GIT Department of Computer Engineering

CSE 222/505 - Spring 2021

Homework 2 Report

Muhammed Bedir ULUCAY

1901042697

Branch Class Time Complexity :

```
/**<p>Adding employee next free space in empArr</p>*/
public boolean addEmployee(Employee employee) {
    return emps.add(employee);
}
```

T(n) -> amortized constant Θ(1)

If array is full. Then you need to reallocate the array > Θ(n)

```
/**<p>Checking employee account is in customer array data</p>*/
public Employee checkEmployeeAccount(Employee employee) {

    for(int i=0; i<emps.getSize(); ++i) {
        if(emps.get(i).getEmail().equals(employee.getEmail())
                && emps.get(i).getPassword().equals(employee.getPassword()))
            return emps.get(i);
    }
    return null;
}
```

T(n) -> Θ(n)

ArrayList get -> constant but we are compare one by one for each

```
public Furniture getProduct(int index) {
    if(index > getProductNumber())
        throw new IndexOutOfBoundsException(Integer.toString(index));

    HIterator<Furniture> HIter = furnitures.iterator();

    for(int i=0; i<index / furnitures.getMaxNumber(); ++i)
        HIter.next();

    Iterator<Furniture> AIter = HIter.next().iterator();


    int nth = index - ((index / furnitures.getMaxNumber()) * 10);
    for(int i=0;i<nth; ++i)
        AIter.next();

    return AIter.next();
}
```

T(n,n/m) -> Θ(n + m)

N = Number of node in Hybrid List(big steps)

M = Max_Number(small steps)

```java
public int getEmployeeIndex(Employee e) {

    Iterator<Employee> iter = emps.iterator();
    int counter = 0;

    while(iter.hasNext()) {
        Employee emp = iter.next();
        if(emp.equals(e))
            return counter;
        counter++;
    }
    return -1;
}
```

$T(n) \rightarrow \Theta(n)$

Linear Search $\Theta(n)$ others $\Theta(1)$

```java
/**<p>Query furniture is in branch or not using linear search</p>*/
public boolean queryProduct(Furniture f) {
    return furnitures.indexOf(f) != -1;
}
```

$T(n,m) \rightarrow \Theta(n*m)$

N = Number of node in Hybrid List(big steps)

M = Max_Number(small steps)

```java
/**<p>Removing product in specific index</p>*/
public Furniture removeProduct(int index) {
    return furnitures.remove(index);
}
```

Hyrid remove -> Time Complexity -> $\Theta((n / m) + m)$

N = Number of node in Hybrid List(big steps)

M = Max_Number(small steps)

```java
public boolean addProduct(Furniture f) {
    return furnitures.add(f);
}
```

$T(n,m) \rightarrow \Theta(n)$

N = Number of node in Hybrid List(big steps)

M = Max_Number(small steps)

```java
/**<p>Selling a product to the parameter customer using branch function</p>*/
public boolean sellProduct(Furniture furniture, Customer customer) {

    if(queryProduct(furniture)) {
        removeProduct(indexOfFurniture(furniture));
        customer.buyProduct(furniture);
        return true;
    }
    return false;
}
```

T(n) -> Θ(n)

Query -> Θ(n*m)

IndexOfFurniture -> Θ(n * m)

Remove -> Θ((n / m) + m)

Buy -> Θ(1)

N = Number of node in Hybrid List(big steps)

M = Max_Number(small steps)

```java
public int indexOfFurniture(Furniture f) {
    return furnitures.indexOf(f);
}
```

Hybrid indexOf T(n) -> Θ(n * m)

N = Number of node in Hybrid List(big steps)

M = Max_Number(small steps)

```java
/**<p>Removing employee using array shifting method</p>*/
public Employee removeEmployee(int index) {
    return emps.remove(index);
}
```

T(n) -> Θ(n)

Array List remove

Data Class Time Complexity :

```java
public boolean addAdmin(Admin admin) {
    return adminList.add(admin);
}
```

T(n) -> amortized constant Θ(1)

If array is full. Then you need to reallocate the array > Θ(n)

```java
public boolean addBranch(Branch newBranch) {
    return  branchList.add(newBranch);
}
```

T(n) -> Θ(n)

Linked List adding new element end of the list

```java
public void addEmployee(Branch branch, Employee employee) {
    employeeList.add(employee);
    branch.addEmployee(employee);
}
```

