**Gebze Technical University**
**Computer Engineering**

# CSE 222 - 2018 Spring

# HOMEWORK 5 REPORT

**DENİZ BABAT**
**131044013**

Course Assistant:

# 1 Double Hashing Map

Double hashing map, collision olan durumları çözmek için, probe'lar arasında başka bir hash fonksiyonu ile hesaplandığı bir algoritmadır.

## 1.1 Pseudocode and Explanation

The using structures are that map interface ve Entry class'ını kullandım.

Pseudocode;

- index = hash%table boyutu
- if index < 0
    - index += tablo boyutu
- while tablo[index] null değilse ve key != tablo[index].key
    - index = (index+hash2(key))%table.length
- return index

## 1.2 Test Cases

```
1. HASH SQUENCE
aspx
java
c++
c
aspx
ruby
coffee
.net
sql
oracle
sunny
sony

1. HASH SQUENCE
python
groovy
go
ajax
javascript
Arduino
swift
html
php
c#
scalla
d
```

# 2 Recursive Hashing Set

Bu kısımı yazmaya çalıştım fakat pdf te istenilen yapıyı tam olarak anlamadım.

## 2.1 Pseudocode and Explanation

## 2.2 Test Cases

Try this code least 2 different hash table size and 2 different sequence of keys. Report all of situations.

# 3 Sorting Algortihms

## 3.1 MergeSort with DoubleLinkedList

Bu kısımda double linkedlist ile merge algoritması geliştirilmiştir.

### 3.1.1 Pseudocode and Explanation

Kullanılan yapı Node class'ı olmuştur. Node class'ı kullanılarak bir eleman dizisi oluşturuldu. Daha sonra bu yapı iki pointer ile bir önünü ve bir arkasını işaret eder. Node class'ında bulunan member'lar şunlardır; data, next ve prev member'larıdır.

Merge algoritması şöyle geliştirilmiştir. Total de üç fonksiyon vardır. Bunlar mergesort, divide ve merge metodlarıdır.

MERGESORT:

1. Merge(node local)
2. If local = null and local.next = null
    1. Return local
3. Node second  = divide(local)
4. local = mergeSort(local);
5. second = mergeSort(second);
6. return merge(local, second);

DIVIDE:

1. Node divide(Node head)
2. Node first = head
3. Node second = head
4. while first.next != null && first.next.next != null
    1. first = first.next.next

        2. second = second.next
    5. Node temp = second.next
    6. second.next = null
    7. return temp

MERGE:

1. Node merge(Node first, Node second)

2. if first == null // If first linked list is empty
    1. return second
3. if second == null // If second linked list is empty
    1. return first

4. if first.data < second.data // küçük değeri topla
    1. first.next = merge(first.next, second)
    2. first.next.prev = first
    3. first.prev = null
    4. return first
5. else//büyüğü topla
    1. second.next = merge(first, second.next)
    2. second.next.prev = second
    3. second.prev = null
    4. return second

pseudocode yukarıdaki gibidir.

#### 3.1.2 Average Run Time Analysis

T(n) = O(nlog(n)) dir.

#### 3.1.3 Wort-case Performance Analysis

T(n) = O(nlog(n ) dir.

### MergeSort

#### 3.1.4 Average Run Time Analysis

T(n) = O(nlog(n)) dir

#### 3.1.5 Wort-case Performance Analysis

T(n) = O(nlog(n)) dir

## 3.2 Insertion Sort

#### 3.2.1 Average Run Time Analysis

T(n) = O(n) dir.

### 3.2.2 Wort-case Performance Analysis

$T(n) = O(n^2)$ dir.

## 3.3 Quick Sort

### 3.3.1 Average Run Time Analysis

$T(n) = O(nlog(n))$ dir

### 3.3.2 Wort-case Performance Analysis

$T(n) = O(n^2)$ dir.

## 3.4 Heap Sort

### 3.4.1 Average Run Time Analysis

$T(n) = O(n)$ dir.

### 3.4.2 Wort-case Performance Analysis

$T(n) = O(nlog(n))$ dir.

# 4 Comparison the Analysis Results

Aşağıdaki algoritmaların zaman hesaplamaları nanosaniye'ye göre hesaplanmıştır.