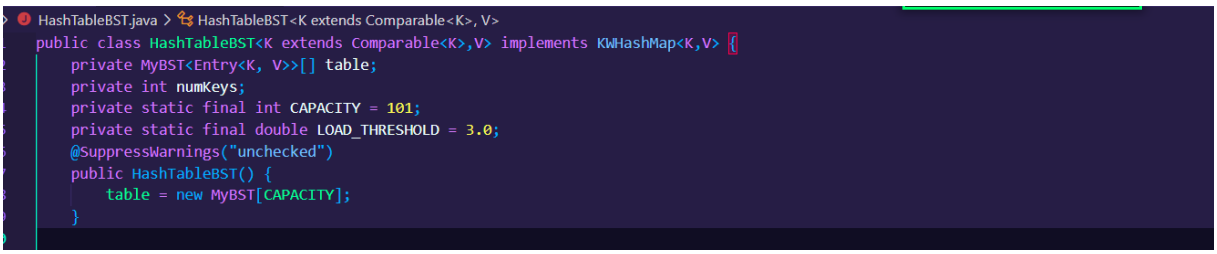
First, there needs to be KWHashMap interface to be implemented

Then there needs to be Binary Search Tree class which implements Search Tree interface





And finally, there needs to be HashTableBST class which implements KWHashMap and uses MyBST class for chaining



There needs to be an Entry class within the HashTableBST class

```
    }

    /**
     * This class implements Entry
     */
    private static class Entry<K extends Comparable<K>, V> implements Comparable<Entry<K,V>>{
        private final K key;
        private V value;

        public Entry(K key, V value) {
            this.key = key;
            this.value = value;
        }

        /**
```

There needs to be get method

```
    /**
     * This method returns the value of the given key
     * @param key key of the value wanted
     * @return value of the given key
     */
    @Override
    public V get(K key) {
```

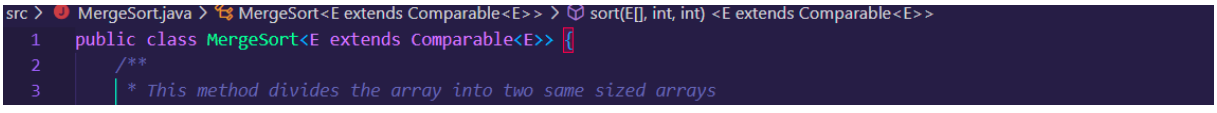
There needs to be isEmpty and put methods

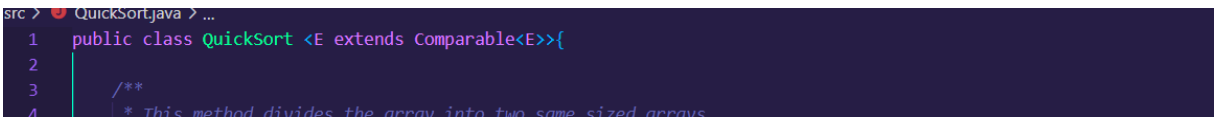
```
    /**
     * This method returns whether the map is empty or not
     */
    @Override
    public boolean isEmpty() {

        return false;
    }

    /**
     * This method puts the given value to the given key
     * @param key key that the value will be put onto
     * @param value value to be put
     * @return old value of the key
     */
    @Override
    public V put(K key, V value) {
```

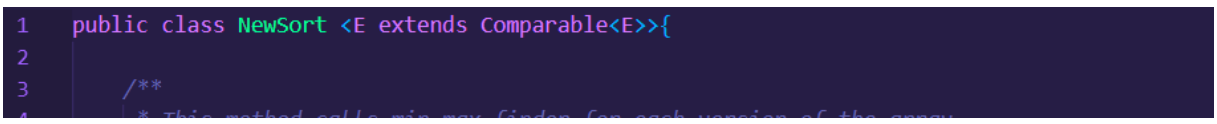
There needs to be remove and size methods

```java
/**
 * This method removes given key and its value
 * @param key key of the value that will be removed
 * @return key that will be removed
 */
@Override
public V remove(K key) {
    int index = key.hashCode() % table.length;
    if (index < 0)
        index += table.length;
    if (table[index] == null)
        return null;
    for (Entry<K, V> nextItem : table[index].getData()) {
        if (nextItem.getKey().equals(key)) {
            V oldVal = nextItem.getValue();
            table[index].remove(nextItem);
            numKeys--;
            if (table[index].size()==0) {
                table[index]=null;
            }
            return oldVal;
        }
    }
    return null;
}

/**
 * This method returns the size of the map
 * @return size
 */
@Override
public int size() {
    return numKeys;
}
```

And for the sorting algorithms, there needs to be three seperate classes for three different algorithms







3- Problem Solving Approach

For first part which I implemented Hash Map, I used Binary Search Tree for chaining, this helped me implement the hash map for chaining. For second part which I implemented different types of sorting algorithms, I had a hard time with the third one because I had to implement a recursive method.

