Gebze Technical University
Computer Engineering

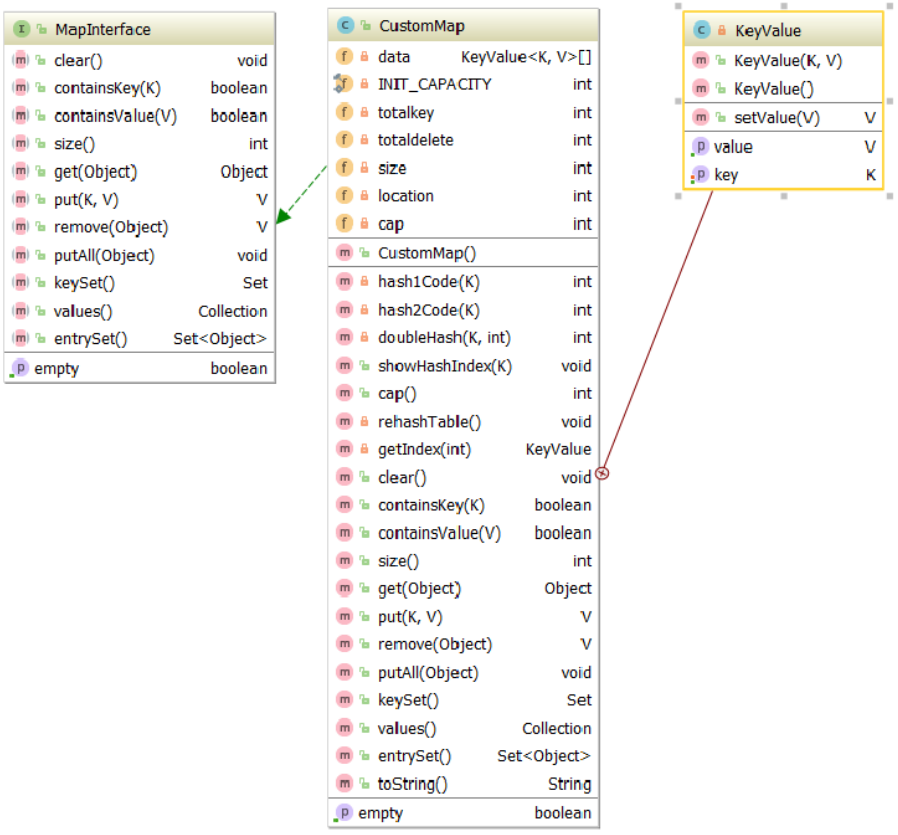
CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

AKIN ÇAM
151044007

Course Assistant: Fatma Nur Esirci

1 Double Hashing Map



MapInterface classı oluşturuldu.
CustomMap classında MapInterface metodları implement edilmiştir. Aralarında Is-A relation ilişkisi bulunmaktadır.
Bu class içerisinde inner KeyValue tipinde bir class oluşturulmuştur. K tipinde key, V tipinde value değişkenlerini tutar. Aralarında Is-A relation ilişkisi bulunmaktadır.
CustomMap KeyValue tipinde array tutmaktadır ve map elemanları burada tutulmaktadır.

1.1 Pseudocode and Explanation

MapInterface classı oluşturuldu.
CustomMap classında MapInterface metodları implement edilmiştir.
Inner KeyValue tipinde bir class oluşturulmuştur.
CustomMap metodları şu tasarıma göre implement edilmiştir:
KeyValue[] başlangıç boyutu bir sabit asal sayı olarak belirlenmiştir.
DoubleHashing su formüle göre yapılmaktadır:

```
        hash1Code(K key){
   return key.hashCode()%data.length)
        hash2Code(K key){
   return 13-(key.hashCode()%13);
        asıl formül== doubleHash(K key,int num){
      return (hash1Code(key)+num*hash2Code(key))%data.length;
```

Put,get,remove methodları containsKey methodunu kullanmaktadır.

ContainsKey(K key)→

- Verilen key in doubleHash ile indexi bulunur. Elemanın bulunup bulunmadığı gösteren bir boolean değişkeni tutulur
- Verilen indexin key i null olmadığı ve boolean false olduğu sürece while döngüsü çalışır.
- Eğer verilen index in value değeri null ise(bu o indexdeki eleman silinmiş demektir.) yeniden hashDouble() ile yeniden index hesaplanır.
- Eğer üsteki kontrol durumunu sağlamıyorsa verilen indexteki key ile aranan key eşleşiyor mu diye bakılır. Eşleşiyorsa boolean değeri true olur ve class değişkeni olan location değişkenine bulunan key in indexi atanır.
- Eğer üsteki iki durumuda sağlamıyorsa yeniden hashDouble() ile yeniden index hesaplanır.

Bu methodun çalışma zamanı best case de elemanın içinde olmaması ve o indexten eleman silinmeme durumudur Omega(1) zaman alır. Worst case de while döngüsü n defa çalışır ve O(n) zaman alır. Genel çalışma zamanına Teta(n) diyebiliriz.