T(n) -> amortized constant Θ(1)

Both line has arrayList<Emp> add method . If array is full. Then you need to reallocate the array > Θ(n)

```java
public Admin checkAdminAccount(Admin admin) {
    Iterator<Admin> iter = adminList.iterator();

    while(iter.hasNext()) {

        Admin ad = iter.next();
        if(ad.getEmail().equals(admin.getEmail()) && ad.getPassword().equals(admin.getPassword()))
            return ad;
    }
    return null;
}
```

T(n) -> Θ(n)

Linear Search comparing one by one for each

```java
public Customer checkCustomerAccount(Customer customer) {
    Iterator<Customer> iter = customerList.iterator();

    while(iter.hasNext()) {

        Customer cu = iter.next();
        if(cu.getEmail().equals(customer.getEmail()) && cu.getPassword().equals(customer.getPassword()))
            return cu;
    }
    return null;
}
```

T(n) -> Θ(n)

Linear Search comparing one by one for each

```java
public Employee checkEmployeeAccount(Employee employee) {
    Iterator<Employee> iter = employeeList.iterator();

    while(iter.hasNext()) {

        Employee emp = iter.next();
        if(emp.getEmail().equals(employee.getEmail()) && emp.getPassword().equals(employee.getPassword()))
            return emp;
    }
    return null;
}
```

T(n) -> Θ(n)

Linear Search comparing one by one for each

```java
public int indexOfCustomer(Customer c) {
    return customerList.indexOf(c);
}
```

T(n) -> Θ(n)

Array List indexOf linear search -> Θ(n)

```java
public void displayCustomerShopList(int customerId) {
    Iterator<Customer> iter = customerList.iterator();

    while(iter.hasNext()) {
        Customer c = iter.next();
        if(c.getId() == customerId) {
            c.displayShoppingList();
            return;
        }
    }
    System.out.println("Customer id is not defined");
}
```

T(n,m) -> O(n * m)

If customer not in the list this time T(n) -> Θ(n) but we have to consider worst case

N = Number of customer

M = Lenght of customer shop list

```java
public void displayProducts() {
    Iterator<Branch> iter = branchList.iterator();

    while(iter.hasNext()) {
        Branch br = iter.next();

        System.out.println("Branch " + br.getId());
        br.displayProducts();
    }
}
```

T(n,m) -> O(n*m)

N = Number of Branch

M = Number of furniture in branch

```java
public Branch getBranch(int index) {
    return branchList.get(index);
}
```

T(n) -> Θ(n)

Linked List get method -> Θ(n)

```java
public Customer getCustomer(int id) {
    return customerList.get(id);
}
```

T(n) -> Θ(1)

Array List get method -> Θ(1)

```java
public Admin makeAdmin(){
    return objectCreator.makeAdmin(this);
}

/**<p>Making employee and return it</p>*/
public Employee makeEmployee(){
    return objectCreator.makeEmployee(this);
}

public Customer makeCustomer() {
    return objectCreator.makeCustomer(this);
}

public Branch makeBranch() {
    return objectCreator.makeBranch();
}

public Furniture makeProduct() {
    return objectCreator.makeProduct();
}
```

T(n) -> Θ(1)

Functions just doing constant time operations take some data and creating an object and return it

```java
public Branch queryProduct(Furniture obj) {

    Iterator<Branch> iter = branchList.iterator();

    while(iter.hasNext()) {

        Branch br = iter.next();
        if(br.queryProduct(obj))
            return br;
    }
    return null;
}
```

T(n,m,l) -> Θ(n*m*l)

N = Number of Branch

M = Number of node in Hybrid List in branch

L = Max_Number of arrayList in Linked List in HybridList

```java
public Branch removeBranch(int index) {
    return branchList.remove(index);
}
```

T(n) -> Θ(n)

Linked List remove method -> Θ(n)

```java
public Branch removeBranch(Branch branch) {
    return branchList.remove(branch);
}
```

T(n) -> Θ(n)

Linked List remove method -> Θ(n)

```java
public Employee removeEmployee(Branch branch, Employee employee) {
    return branch.removeEmployee(branch.getEmployeeIndex(employee));
}
```

T(n) -> Θ(n)

GetEmpIndex doing Linear Seacrh -> Θ(n)

RemoveEmployee doing also linear -> Θ(n)

