

**Gebze Technical University
Computer Engineering**

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

**Burak Özdemir
141044027**

FatmaNur Esirci

1 Double Hashing Map

1.1 Pseudocode and Explanation

Public class DoubleHashMap<K,V> implements Map<K,V>

+public static class HashEntry<K,V>

-private int tableSize

-private int size

-private HashEntry<K,V> table;

-private int primeSize;

+public V put(K key,V val):

 If table size is equal to tableSize

 Print "table full"

 Initialize hash1 and assign hashFirst(key)

 Initialize hash2 and assign hashSecond(key)

 While hash1 of table is not equal to null

 Add hash2 into hash1

 Add hash1 mod tableSize into hash1

 End of while

 Set hash1 of table to new HashEntry

 Add one to size

 Return value of hash1 of table

➔ hash1 ve hash2 degerleri hesaplanır . tabloda null görülene kadar ilerlenir ve set yapılır .

+public V remove (Object key):

 Initialize hash1 and assign hashFirst(key)

 Initialize hash2 and assign hashSecond(key)

 Initialize val

 While (hash1 of table is not equal to null) and (key is not equal to key of hash1 of table)

 Add hash2 to hash1

 Add hash1 mod tableSize to hash1

 Set value of hash1 of table to val

 Set null to hash1 of table

 Extract one to size

 Return val

➔ Her defasında hash2 hash1 e eklenerek silinecek olan key degeri tabloda aranır

+public void clear()

 Set zero to size

 For set zero to i; i smaller than tableSize; add one to i

 Set null to i of table

 End of for

+public V get(Object key)

 Initialize hash1 and assign hashFirst(key)

 Initialize hash2 and assign hashSecond(key)

 While (hash1 of table is not equal to null) and (key is not equal to key of hash1 of table)

 Add hash2 to hash1

 Add hash1 mod tableSize to hash1

Return value of hash1 of table

→ Dongu içinde hash2 hash1 e eklenir ve mod alınarak sürekli tekrarlanır işlem . eslesme oldgunda return edilir

-private int hashFirst(object x)

Initialize hashVal and set hashCode of x

Set hash1 mod tablesize to hash1

If hashval is smaller than zero

Add tablesize to hash1

Return hashVal

→ objenin ilk hashdegeri return edilir

-private int hashSecond(object x)

Initialize hashVal and set hashCode of x

Set hash1 mod tablesize to hash1

If hashval is smaller than zero

Add tablesize to hash1

Return primeSize – hashVal mod primeSize

→ objenin ikinci hash degeri return edilir

1.2 Test Cases

İlk obje için size 5 tir. 5 tane eleman put edilmıştır . Ve diger get ve remove metodları denenmiştir.

2. obje için size 3 secilmiştir yine farklı 3 eleman put edilip diger metodlar denenmiştir. (Basarılı)

2 Recursive Hashing Set

2.1 Pseudocode and Explanation

Public class myRecursiveHashSet<E> implements Set<E>

/*Multidimensional buckets yapısı ile recursive hash saglanmıştır*/

/*Diger table size 10 ile sabitlenmiştir.*/

-private static class Entry<E>

-private Entry[][] buckets

-private int size

-private hashFunction(int hashCode)

Initialize index and set hashCode

If index smaller than zero

Decrease one to index and set index

Return index mod buckets.length

→ objenin hashCode unu alarak tablesize a gore hashdegerini return eder.

+public boolean contains(Object elem)

Initialize indY and set hashFunction(hashcode of elem)

Return helperContains(0,indy,elem)

-private boolean helperContains(int x,int y,object elem)

If key of buckets[y][x] is equal to elem

```

        Return true
    If bucket[y][x] is equal to null
        Return false
    If indexY of bucket[y][x] is equal to y
        Return helperContains(x+1,y,elem)
    Else
        Return helperContains(x,y+1,elem)

