Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

SİNAN ELVEREN
111044074

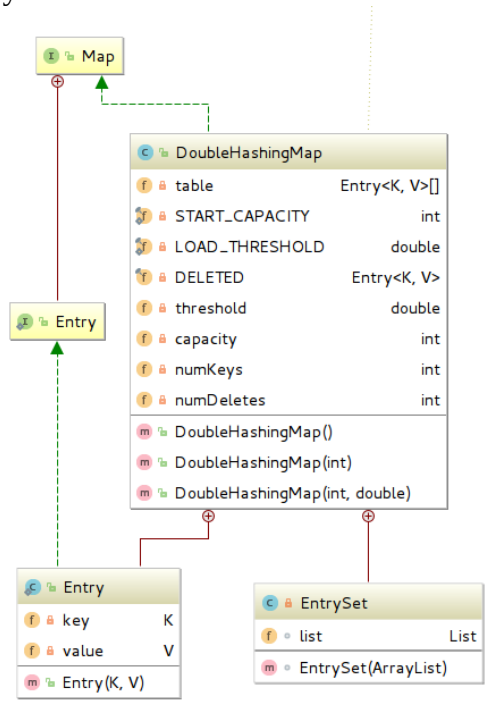
Course Assistant:
Fatma Nur Esirci, Tuğbagül Altan Akın, Mehmet Burak Koca

1 Double Hashing Map

Implementation of Java Map by double hashing. First hash is calculating index for table. İf there are any collision then hash2 will callculate new index for table. After that, if there are any collision then wil apply lineer probing.

1.1 Pseudocode and Explanation

There are one main class named DoubleHashingMap that implements MapInterface. This class is including 2 subClasses, first one is Entry class that implement MapEntry Class, and second one EntrySet class that extends AbstractSet.



>DoubleHashingMap<K, V> implements MapInterface
>SubClass Entry<K, V> implement MapEntry
>subClass EntrySet Extends AbstractSet <Map.Entry<K,V>>

>put(K, V)
index ← findIndex
if table[index] equ null then
table[index] ← Entry(K,V)
numKeys ← numKeys + 1
if (loadFactor greater (numKeys+ numDeletes / table.size)) then
reHash()
return null

return oldValue

Method takes a pair and add this into map after find index . Using 1 or 2 hash to find the index. If there are over capacity then rehash the struct.

>find(key)

```
index = key.hashCode() % table.size
if  index smaller  0  then
        index ← table.size + index

if(table[index] equ null ) and (key NOT equ  table[index]) then
        oldIndex ← index
        index ←  getPrimeNumber() - ( key.hashCode() mod getPrimeNumber() )
        index ← ( index + old) mod table.size

while( (table[index ] NOT equ null ) and (key NOT equ table[index].key) )
        index ← index + 1
        if ( index greater or equ table.size) then
                index ← 0                //wrap
return index;
```

Method calculating index acording to has code of key, returns index which table's slot is empty acording to calculating(hashing and lineer probing).

>rehash()

```
updateCapacity()
new table[capacity]
numKeys ← 0
numDeletes ← 0
for  ( from   index ← 0 to table.size ,  index smaller  then table.size)
        if ( ( oldTable not Equ null ) and ( oldTable[i] != DELETED) then
                put(oldTable[i].key , oldTable[i].value)
```

Method is updating to capacity, making zero to numKeys and numDeletes. After that, copy all elements from old table to new table.

>updateCapacity

```
this capacity ← this capacity * 2 + 1
```

>containsKey(key)

```
index ← find(key)
if(table[index] NOT equ null) then
        if(table[index].key equ key ) then
                return true
return false
```

Find the index of spesific key, and check it if table contains this key.

>containsValue(value)

```
for (from index ← 0 to table.size)
        if ((table[index] not equ null)  and (table[index] not equ DELETED) ) then
                if( value equ table[index].value) then
                        return true
return false;
```

Find the index of spesific value, and check it if table contains this value.

>get(key)

```
index ← find(key)
if table[index].value not equ then
        return table[index].value
else
        return null                // key not found
```

Find the index of spesific key and, return the value of this key.