ContainsValue(V value)→

- Map için de bir for döngüsü ile arama yapılır. Bir boolean değişkeni tutulur. .
- Verilen for map sizeından küçük olduğu sürece ve kontrol durumu false olduğu sürece indexlerde dolaşır.
- Verilen index null değil ve aranan value ise location a atama yapar ve boolean true yapar.

Bu methodun çalışma zamanı Teta(n) dir.

Get(Object key)→

- Verilen key null ise null return eder ve Omega(1) zaman alır.
- ContainsKey() methodu çağrılır. Return değeri true ise key in değeri return edilir.

Bu methodun çalışma zamanı containsKey() methodu ile ilişkili olarak Teta(n) dir.

Put(K key, V value)→

- ContainsKey() methodu çağrılır. Return değeri true ise;
- Key in değerini değiştirir. False ise;
- O indexi initializes eder size++ totalnum++
- Load faktörü kontrol eder. Eğer load faktörü geçmişse rehashtable çağrılır.

Bu methodun çalışma zamanı Teta(n) dir.

Remove(K key)→

- ContainsKey() methodu çağrılır. Return değeri true ise;
- O indexin value değerini siler put da aynı indexe eklememek için. Size--. False ise
- Null return eder.

Bu methodun çalışma zamanı Teta(n) dir.

RehashTable()→

- Eski map referansı bir başka değişkene atanır.
- Tüm değerler tekrar initialize edilir.
- Sonra elemanlar yeniden put ile yeni indexlerine eklenir.

1.2 Test Cases

```
Map size is--->0
Map cap is---->101
-----------//---------
first collision first index of two key is --->first-->5  second-->5
key 5 index is---->5
key 106 index is---->16
----------------------
first collision first index of two key is --->first-->52  second-->52
key 52 index is---->52
key 153 index is---->55
----------------------
first collision first index of two key is --->first-->6  second-->6
key 6 index is---->6
key 107 index is---->26
----------------------
first collision first index of two key is --->first-->99  second-->99
key 99 index is---->99
key 200 index is---->14
----------------------
first collision first index of two key is --->first-->3  second-->3
key 3 index is---->3
key 104 index is---->29
-----------//---------
Map size is--->10
-----------//---------
--------------------
key-->3  value-->9
key-->5  value-->1
key-->6  value-->5
key-->200  value-->8
key-->106  value-->2
key-->107  value-->6
key-->104  value-->10
key-->52  value-->3
key-->153  value-->4
key-->99  value-->7
--------------------
```

```
------------------
99(key) is searched in map--->true
998(key) is searched in map--->false
----------//---------
1(value) is searched in map--->true
15(value) is searched in map--->false
----------//---------
isEmpty Method-->false
----------//---------
get method---->6 is searched value of 6 is 5-->5
get method---->41 is searched key is not list-->null
5 is removed from list
------------------
key-->3  value-->9
key-->5  value-->null
key-->6  value-->5
key-->200  value-->8
key-->106  value-->2
key-->107  value-->6
key-->104  value-->10
key-->52  value-->3
key-->153  value-->4
key-->99  value-->7
------------------
first collision first index of two key 5 before removed --->first-->5
after removed--->
key 5 index is---->13
putAll method. Added 3 item
All elements before putAll
----------//---------
after add
------------------
```

----------UNİT TEST----------

containsKey()→

```
7       @org.junit.jupiter.api.Test
8       void containsKey() {
9           CustomMap cs=new CustomMap();
10          cs.put( o: "akin", o2: "akin");
11          System.out.println("a element which is in map searched. return is true-->");
12          System.out.println(cs.containsKey("akin"));
13          assertEquals(cs.containsKey("akin"), actual: true);
14          System.out.println("a element which isn't in map searched. return is false-->");
15          assertEquals(cs.containsKey("akinn"), actual: false);
16          System.out.println(cs.containsKey("akinn"));
17      }
```

CustomMapTest › containsKey()

1 test passed - 12ms

```
"C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
a element which is in map searched. return is true-->
true
a element which isn't in map searched. return is false-->
false

Process finished with exit code 0
```

Get()→

```
18
19      @org.junit.jupiter.api.Test
20      void get() {
21          CustomMap cs=new CustomMap();
22          cs.put( o: "akin", o2: "akin");
23          cs.put( o: "ak21in", o2: "akin11");
24          System.out.println("a element which is in map gets. return is akin-->");
25          System.out.println(cs.get("akin"));
26          assertEquals(cs.get("akin"), actual: "akin");
27          System.out.println("a element which isn't in map get. return is null-->");
28          assertEquals(cs.get("akinn"), actual: null);
29          System.out.println(cs.get("akinn"));
```

CustomMapTest › containsKey()

1 test passed - 11ms

```
"C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
a element which is in map gets. return is akin-->
akin
a element which isn't in map get. return is null-->
null

Process finished with exit code 0
```

Put()→

```
30      }
31
32      @org.junit.jupiter.api.Test
33      void put() {
34          CustomMap cs=new CustomMap();
35
36          System.out.println("a element which isnt in map put this element. return is null-->");
37          assertEquals(cs.put( o: "akin", o2: "akin"), actual: null);
38
39          System.out.println("a element which is in map put this element. return is before value-->");
40          assertEquals(cs.put( o: "akin", o2: "ebru"), actual: "akin");
41      }
42
```