```

➔ Recursive şekilde hashFunction metodunu kullanarak objenin içinde olup olmadığını kontrol eder

```

+public boolean remove(Object o)
    Initialize index and set hashFunction(hashCode() of o)
    Return helperRemove(0,index,o)
-private boolean helperRemove(int x,int y,object elem)
    If bucket[y][x] is equal to null
        Return false
    If key of bucket[y][x] is equal to elem
        Initialize newBuckets and set new Entry[bucket.length][10]
        For set zero to i; i smaller than buckets.length; add one to i
            For set zero to j; j smaller than length of buckets[i]; add one to j
                If buckets[i][j] is not equal to null
                    If key of buckets[i][j] is equal to elem
                        Else
                            Set buckets[i][j] to newBuckets[i][j]
            End for
        Endfor
    Set newbuckets to buckets
    Decrease one to size
    Return true

```

➔ Recursive şekilde hashFunction kullanarak eğerki obje içinde varsa remove eder yoksa false return eder.

```

+public boolean add(Object elem)
    Initialize indexy and set hashFunction(hashCode() of elem)
    Return helperAdd(0,indexy,elem)
-private boolean helperAdd(int x,int y,object elem)
    If buckets[y][x] is not equal to null
        If key of buckets[y][x] is equal to elem
            Return false
    If buckets[y][s] is equal to null
        Initialize entry and set new Entry(elem,x,y)
        Set entry to buckets[y][x]
        Add one to size
        Return true
    If indexy of buckets[y][x] is equal to y
        Return helperAdd(x+1,y,elem)
    Else
        Return helperAdd(x,y+1,elem)

```

➔ Recursive şekilde hashFunction metodunu kullanarak objeyi uygun yere ekleme yapar . Başarılı durumda true aksi durumda false return eder.

2.2 Test Cases

Obje 1 size 3 ile belirlenmiştir .Sırası ile elemanlar eklenmiştir . Size 1 geçme durumunda hashFunctionun degerine gore sanki başka bir tablo oluşturmuş gibi arrayin 2. Boyutuna eklenerek devam eder . Aynı eleman ekleme durumunda false return edilir .

Obje 2 size yine 3 seçilip farklı elemanlar ile denenmiştir.Obje 1 in tum testleri bu objeyede yapılmıştır.(Basarılı)

3 Sorting Algorithms

3.1 MergeSort with DoubleLinkedList

3.1.1 Pseudocode and Explanation

Public class MergeSortDoubleLinkedList

-private static class Node

-private static Node head

➔ Sınıf DoubleLinkedList ozelligi için Node sınıfını kullanır ve yapının bas kısmını head ile elinde tutar.

+public Node mergeSort(Node node)

 If (node is equal)or(next of node is equal null)

 Return node

 Initialize second and set split(node)

 Set mergeSort(node) to node

 Set mergeSort(second) to second

 Return merge(node,second)

➔ listeyi en kucuk birimlerine kadar split metodu ile ayırır daha sonra merge metoduna vererek sıralamalı bir şekilde birleştirir.

-private Node split(Node head)

 Initialize fast and set head

 Initialize slow and set head

 While (next of fast is not equal to null) and (next of next of fast is not equal to null)

 Set next of next of fast to fast

 Set next of slow to slow

 Initialize temp and set next of slow

 Set null to next of slow

 Return temp

➔ Verilen node dan itibaren listeyi 2 parçaya ayırır

-private Node merge (Node first,Node second)

 If first is equal to null

 Return second

 If second is equal to null

 Return first

 If data of first smaller than data of second

 Set merge(next of first,second) to next of first

 Set first to prev of next of first

 Set null to prev of first

 Return first

Else

Set merge(first,next of second) to next of second
Set first to prev of next of first
Set null to prev of second
Return second

→ Verilen iki linkedlist yapısının elemanlarını karşılaştırarak sıralı tek bir liste return eder.

3.1.2 Average Run Time Analysis

$O(n \log n)$ zaman alır . Listeyi bölerken $\log n$ zamanda bölme işlemi yapılır. En küçük birime kadar bölünceği için her türlü $\log n$ olur daha sonra birleştirirken her eleman için kontrol edilme işlemi yapılır buda n zaman alır . bölme işlemi sırasında karşılaştırma olduğu için $\log n * n$ zaman alır average case durumunda.

3.1.3 Worst-case Performance Analysis

Algoritma worst case , average case ve best case de de en küçük elemana kadar bölme ve her defasında tüm elemanları her türlü karşılaştıracığı için yine $O(n \log n)$ zaman alır .