>remove(key)

```
index ← find(key)
if table[index] not equ  null then
        oldValue ← table[index].value
        numKeys ← numKeys -1
        numDeletes ← numDeletes – 1

        table[index] ← DELETED
        return oldValue
else
        null
```

Find the index of spesific key and, remove the item which have this key

>putAll(<K, V> m)

```
m.keySet.forEach (k)
        put(k, m.value)
```

Find the key set of m map and, add them in to table which have this keys.

>clear()

```
for (from index ← 0 to this.capacity )
        table[index] ← null
numKeys ← o
numDeletes ← 0
```

Traverse all table and make them null, so clear to the table.

>keySet()

```
keySet ← new Set
for (from index ← 0 to table.size )
        if table[index] is valid then
                keySet.add ← table[index].value
return keySet
```

Traverse all table and find the not null/deleted items and add them into new set

>values()

```
collection ← new collection
for (from index ← 0 to tablele.size)
        if table[index] is valid then
                collection.add ← table[index.value]
return collection
```

>entrySet()

```
for ( from index ← 0 to table.size)
    if table[index] is valid then
        Entry() ← table[index].key
        Entry() ← table[index].value
        arrayList.add ← Entry()
return set of ArrayList
```

1.2 Test Cases

All excepted function working correctly

```
Map<Integer, String> testDoubleHash = new DoubleHashingMap<Integer, String>();
Map<Integer, String> testDoubleHash2 = new DoubleHashingMap<Integer, String>( capacity: 20, threshold: 0.5);
```

First map 's capacity is default(11), threshold is 0.75
Second one 20 and 0.5

```
System.out.println("Size() : " + testDoubleHash.size());
System.out.println("containsKey(2) : " + testDoubleHash.containsKey(2));

System.out.println("\nput(1, sinan) : " + testDoubleHash.put(1,"sinan"));
System.out.println("put(2, elveren) : " + testDoubleHash.put(2,"elveren"));

System.out.println("\ncontainsKey(2) : " + testDoubleHash.containsKey(2));

System.out.println(testDoubleHash.toString());
```

Check size firtly, and check put, containsKey methods.
Output is:

You can see the table as column by column as 5

```
/usr/lib/jvm/java-8-oracle/bin/java ...
Testing PART 1
Size() : 0
containsKey(2) : false

put(1, sinan) : null
put(2, elveren) : null

containsKey(2) : true

MAP :
0: [0 ->  NULL ]        1: [1 -> sinan]        2: [2 -> elveren]      3: [3 ->  NULL ]        4: [4 ->  NULL ]
5: [5 ->  NULL ]        6: [6 ->  NULL ]        7: [7 ->  NULL ]       8: [8 ->  NULL ]        9: [9 ->  NULL ]
10: [10 ->  NULL ]
```

call : `testDoubleHash.clear();`

```
clear()
size() : 0
```

add some elements and check containsKey again

```
put(1, sinan) : null
put(2, elveren) : null
put(10, Gebze) : null
put(11, Tech) : null
put(12, Univ) : null

containsKey(2) : true
```

See table via toString()

```
MAP :
0: [11 -> Tech]         1: [1 -> sinan]         2: [2 -> elveren]       3: [3 ->  NULL ]        4: [4 ->  NULL ]
5: [5 ->  NULL ]        6: [6 ->  NULL ]        7: [12 -> Univ]         8: [8 ->  NULL ]        9: [9 ->  NULL ]
10: [10 -> Gebze]
```

There are some collision, you can see
Check get(key) returns null or old value correctly

```
Get(2) : elveren
Remove(2) : elveren
Get(2) : null
```

Also checked remove() method

Now check put() again, add same element and see result.

```
put(1, sinan) change : sinan

put(13, onUc)  : null

put(14, onDort)  : null
```

Put(1, sinan) is already in the table, so it returns old value.