CustomMapTest › containsKey()

1 test passed - 10ms

```
"C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
a element which isnt in map put this element. return is null-->
a element which is in map put this element. return is before value-->

Process finished with exit code 0
```

Remove()→

```
43      @org.junit.jupiter.api.Test
44      void remove() {
45          CustomMap cs=new CustomMap();
46          cs.put( o: "akin", o2: "akin");
47          cs.put( o: "ak21in", o2: "akin11");
48          System.out.println("a element which isnt in map put this element. return is null-->");
49          assertEquals(cs.remove( o: "a"), actual: null);
50          System.out.println("a element which is in map put this element. return is  value-->");
51          assertEquals(cs.remove( o: "akin"), actual: "akin");
52      }
53  }
```

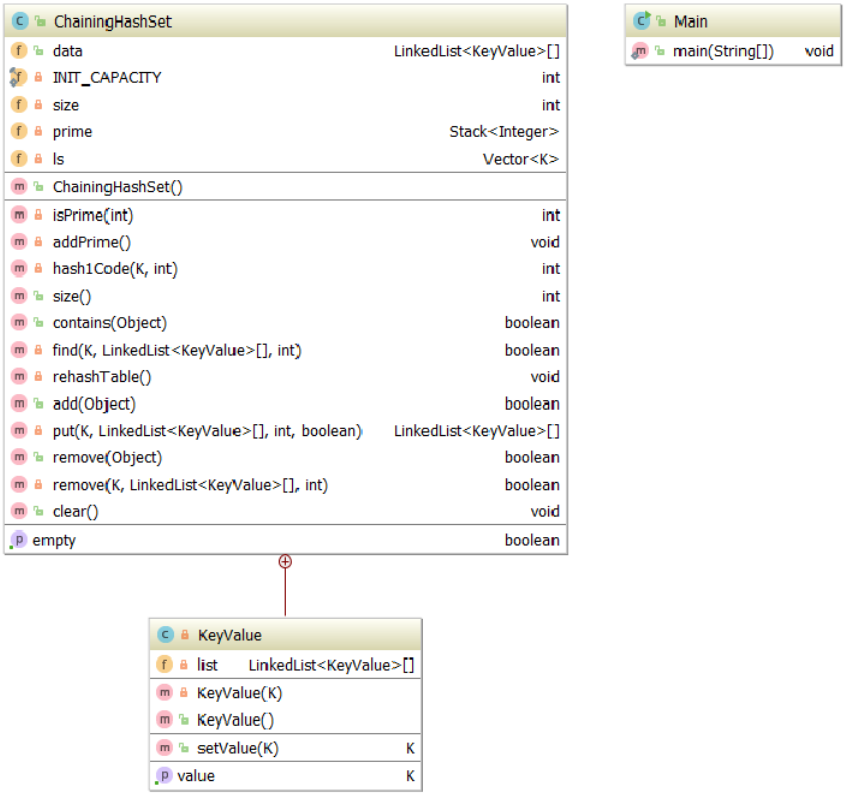
CustomMapTest › containsKey()

1 test passed - 12ms

```
"C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
a element which isnt in map put this element. return is null-->
a element which is in map put this element. return is  value-->


Process finished with exit code 0
```

2 Recursive Hashing Set



2.1 Pseudocode and Explanation

KeyValue inner classı bulunmaktadır.
KeyValue key,deleted ve LinkedList<KeyValue>[] tipinde array tutmaktadır.
Bunun nedeni hashing set i recursive olarak yapmak.
Deleted ile key eşit ise oradan eleman silinmiştir.
ChaningHashSet classında set methodları implement edilmiştir.
Chaining için recursive olarak oluşturulan yeni tabloların size ını bir Stack için küçükten büyüğe prime numberları ekleyerek ve sırasıyla çıkararak initialize edildi.
Collision durumunda KeyValue içinde bulunan LinkedList<KeyValue>[] ile recursive olarak işlemlere devam edilir.
Eğer load factor dengesi bozulursa rehash ile tüm değerler yeniden initialize edilir. Stackten alınan elemanlar sete tekrar eklenir. Diğer türlü çalışma zamanı artacaktır çünkü.
Önemli methodlar aşağıda açıklanmıştır:

Contains()→

- İndex için hash1Code() methodunu çağır ve hesapla.
- İndex in bulunduğu yer null ise false return et
- Eğer index in bulunduğu yerde value null ise içindeki array linked liste bakılır:
- Eğer linked list null değilse oradan devam edilir(find çağrılır (contains yardımcı) methodu çağrılır).
- Eğer linked list de null ise false return edilir
- Eğere key var ise true return edilir.
- Üstekilerden hiçbiri sağlanmıyorsa find methodu çağrılır.

İçindeki işlemler constant time da gerçekleşir. Find methodu n zamanda çalışır. Çalışma zamanı için O(n) deriz.