3.2 MergeSort

3.2.1 Average Run Time Analysis

$O(n \log n)$ zaman alır . Listeyi bölerken $\log n$ zamanda bölme işlemi yapılır. En küçük birime kadar bölünceği için her türlü $\log n$ olur daha sonra birleştirirken her eleman için kontrol edilme işlemi yapılır buda n zaman alır . Bölme işlemi sırasında karşılaştırma olduğu için $\log n * n$ zaman alır average case durumunda.

3.2.2 Worst-case Performance Analysis

Algoritma worst case , average case ve best case de de en küçük elemana kadar bölme ve her defasında tüm elemanları her türlü karşılaştıracığı için yine $O(n \log n)$ zaman alır .

3.3 Insertion Sort

3.3.1 Average Run Time Analysis

Sıra ile seçilen her eleman yeni listenin her elemanı ile her durumda karşılaştırma yapılacağı için $n * (n+1)/2$ karşılaştırma yapacağı için $O(n^2)$ zaman karmaşıklığı olacaktır.

3.3.2 Worst-case Performance Analysis

Worst-case durumu insertion sort da elemanlar ters sıralı olduğu durumda gerçekleşir . Yine average case de olduğu gibi her eleman yeni listenin her elemanı ile karşılaştıracığından $O(n^2)$ complexity değeri çıkar .

3.4 Quick Sort

3.4.1 Average Run Time Analysis

Her satırdaki her elemana bakar eğer n tane sayı varsa n tane sayıya bakılır .Adım sayısı $\log n$ olur .Ortalama durumda elemanların pivottan küçük mü büyük mü olacağı kestirmek mümkün

degildir o yüzden zaman karmasklığı $O(n \log n)$ olur .

3.4.2 Wort-case Performance Analysis

Her satırdaki her elemana bakar eger n tane sayı varsa n tane sayıya bakılır.En kotu durumda butun elemanlar eger pivottan kucuk ise elemanlar ıkiye bölünmez . Her zaman pivot bır soldaki secılcegi ıcın n-1 tane sayı olacaktır .Ve bu ıstem tekralanır . böylece $O(n^2)$ karmaşıklıkđı elde edılır.

3.5 Heap Sort

3.5.1 Average Run Time Analysis

Bu algoritma her eleman ıcın bir kere silme metodunu çağıracağı ıcın(her eleman ıcın oldgundan n-1 kere) ve her silme operasyonu logn zaman alacağından algoritmanın zaman karmaşıklıkđı $O(n \log n)$ olacaktır .

3.5.2 Wort-case Performance Analysis

$O(n \log n)$ karmaşıklıkđına sahip olacaktır . Cunku her türlü ağaç yukseklığı logn karmasıklıkđına sahip olacaktır ve worstcase durumunda tek bır dal seklinde düşünürsek n-1 tane karsılastrma olacaktır ve yine $n * \log n$ karmasıklıkđını elde etmis olacağız.

4 Comparison the Analysis Results

/*Bazı eleman sayılı testlerde bazı algoritmalar stack hatası verdiđi ıcın o algoritmalar alınmamıştır*/

Q1:

```
public static void main(String args[]){
    try{
        System.out.println(".....OBJE 1:");
        DoubleHashMap<String,Integer> obj1=new DoubleHashMap<> ( 10, 5);
        obj1.put("burak",22);
        obj1.printHashTable();
        obj1.put("cagla",21);
        obj1.printHashTable();
        obj1.put("tahir",23);
        obj1.printHashTable();
        obj1.put("bahar",20);
        obj1.printHashTable();
        obj1.put("tuana",22);
        System.out.println("obj1.get(tuana)="+obj1.get("tuana").toString());
        obj1.remove("cagla");
        System.out.println("obj1.remove(cagla)");
        obj1.printHashTable();
        //System.out.println("obj1.get(cagla)="+obj1.get("cagla"));
        System.out.println(".....OBJE 2");
        DoubleHashMap<String,Integer> obj2=new DoubleHashMap<> ( 10, 3);
        obj2.put("a",1);
        obj2.printHashTable();
        obj2.put("b",2);
        obj2.printHashTable();
        obj2.put("c",3);
        obj2.printHashTable();
        obj2.put("c",1);
        obj2.remove("c");
        obj2.printHashTable();
        obj2.remove("c");
        obj2.printHashTable();
    }catch (Exception e){
        System.out.println(e.getMessage());
    }
}
```