See table again via toString()
and check the size()

```
Size() :6

MAP :
0: [11 -> Tech]         1: [1 -> SINAN]         2: [2 ->  DELETED ]     3: [14 -> onDort]       4: [4 ->  NULL ]
5: [5 ->  NULL ]        6: [6 ->  NULL ]        7: [12 -> Univ]         8: [13 -> onUc]         9: [9 ->  NULL ]
10: [10 -> Gebze]
```

```
put(22, yirmiIki)  : null

put(20, yirmi)  : null

oto rehash
Size() :8

MAP :
0: [0 ->  NULL ]        1: [1 -> SINAN]         2: [2 ->  NULL ]        3: [3 ->  NULL ]        4: [4 ->  NULL ]
5: [5 ->  NULL ]        6: [6 ->  NULL ]        7: [7 ->  NULL ]        8: [8 ->  NULL ]        9: [9 ->  NULL ]
10: [10 -> Gebze]       11: [11 -> Tech]        12: [12 -> Univ]        13: [13 -> onUc]        14: [14 -> onDort]
15: [15 ->  NULL ]      16: [16 ->  NULL ]      17: [17 ->  NULL ]      18: [18 ->  NULL ]      19: [19 ->  NULL ]
20: [20 -> yirmi]       21: [21 ->  NULL ]      22: [22 -> yirmiiki]
```

now add some elements and see collision.

```
containsKey(11)true
containsKey(15)false
containsKey(20)true
containsValue(Tech)true
containsValue(yok)false
containsValue(yirmi)true
```

ContainsKey( ) and containsValue() are working correctly
EntrySet() check via forEach()

```
EntrySet() - Iterator via forEach
[1]->   [SINAN]
[10]->  [Gebze]
[11]->  [Tech]
[12]->  [Univ]
[13]->  [onUc]
[14]->  [onDort]
[20]->  [yirmi]
[22]->  [yirmiiki]
Size() : 8
```

keySet() via system.out.println()

```
Test keySet() : [1, 20, 22, 10, 11, 12, 13, 14]

put(7) & put(24) in to newMap & putAll to oldMap

keySet() : [0, 1, 4, 5, 7, 10, 11, 12, 13, 14, 20, 22, 24]
Get(24) : 2.4.
Remove(24) : 2.4.

keySet() : [0, 1, 4, 20, 5, 22, 7, 10, 11, 12, 13, 14]
```

also tested putAll()
new map include 7, 24, 5, 0, 4

```
MAP :
0: [0 -> ZERO]      1: [1 ->  NULL ]        2: [2 ->  NULL ]        3: [3 ->  NULL ]        4: [24 -> 2.4.]
5: [5 -> FIVE]      6: [6 ->  NULL ]        7: [7 -> SEVEN]         8: [8 ->  NULL ]        9: [9 ->  NULL ]
10: [10 ->  NULL ]      11: [11 ->  NULL ]      12: [12 ->  NULL ]      13: [13 ->  NULL ]      14: [14 ->  NULL ]
15: [15 ->  NULL ]      16: [16 ->  NULL ]      17: [17 ->  NULL ]      18: [18 ->  NULL ]      19: [4 -> FOUR]
```

Ok, check the values() now.

```
keySet() : [0, 1, 4, 20, 5, 22, 7, 10, 11, 12, 13, 14]

Values() : [ZERO, SINAN, FOUR, FIVE, SEVEN, Gebze, Tech, Univ, onUc, onDort, yirmi, yirmiiki]
```

See the table again

```
MAP :
0: [0 -> ZERO]      1: [1 -> SINAN]         2: [2 ->  NULL ]        3: [3 ->  NULL ]        4: [4 -> FOUR]
5: [5 -> FIVE]      6: [6 ->  NULL ]        7: [7 -> SEVEN]         8: [8 ->  NULL ]        9: [9 ->  NULL ]
10: [10 -> Gebze]       11: [11 -> Tech]        12: [12 -> Univ]        13: [13 -> onUc]        14: [14 -> onDort]
15: [15 ->  DELETED ]   16: [16 ->  NULL ]      17: [17 ->  NULL ]      18: [18 ->  NULL ]      19: [19 ->  NULL ]
20: [20 -> yirmi]       21: [21 ->  NULL ]      22: [22 -> yirmiiki]
```