Add()→

- İndex için hash1Code() methodunu çağır ve hesapla.
- İndex in bulunduğu yer null ise orayı ilklendir ve atama yap. Rehash için stack e bu elemanı ekle size++
- Eğer daha önce eklenip silinmiş bir yere geldiyse yani value null ise put yeniden çağrılır.
- Eğer eleman listede ise false ataması yapar ve listeyi return eder.
- Eğer eklenmek istenen yerde başka bir değer var ise
- Arrayde yer yoksa yer ayrılır var ise put methodu tekrar çalışır.

Çalışma zamanı için lineer zamanda yani Teta(n) zaman çalışma süresine sahiptir.

Remove()→

- İndex için hash1Code() methodunu çağır ve hesapla
- İndex in bulunduğu yer null ise false return eder
- Elemanı bulursa stackten çıkarır ve true return eder.
- Eğer null ise bir alt arrayden devam eder. find(recursive)
- Orası dolu ise find methodu ile devam edilir.

Çalışma zamanı için tüm işlemleri constant olur ve find methodu çalışma zamanı vardır sadece. Lineer dir Teta(n)

2.2 Test Cases

```
collision 1-->

key 3 index is---->3
chaning index->997
key 34 index is---->34



collision 2-->

key 32 index is---->1
chaning index->991
key 1 index is---->1



collision 3-->

key 7 index is---->7
chaning index->983
key 38 index is---->38



size---->6



remove test--->
3 is removed result is true----->true
size---->5
65 is added  but 65 so new 65 dont assigns index of 3
key 65 index is---->65
size---->6
-------------------Test 2----------------------
```

```
--------------------Test 2----------------------


size---->0

collision 1-->

key a index is---->4
key b index is---->5



collision 2-->

key d index is---->7
key e index is---->8



Contains--->
true
false



size---->4



remove test--->
3 is removed result is true----->true
size---->3

Process finished with exit code 0
```

----------UNIT TEST----------

Contains()→

```java
    @Test
    void contains() {
        ChainingHashSet cs=new ChainingHashSet();
        cs.add(3);
        cs.add(5);
        cs.add(2);
        System.out.println("contains method---->");
        System.out.println(cs.contains(3));
        assertEquals(cs.contains(3), actual: true);
        System.out.println("contains method element which is not in set ---->");
        System.out.println(cs.contains(6));
        assertEquals(cs.contains(6), actual: false);
```

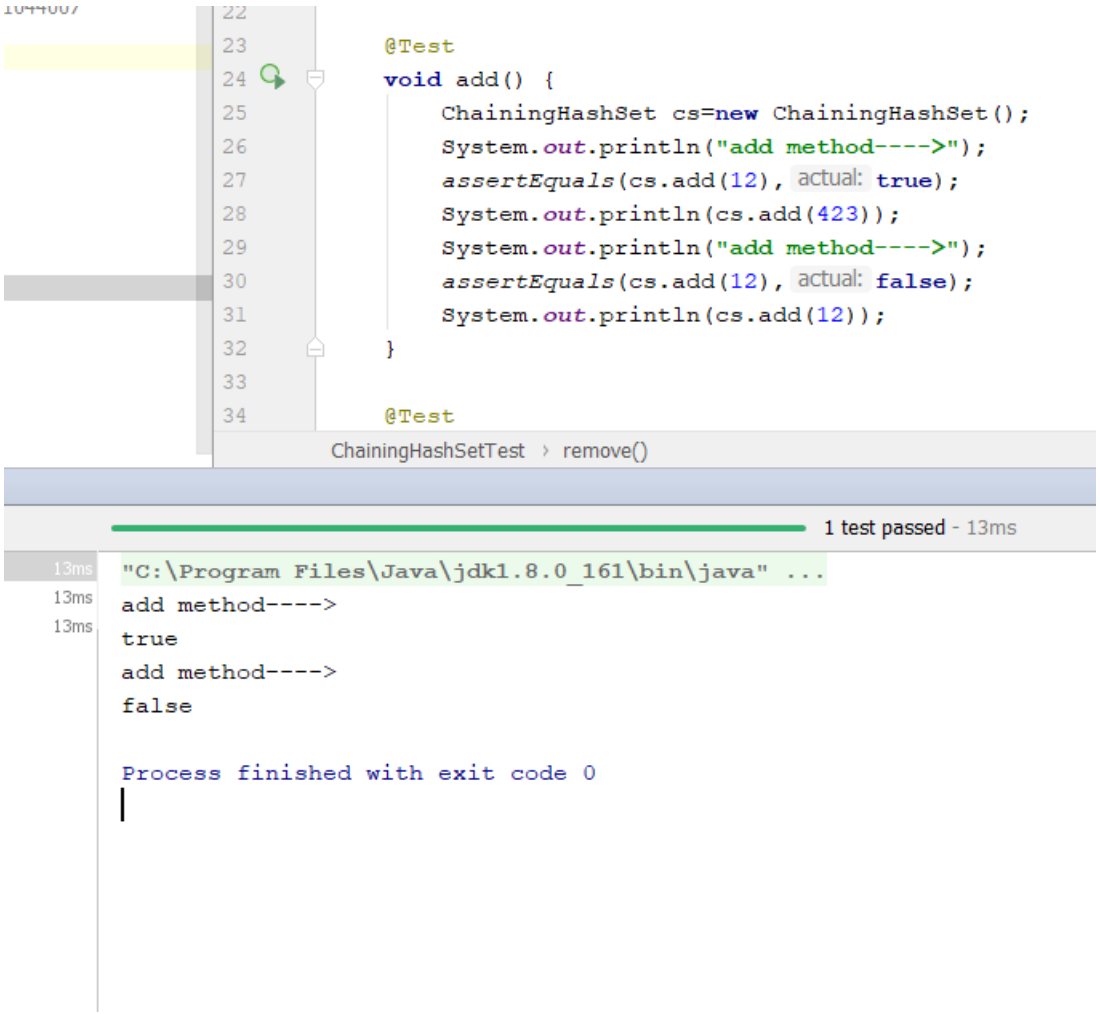
ChainingHashSetTest › remove()