4- Test Cases

```
    for(int x=0; x<100; x++){
        start = System.nanoTime();
        for(int i=0; i<100; i++){
            try{myHash.put(i, i);}catch(ClassCastException e){}
        }
        end = System.nanoTime();
        put100 = put100 + (end-start);
    }

    System.out.println("Average running time for put operation with size 100: " + put100/100);


    for(int x=0; x<100; x++){
        start = System.nanoTime();
        for(int i=0; i<1000; i++){
            try{myHash.put(i, i);}catch(ClassCastException e){}
        }
        end = System.nanoTime();
        put1000 = put1000 + (end-start);
    }


    System.out.println("Average running time for put operation with size 1000: " + put1000/100);


    for(int x=0; x<100; x++){
        start = System.nanoTime();
        for(int i=0; i<10000; i++){
            try{myHash.put(i, i);}catch(ClassCastException e){}
        }
        end = System.nanoTime();
        put10000 = put10000 + (end-start);
    }


    System.out.println("Average running time for put operation with size 10000: " + put10000/100);
```

```
for(int x=0; x<100; x++){
    start= System.nanoTime();
    for(int i=0; i<100; i++){
        try{myHash.get(i);}catch(ClassCastException e){}
    }
    end = System.nanoTime();
    get100 = get100 + (end-start);
}


System.out.println("Average running time for get operation with size 1000: " + get100/100);


for(int x=0; x<100; x++){
    start= System.nanoTime();
    for(int i=0; i<1000; i++){
        try{myHash.get(i);}catch(ClassCastException e){}
    }
    end = System.nanoTime();
    get1000 = get1000 + (end-start);
}


System.out.println("Average running time for get operation with size 1000: " + get1000/100);

for(int x=0; x<100; x++){
    start= System.nanoTime();
    for(int i=0; i<10000; i++){
        try{myHash.get(i);}catch(ClassCastException e){}
    }
    end = System.nanoTime();
    get10000 = get10000 + (end-start);
}


System.out.println("Average running time for get operation with size 10000: " + get10000/100);
```

```
for(int x=0; x<100; x++){
    start = System.nanoTime();
    for(int i=0; i<100; i++){
        try{myHash.remove(i);}catch(ClassCastException e){}
    }
    end = System.nanoTime();
    remove100 = remove100 + (end-start);
}


System.out.println("Average running time for remove operation with size 100: " + remove100/100);

for(int x=0; x<100; x++){
    start = System.nanoTime();
    for(int i=0; i<1000; i++){
        try{myHash.remove(i);}catch(ClassCastException e){}
    }
    end = System.nanoTime();
    remove1000 = remove1000 + (end-start);
}


System.out.println("Average running time for remove operation with size 1000: " + remove1000/100);


for(int x=0; x<100; x++){
    start = System.nanoTime();
    for(int i=0; i<10000; i++){
        try{myHash.remove(i);}catch(ClassCastException e){}
    }
    end = System.nanoTime();
    remove10000 = remove10000 + (end-start);
}


System.out.println("Average running time for remove operation with size 10000: " + remove10000/100);
```

```
for(y=0; y<1000; y++){
    for(int i=0; i<100; i++){
        int_random = rand.nextInt(upperbound);
        myArray1[i] = int_random;
    }
    temp1 = myArray1;
    temp12 = myArray1;

    start = System.nanoTime();
    MergeSort.sort(myArray1, 0, myArray1.length-1);
    end = System.nanoTime();
    time = end-start;
    merge100 = merge100+time;

    start = System.nanoTime();
    QuickSort.sort(temp1, 0, temp1.length-1);
    end = System.nanoTime();
    time = end-start;
    quick100 = quick100+time;

    start = System.nanoTime();
    NewSort.sort(temp12, 0, temp12.length);
    end = System.nanoTime();
    time = end-start;
    new100 = new100+time;
}

merge100=merge100/100;
quick100=quick100/100;
new100=new100/100;
```

```
for(y=0; y<1000; y++){
    for(int i=0; i<1000; i++){
        int_random = rand.nextInt(upperbound);
        myArray2[i] = int_random;
    }
    temp2 = myArray2;
    temp22 = myArray2;

    start = System.nanoTime();
    MergeSort.sort(myArray2, 0, myArray2.length-1);
    end = System.nanoTime();
    time = end-start;
    merge1000 = merge1000+time;

    start = System.nanoTime();
    QuickSort.sort(temp2, 0, temp2.length-1);
    end = System.nanoTime();
    time = end-start;
    quick1000 = quick1000+time;

    start = System.nanoTime();
    NewSort.sort(temp22, 0, temp22.length);
    end = System.nanoTime();
    time = end-start;
    new1000 = new1000+time;
}

merge1000=merge1000/100;
quick1000=quick1000/100;
new1000=new1000/100;
```

```
upperbound = 15000;
for(y=0; y<1000; y++){
    for(int i=0; i<10000; i++){
        int_random = rand.nextInt(upperbound);
        myArray3[i] = int_random;
    }
    temp3 = myArray3;
    temp32 = myArray3;

    start = System.nanoTime();
    MergeSort.sort(myArray3, 0, myArray3.length-1);
    end = System.nanoTime();
    time = end-start;
    merge10000 = merge10000+time;

    start = System.nanoTime();
    QuickSort.sort(temp3, 0, temp3.length-1);
    end = System.nanoTime();
    time = end-start;
    quick10000 = quick10000+time;

    start = System.nanoTime();
    NewSort.sort(temp32, 0, temp32.length);
    end = System.nanoTime();
    time = end-start;
    new10000 = new10000+time;
}

merge10000=merge10000/100;
quick10000=quick10000/100;
new10000=new10000/100;
```

5- Running Command and Results

```
va.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\burcu\Desktop\hw6\1901042667_hw6\bin' 'D
Average running time for put operation with size 100: 431876
Average running time for put operation with size 1000: 274963
Average running time for put operation with size 10000: 483313
Average running time for get operation with size 1000: 384829
Average running time for get operation with size 1000: 193233
Average running time for get operation with size 10000: 339530
Average running time for remove operation with size 100: 389211
Average running time for remove operation with size 1000: 179274
Average running time for remove operation with size 10000: 251532
Average running time for Mergesort with size 100: 12543
with size 1000: 78926
with size 10000: 189883
Average running time for Quicksort with size 100: 23346
with size 1000: 2469001
with size 10000: 25397285
Average running time for Newsort with size 100: 5007
with size 1000: 4484
with size 10000: 392
PS C:\Users\burcu\Desktop\hw6\1901042667_hw6>
```