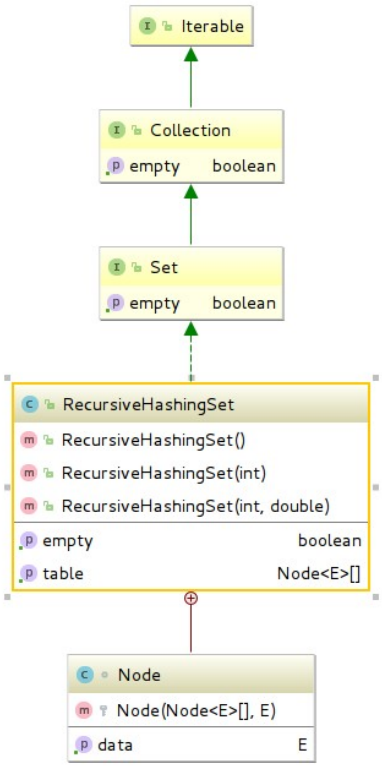
Final table.

2 Recursive Hashing Set

Implementation of Java Set by recursive hashing. Hash1 is calculating index for table. İf there are any collision then call new table for hash it again and callculate new index for table, so it is recursively. Next table 's capacity is calculeted as prime number that smaller then table size, if the new table's capacity is smaller then 1/3 of table size,then multiple the size as 4x+1. So, the tables size decreasing firstlty, but no more until to tableSize/3.

2.1 Pseudocode and Explanation

There are one main class named RecursiveHashingSet that implements SetInterface(iterable-Collection-Set). This class is including one sub class named node. That keeps item's data and nex table.



>**RecursiveHashingSet<E>** implements SetInterface

>**SubClass Node<E>** Keeps data & nextTable

>**add(element)**

if table contains element then
return false
return add(table.next, element)

>**add(table[], element)**

```
index ← find()
if table[index] equ null then
        table[index] ← new Node(table, e)
        numElements ← numelements + 1
        if( size greater then LoadFactor)
                reSize();
        return true;
else
        return false
```

>**remove(element) and remove(table, element)**
workink similar like add methods

>**toString()**

new StringBuilder sb
CALL traverseTable(this.table, depth:1, sb)
return sb.toString

>traverseTable(table[], depth, StrgingBuilder)

```
if depth EQU 1
        sb ← “index”
for ( from index ← 0  to depth)
        sb ← TAB indention
if (table NOT EQU null) then
        if (table[0] EQU null) then
                sb ← null
        else
                if (table[0]  EQU isDeleted)
                        sb ← DELETED
                else
                        sb ← new line
                if ( table[0].nextTable NOT EQU null) then
                        traverseTable(table[0], nextTable, depth, sb)
        if ( table[0].nextTable NOT EQU nıull) then
                traverseTable(table[0], nextTable, depth, sb)
```

>toArray(a[])

```
new ArrayList → array
for (from i<- 0 to array lengh)
        if (a[i] NOT EQU null) then
                array[i] ← a[i]
return helperToArray(array, this.table).toArray
```

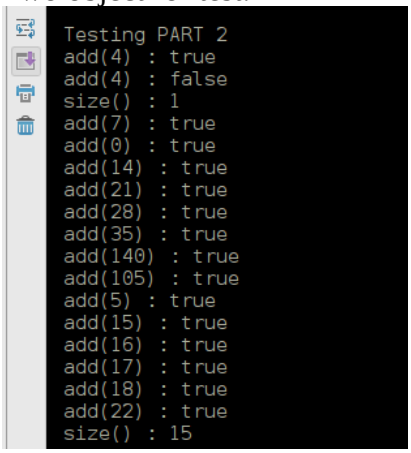
2.2 Test Cases

All excepted function working correctly excluding iterator method

First I have no so much time for part 3 4 and 5, so this parts are not include all excepted function. I dint show them e xactly in test.