1 test passed - 13ms

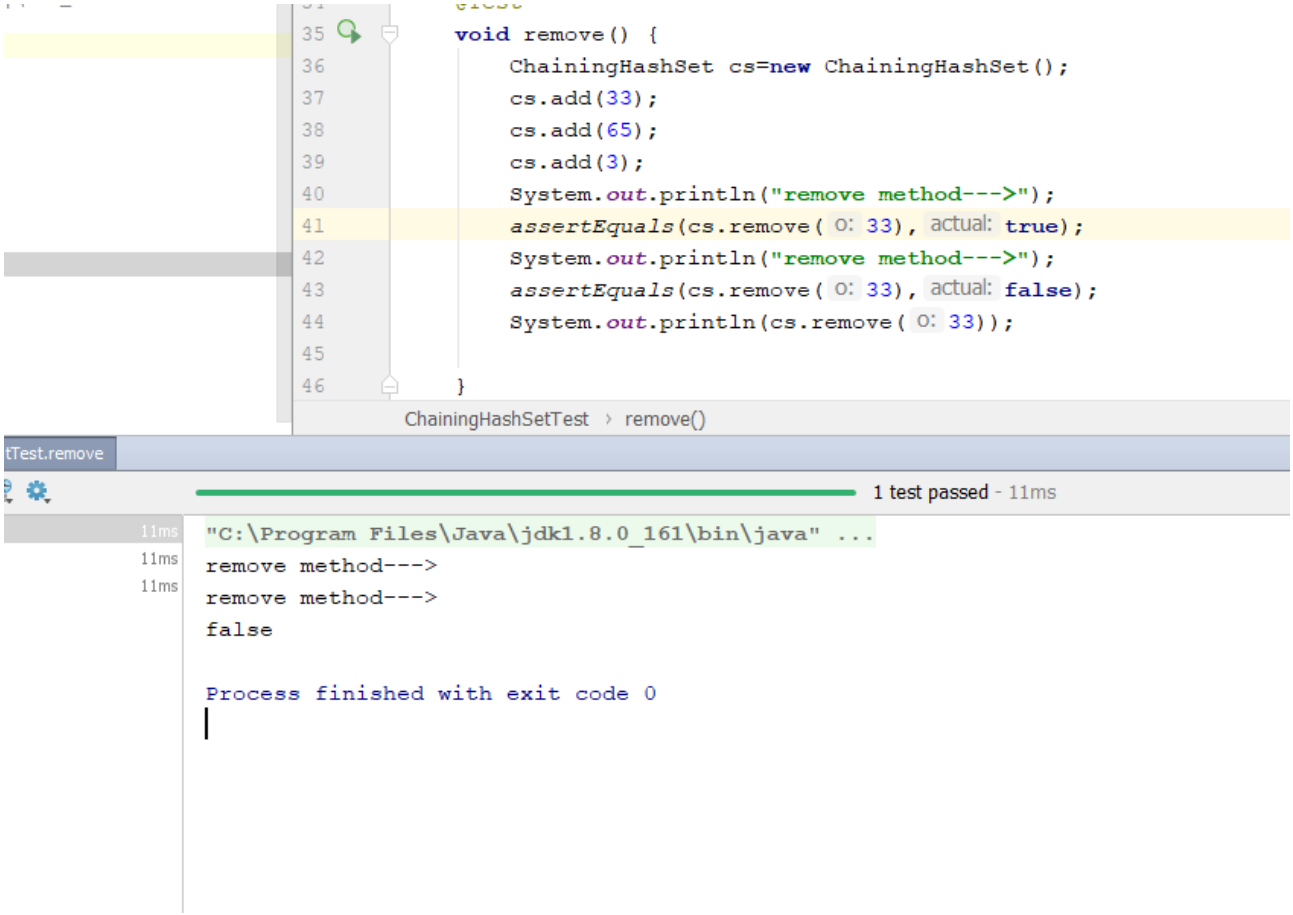
```
"C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
contains method---->
true
contains method element which is not in set ---->
false

Process finished with exit code 0
```