Q1 Main Test

Q2:

```

myRecursiveHashSet<String> obj1=new myRecursiveHashSet<> (cap
obj1.add("b");System.out.println("obj1.add(b)");
System.out.println(obj1.toString());
obj1.add("c");System.out.println("obj1.add(c)");
System.out.println(obj1.toString());
obj1.add("d");System.out.println("obj1.add(d)");
System.out.println(obj1.toString());
obj1.add("e");System.out.println("obj1.add(e)");
System.out.println(obj1.toString());
obj1.add("x");System.out.println("obj1.add(x)");
System.out.println(obj1.toString());
obj1.add("y");System.out.println("obj1.add(y)");
System.out.println(obj1.toString());
obj1.add("h");System.out.println("obj1.add(h)");
System.out.println(obj1.toString());
obj1.add("g");System.out.println("obj1.add(g)");
System.out.println(obj1.toString());
obj1.remove(0);System.out.println("obj1.remove(g)");
System.out.println(obj1.toString());
myRecursiveHashSet<String> obj2=new myRecursiveHashSet<> (cap
obj2.add(1);System.out.println("obj2.add(1)");
System.out.println(obj2.toString());
obj2.add(2);System.out.println("obj2.add(2)");
System.out.println(obj2.toString());
obj2.add(3);System.out.println("obj2.add(3)");
System.out.println(obj2.toString());
obj2.add(1);System.out.println("obj2.add(1)");
System.out.println(obj2.toString());
obj2.add(4);System.out.println("obj2.add(4)");
System.out.println(obj2.toString());
obj2.add(5);System.out.println("obj2.add(5)");
System.out.println(obj2.toString());
obj2.add(6);System.out.println("obj2.add(6)");
System.out.println(obj2.toString());
  
```

```

obj2.add(2)
0:
1:1
2:2
obj2.add(3)
0:3
1:1
2:2
obj2.add(1)
0:3
1:1
2:2
obj2.add(4)
0:3
1:1 4
2:2
obj2.add(5)
0:3
1:1 4
2:2 5
obj2.add(6)
0:3 6
1:1 4
2:5
obj2.remove(2)
0:3 6
1:1 4
2:5
  