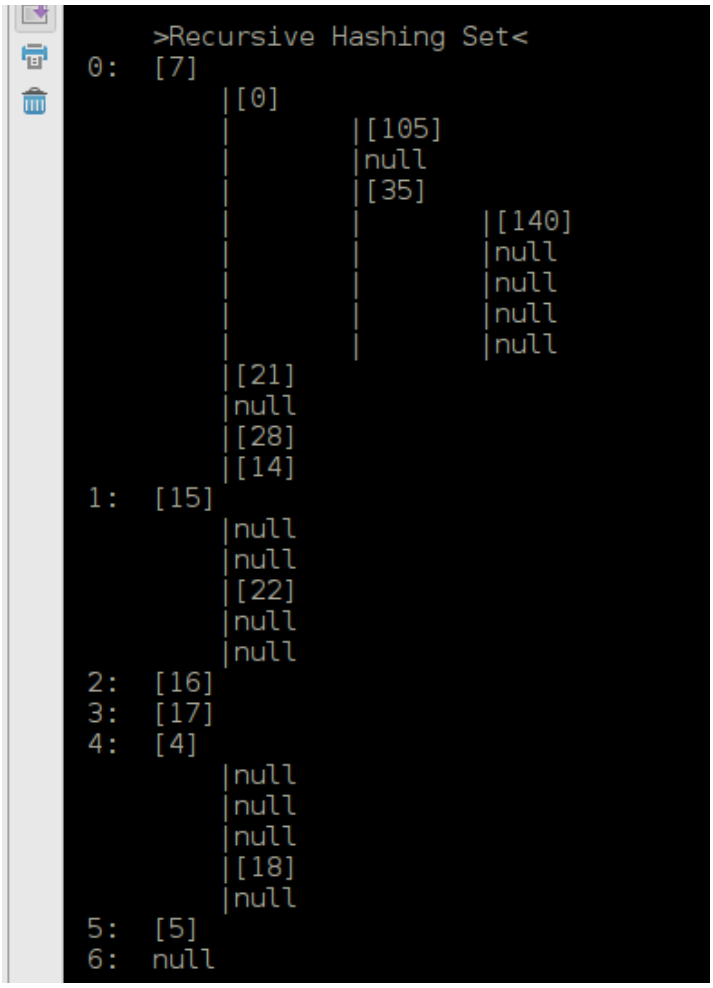
```
RecursiveHashingSet<Integer> testRecHashSet = new RecursiveHashingSet<>();
RecursiveHashingSet<Integer> testRecHashSet2 = new RecursiveHashingSet<>( capacity: 20);
```

Two object for test.

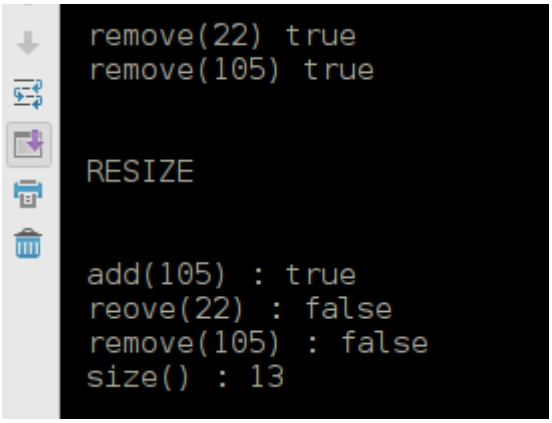


Added some elements

now see the table,call toString that recusrively and shows tablo **correctly** with indentetion acording to column

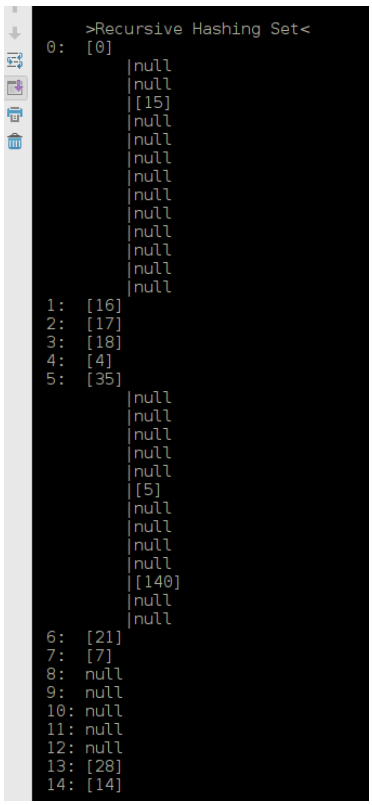


remove()
and size()