Add()→

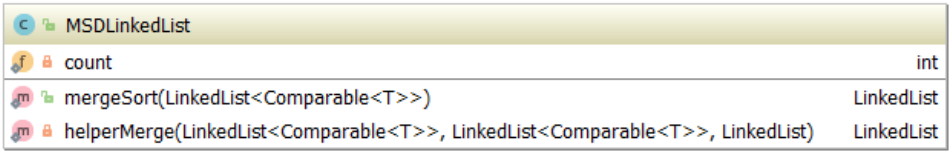

@Test
void add() {
ChainingHashSet cs=new ChainingHashSet();
System.out.println("add method---->");
assertEquals(cs.add(12), actual: true);
System.out.println(cs.add(423));
System.out.println("add method---->");
assertEquals(cs.add(12), actual: false);
System.out.println(cs.add(12));
}
@Test
ChainingHashSetTest › remove()
1 test passed - 13ms
"C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
add method---->
true
add method---->
false
Process finished with exit code 0

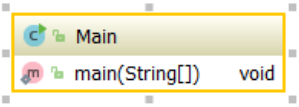
Remove()→



3 Sorting Algortihms

3.1 MergeSort with DoubleLinkedList





3.1.1 Pseudocode and Explanation

mergeSort(LinkedList<Comparable<T>> list)→

- Listenin size ı 1 veya 0 ise direkt listeyi return eder ve çıkar
- Listenin size ı 1 den büyükse iki linked list oluşturur
- Elemanların yarısını birinci linked liste geri kalanını 2. Linked liste gönderir.

- Recursive olarak o linked listleri methoda gönderir ve en küçük parçayı elde eder.
- Daha küçük parçaya bölünemedğinde diğer method çağrılır.
- left.size()>0&&right.size()>0 sürece elemanları karşılltırır ve bir linkedliste ekler
- sonra kalan elemanları yani sıralanmış fakat büyük o elemanı da res liste ekleyip return eder.
-

Çalışma zamanı için ana method recursive olarak çalışır ve Teta(n) çalışma süresine sahiptir.ikinci method 1 2…. Artarak devam eder ve logaritmik . Sonuç olarak n.logn gibi biraz daha yavaş olabilir çünkü constant durumları ekledğimizde O(n^2logn) çalışma zamanına sahiptir.

3.1.2 Average Run Time Analysis

Average time da n(logn) zamanda çalışır

3.1.3 Wort-case Performance Analysis

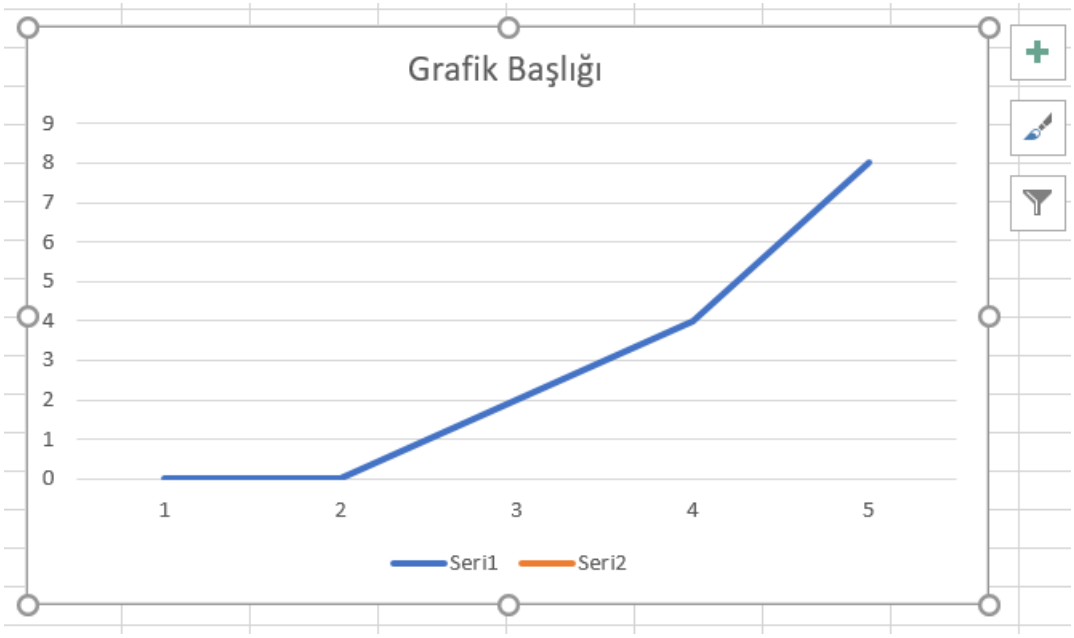
Çalışma zamanı için ana method recursive olarak çalışır ve Teta(n) çalışma süresine sahiptir.ikinci method 1 2…. Artarak devam eder ve logaritmik . Sonuç olarak n.logn gibi biraz daha yavaş olabilir çünkü constant durumları ekledğimizde O(n^2logn) çalışma zamanına sahiptir.

3.2 MergeSort

3.2.1 Average Run Time Analysis

Merge sort average da ne lineerden biraz fazla quadraticten biraz az bir artışla ilerlemektedir. O(nlogn) dir ortalama çalışma süresi.