```

Q2 Main Test

Q4:

```

Random rand = new Random();
int m;
long startTime;
long endTime;

long running_time_MergeSort;
long running_time_InsertionSort;
long running_time_QuickSort;
long running_time_HeapSort;
long running_time_MergeSortDoubleLinkedList;

int size1 = 10;
int size2 = 100;
int size3 = 1000;
int size4 = 5000;
int size5 = 10000;
int size6 = 30000;
int size7 = 60000;
int size8 = 90000;
int size9 = 100000;
int size10 = 200000;

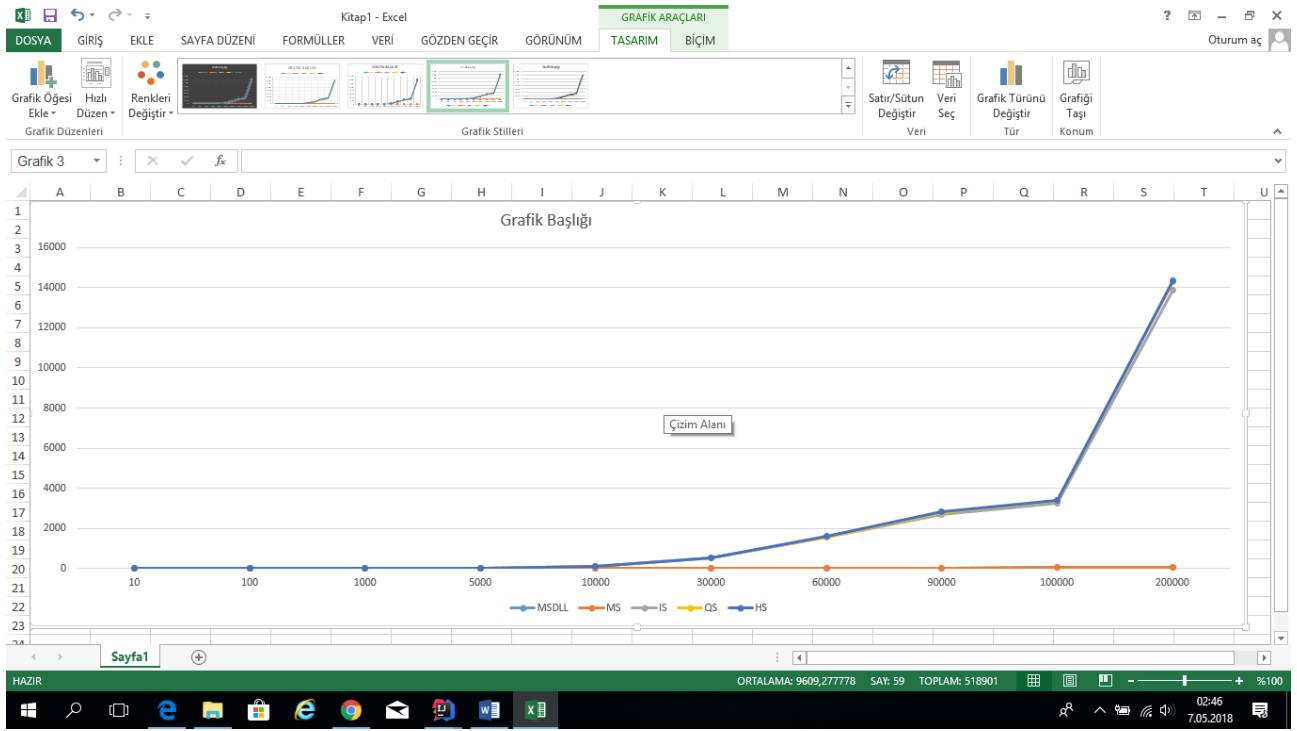
int array1[] = new int[size1];
int array2[] = new int[size2];
int array3[] = new int[size3];
int array4[] = new int[size4];
int array5[] = new int[size5];
int array6[] = new int[size6];
int array7[] = new int[size7];
int array8[] = new int[size8];
int array9[] = new int[size9];
int array10[] = new int[size10];

//SIZE:10
  
```

```

MergeSort(10000): 1
InsertionSort(5000): 17
QuickSort(5000): 0
HeapSort(5000): 1
MergeSort(10000): 7
InsertionSort(10000): 88
QuickSort(10000): 0
HeapSort(10000): 2
MergeSort(30000): 13
InsertionSort(30000): 498
QuickSort(30000): 17
HeapSort(30000): 9
MergeSort(60000): 20
InsertionSort(60000): 1516
QuickSort(60000): 40
HeapSort(60000): 13
MergeSort(90000): 20
InsertionSort(90000): 2668
QuickSort(90000): 100
HeapSort(90000): 26
MergeSort(100000): 31
InsertionSort(100000): 3237
QuickSort(100000): 99
HeapSort(100000): 24
MergeSort(200000): 48
InsertionSort(200000): 13839
QuickSort(200000): 426
HeapSort(200000): 48
  
```

Q4 main Test



Q4-Grafik

Q5:

hw5 [C:\Users\ZEYNEP SU\Desktop\burakshws\141044027\141044027\burk\cse222.hw5] - .../sr...