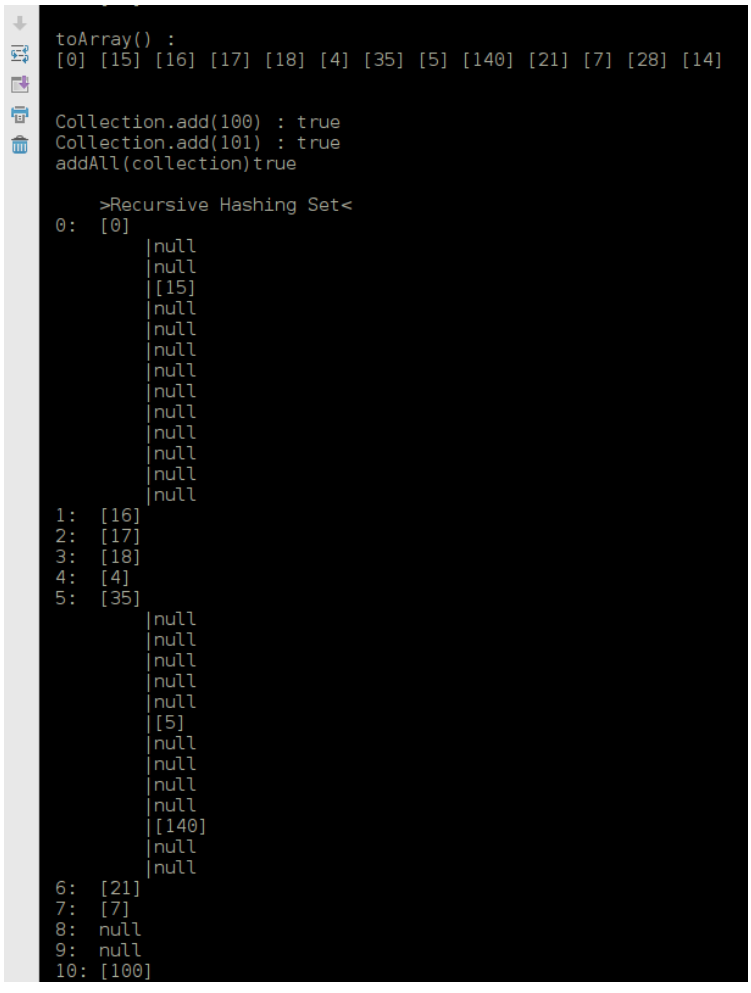


I caused to table's full, so resize when add item

see the new table.



Toarray()
Collection.add
addAll()



```
Test toArray(arr [101, 102, 103]) :

toArray( arr[] ) :
[101] [102] [103] [0] [15] [16] [17] [18] [4] [35] [5] [140] [21] [7] [100] [28] [14]

Contains(7) : true
Contains(13) : false

toArray() :
[0] [15] [16] [17] [18] [4] [35] [5] [140] [21] [7] [100] [28] [14]

RemoveAll(collection [100, 5] ): true
toArray() :
[0] [15] [16] [17] [18] [4] [35] [140] [21] [7] [28] [14]

retainAll( collection arr[100, 5, 7, 21] ) : true
toArray() :
[21] [7]

ContainsAll( arr[100, 5, 21, 7] ) : false


ContainsAll( arr[21, 7] ) : true


set1.equals(set2) : TRUE?true


set1.equals(set2) : FALSE?false


Clear() --> Size() : 0
```

3 Sorting Algortihms

3.1 MergeSort with DoubleLinkedList

This part about Question3 in HW5

3.1.1 Pseudocode and Explanation

Write pseudocode and explanation about code design. Indicate what you are using that interfaces, classes, structures, etc.

3.1.2 Average Run Time Analysis

This part about Question4 in HW5

3.1.3 Wort-case Performance Analysis

This part about Question5 in HW5

3.2 MergeSort

```
Testing PART 3
Forward Traversal using next pointer
37 38 40 47 51 58 60 81 83 86 87 95
Backward Traversal using prev pointer
95 87 86 83 81 60 58 51 47 40 38 37 26 10 8
```

This part about code in course book.