3.2.2 Wort-case Performance Analysis



O(n^2) diyebiliriz

3.3 Insertion Sort

3.3.1 Average Run Time Analysis

Insertion sort belkide en kötü sort algoritmasıdır. Değerlerdende anlaşıldığı üzere average da O(n^2) çalışmaktadır. Quadratictir.

3.3.2 Wort-case Performance Analysis

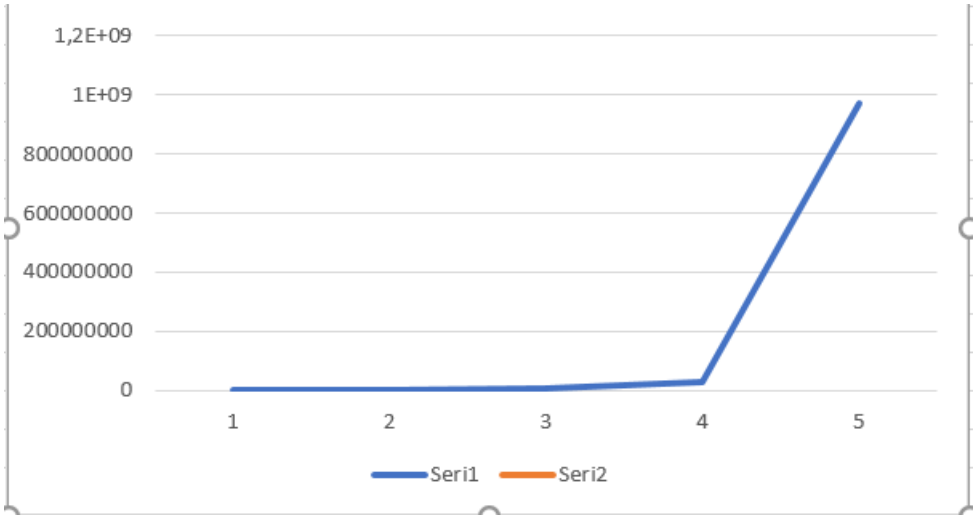
Quadratic bir artış gösterir. O(n^2) dir.

3.4 Quick Sort

3.4.1 Average Run Time Analysis

Quick sort da average da ne lineerden biraz fazla quadraticten biraz az bir artışla ilerlemektedir. O(nlogn) dir ortalama çalışma süresi. Verimli bir sort algoritmasıdır.

3.4.2 Wort-case Performance Analysis



Quick sort size arttıkça quadratic bir artış gösterir O(n^2) dir.

3.5 Heap Sort

3.5.1 Average Run Time Analysis

Heap sort average değerlere bakıldığında lineerden biraz daha uzun süren bir çalışma süresine sahiptir ama bu quadratic kadar değildir. O(nlogn) diyebiliriz.

3.5.2 Wort-case Performance Analysis

Grafikten de görüldüğü gibi worst case de n^2 ile n arasında değişir.

4 Comparison the Analysis Results

This part about Question5 in HW5. Using before analysis results in show that section 3. Show that one graphic (like Figure 4.1) include 5 sorting algorithm worst-case analysis cases.