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

WorstCasePerformanceAnalysis.java

```

339 public static class HeapSort
340 {
341     ...
342 }
343
344 public static class GFG
345 {
346     ...
347 }
348
349 public static void main(String args[]) {
350     Random rand = new Random();
351     int n;
352     long startTime;
353     long endTime;
354
355     long running_time_MergeSort;
356     long running_time_InsertionSort;
357     long running_time_QuickSort;
358     long running_time_HeapSort;
359     long running_time_MergeSortDoubleLinkedList;
360
361     int size1 = 10;
362     int size2 = 100;
363     int size3 = 5000;
364     int size4 = 10000;
365
366     int array1[] = new int [size1];
367     int array2[] = new int [size2];
368     int array3[] = new int [size3];
369     int array4[] = new int [size4];
370
371     ///////////////////////////////////////////////////SIZE:10
372     System.out.println(".....SIZE:10 ...");
373     //MergeSortDoubleLinkedList(10): 0
374
375     ///////////////////////////////////////////////////SIZE:100
376     MergeSortDoubleLinkedList(100): 0
377     MergeSort(100): 0
378     InsertionSort(100): 0
379     QuickSort(100): 0
380     HeapSort(100): 0
381
382     ///////////////////////////////////////////////////SIZE:5000
383     MergeSortDoubleLinkedList(5000): 2
384     MergeSort(5000): 1
385     InsertionSort(5000): 29
386     QuickSort(5000): 17
387     HeapSort(5000): 1
388
389     ///////////////////////////////////////////////////SIZE:10000
390     MergeSort(10000): 1
391     InsertionSort(10000): 66
392     HeapSort(10000): 1
393
394     Process finished with exit code 0

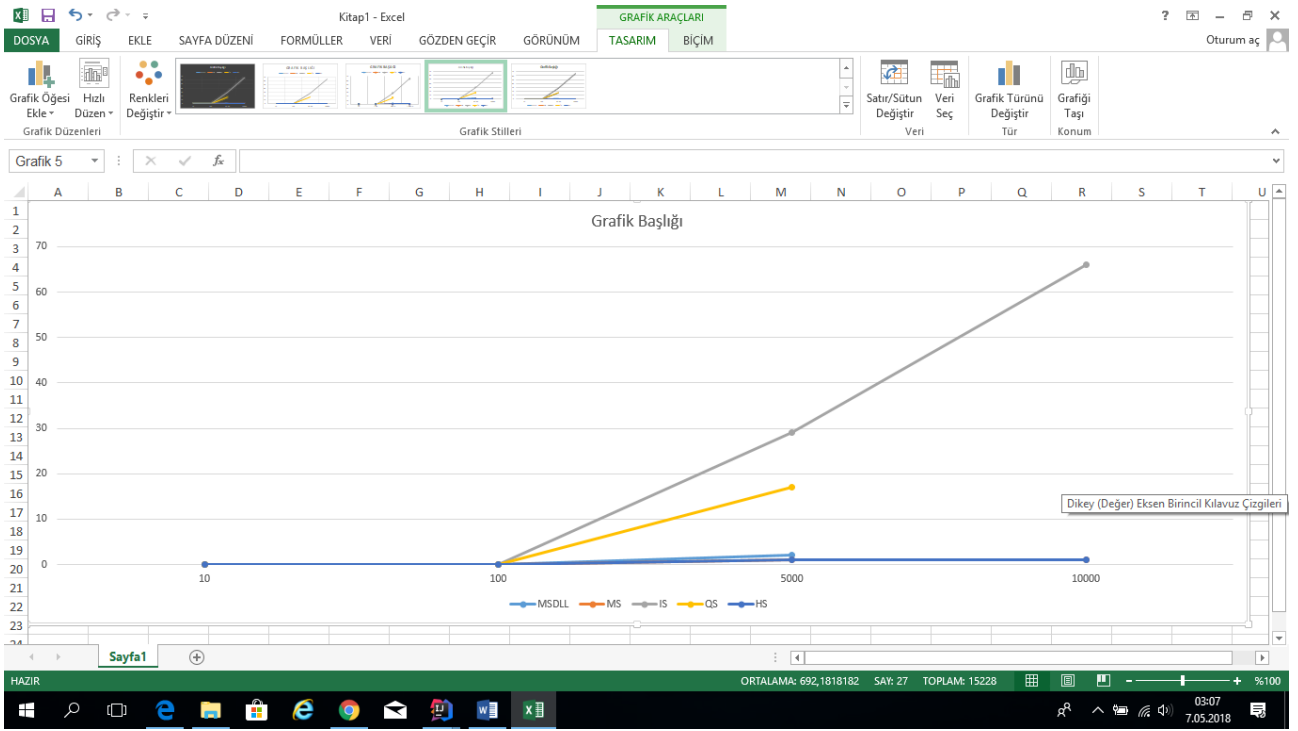
```

WorstCasePerformanceAnalysis - main()

489:37 CRLF UTF-8

02:48 7.05.2018

Q5-Main Test



Q5-Grafik