3.2.1 Average Run Time Analysis

This part about Question4 in HW5

```
Testing PART 4



0. AVARAGE :
 : MERGE DLL = 5194591
 : MERGE = 4796946
 : INSERT = 76491745
 : QUICK = 19264943
 : HEAP = 8528696


1. AVARAGE :
 : MERGE DLL = 3828488
 : MERGE = 814939
 : INSERT = 7446623
 : QUICK = 672945
 : HEAP = 805126


2. AVARAGE :
 : MERGE DLL = 3563920
 : MERGE = 1718410
 : INSERT = 24646595
 : QUICK = 1408855
 : HEAP = 1529578
```

Run the proje and follow for more
I couldn't fnish it before deat line of project.

3.2.2 Wort-case Performance Analysis

This part about Question5 in HW5

3.3 Insertion Sort

3.3.1 Average Run Time Analysis

This part about Question4 in HW5

3.3.2 Wort-case Performance Analysis

This part about Question5 in HW5

3.4 Quick Sort

3.4.1 Average Run Time Analysis

This part about Question4 in HW5

3.4.2 Wort-case Performance Analysis

This part about Question5 in HW5

3.5 Heap Sort

3.5.1 Average Run Time Analysis

This part about Question4 in HW5

3.5.2 Wort-case Performance Analysis

This part about Question5 in HW5

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.

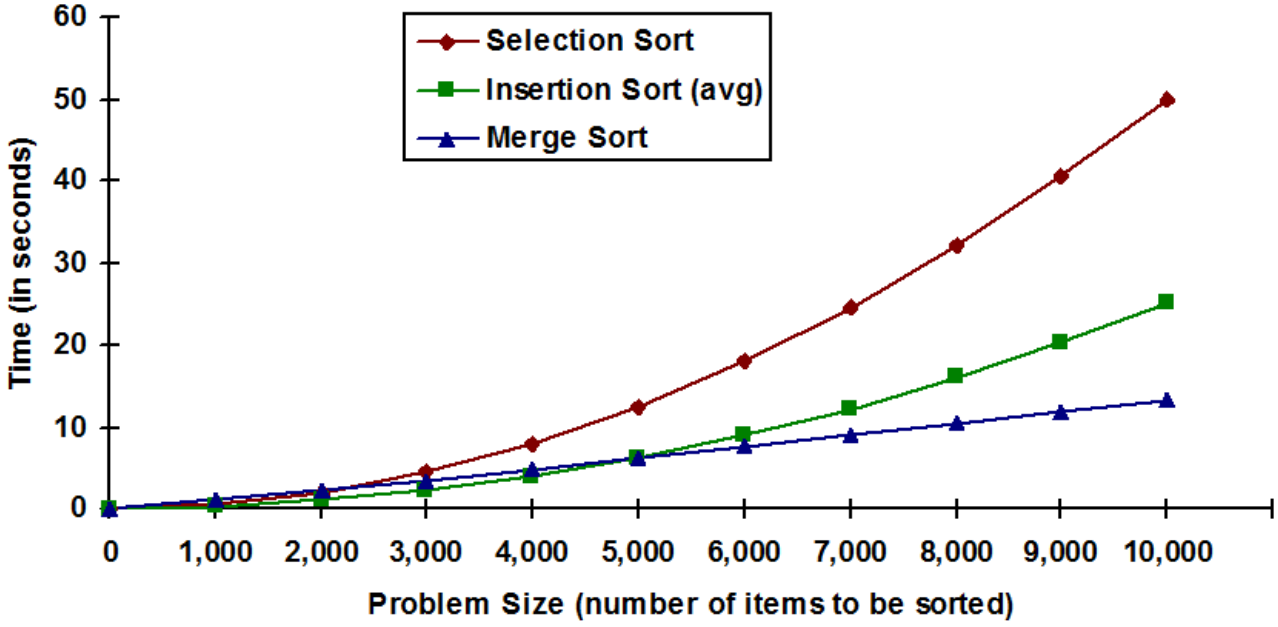


Figure 4.1. Comparison of sorting algorithms (this figure just a example)