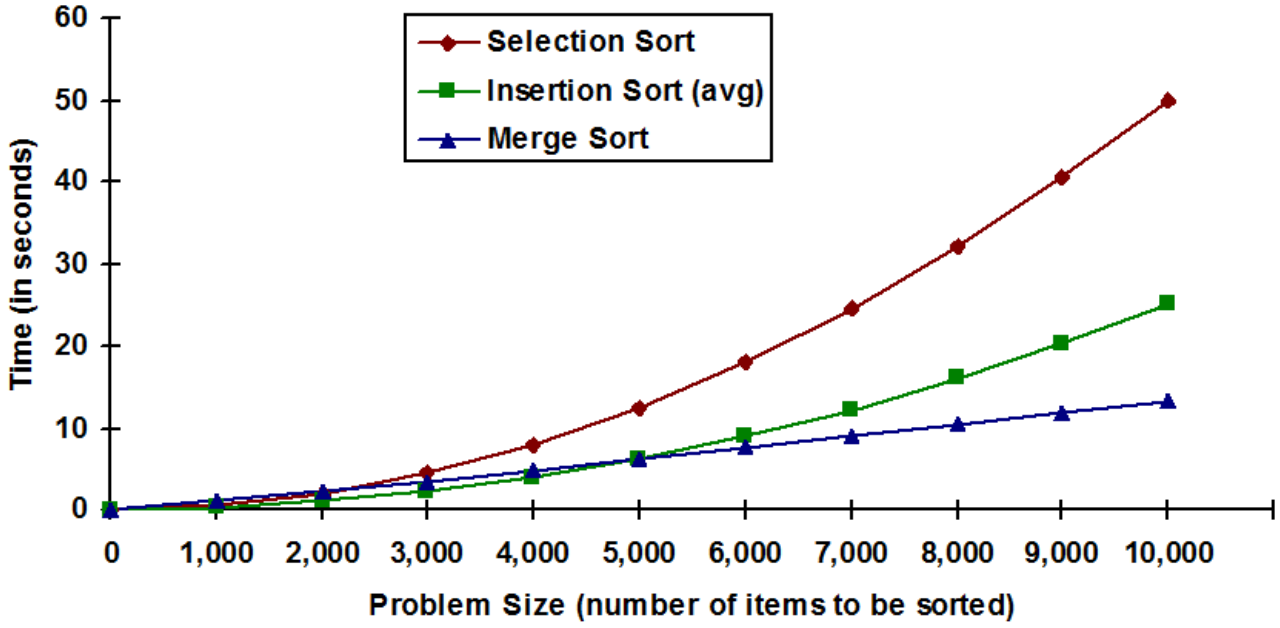
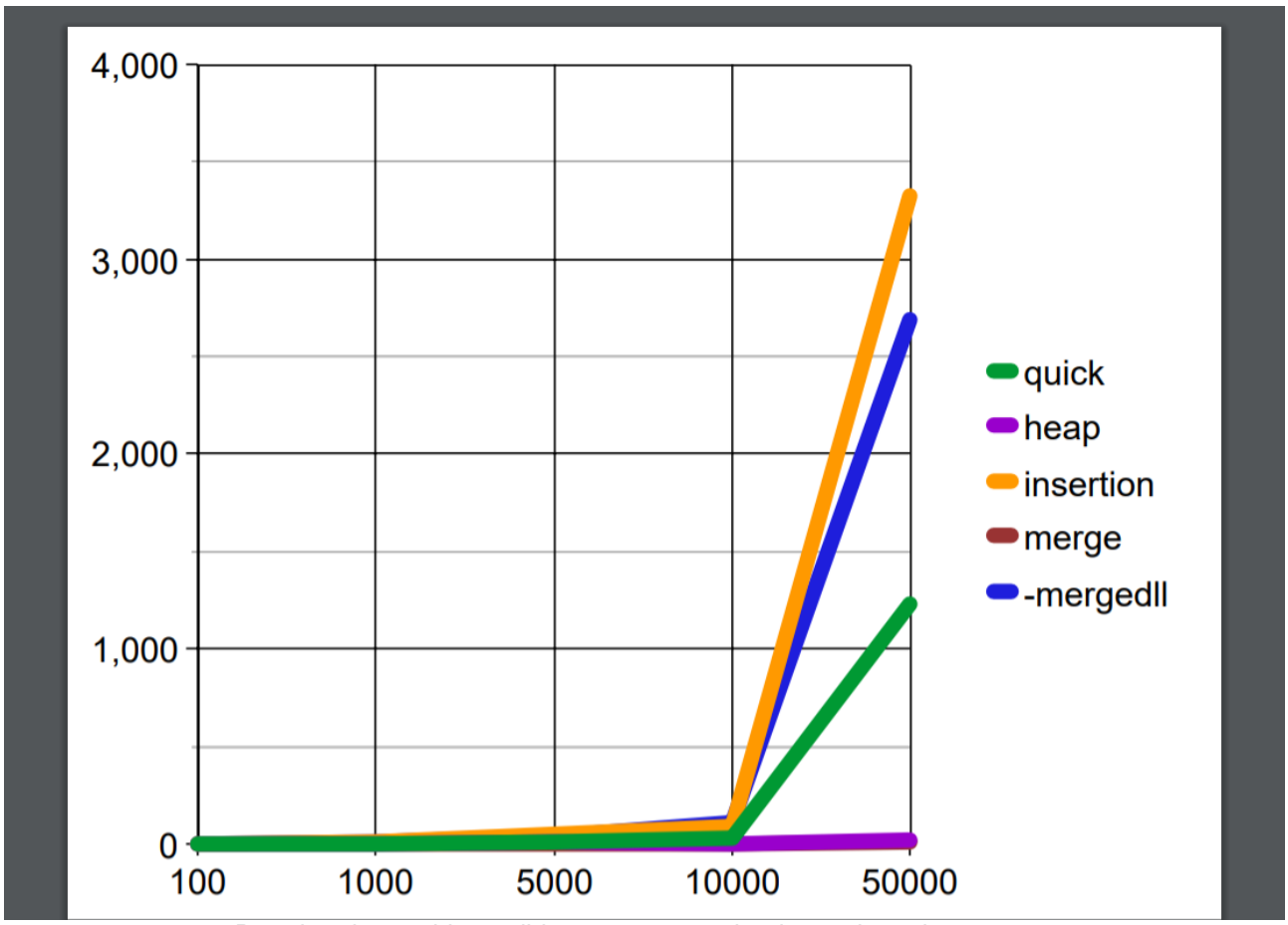


Figure 4.1. Comparison of sorting algorithms (this figure just a example)



Buradan da görüldüğü gibi en yavaş sort algoritması insertion sort tur.
MergeDll de hemen hemen aynıdır.
Quick sort kullanılabilir ve verimli bir sort algoritmasıdır.
Merge Sort gözükmüyor çünkü heap sort ile aynı zamanda çalışırlar.
Heap sort yeterince hızlı bir sort algoritmasıdır.