Θ(n) + Θ(n) = Θ(n)

```java
public void displayBranchs() {
    System.out.println(branchList);
}
```

T(n) -> Θ(n)

```java
public void displayCustomers() {
    System.out.println(customerList);
}
```

T(n) -> Θ(n)

Admin Class Time Complexity :

```
/**<p>Adding new Branch to company </p>*/
public boolean addBranch(Branch newBranch){
    return companyData.addBranch(newBranch);
}
```

T(n) -> Θ(n)

-> Data.addBranch()-> LinkedList.add()

```
/**<p>Adding taken employee adding taking branch</p>*/
public void addEmployee(Branch branch,Employee employee) {
    companyData.addEmployee(branch,employee);
}
```

T(n) -> amortized constant Θ(1)

-> Data.addEmp() -> ArrayList.add() = Θ(1)

```
public void displayBranchs() {
    companyData.displayBranchs();
}
```

T(n) -> Θ(n)

N = Number of Branch

```
public void displayEmployee() {

    Iterator<Branch> branchs = companyData.getBranchs().iterator();

    while(branchs.hasNext()) {
        Branch br = branchs.next();
        System.out.println(br);
        br.displayEmployees();
    }
}
```

T(n,m) -> Θ(n * m)

N = Number of Branch

M = Number of employee in Branch

```java
/**<p>It return branch in array wanted index</p>*/
public Branch getBranch(int index) {
    return companyData.getBranch(index);
}
```

T(n) -> Θ(n)

Data.getBranch() -> LinkedList.getNode() = Θ(n)

```java
public static void getMessage(String message) {
    mailBox = mailBox + "\n" + message;
}
```

T(n) -> Θ(n)

String summation

```java
/**<p>Make Branch and returning it</p>*/
public Branch makeBranch() {
    return companyData.makeBranch();
}

/**<p>Make Employee and return it</p>*/
public Employee makeEmployee() {
    return companyData.makeEmployee();
}
```

T(n) -> Θ(1)

Data.make -> ObjectCreator.make = Θ(1)

```java
/**<p>Query taken product in all branch using linear search</p>*/
public boolean queryProduct(Furniture furniture) {
    Iterator<Branch> br = companyData.getBranchs().iterator();
    while(br.hasNext()) {
        if(br.next().queryProduct(furniture))
            return true;
    }
    return false;
}
```

T(n,m,l) -> Θ(n * m * l)

N = Number of Branch

M = Number of node in Hybrid List in branch

L = Max_Number of arrayList in Linked List in HybridList

```
/**<p>Removing Branch from company</p>*/
public Branch removeBranch(int index) {
    return companyData.removeBranch(index);
}
```

T(n) -> Θ(n)

Data.removeBranch -> LinkedList.remove() = Θ(n)

```
/**<p>Removing Branch in Level 1 using default branch address</p>*/
public Branch removeBranch(Branch branch) {
    return companyData.removeBranch(branch);
}
```

T(n) -> Θ(n)

Data.removeBranch -> LinkedList.remove() = Θ(n)

```
/**<p>Removing taken employee adding taking branch</p>*/
public Employee removeEmployee(Branch branch,Employee employee) {
    return companyData.removeEmployee(branch,employee);
}
```

T(n) -> Θ(n)

Data.removeBranch -> ArrayList.remove() = Θ(n)

Employee Class Time Complexity :

```
/**<p>Adding new customer to the system using dataLevel2 function</p>*/
public boolean addCustomer(Customer customer) {
    return empCompanyData.addCustomer(customer);
}
```

T(n) -> amortized constant time Θ(1)

DATA.addCustomer -> Array add

```
/**<p>Adding new product to the system using branch function</p>*/
public boolean addProduct(Furniture newFurniture) {
    return workBranch.addProduct(newFurniture);
}
```

T(n) -> Θ(n)

Data.addBranch -> Linked List add

```
/**<p>Display all saved customer in system</p>*/
public void displayCustomers(){
    empCompanyData.displayCustomers();
}
```

T(n) -> Θ(n)

N = Number of Customer

```
/**<p>Display a customer shop list</p>*/
public void displayCustomerShopList(int customerId){
    empCompanyData.displayCustomerShopList(customerId);
}
```

T(n,m) -> O(n * m)

N = Number of customer

M = Lenght of customer shop list

```
/**<p>Display furniture in working store</p>*/
public void displayProducts() {
    workBranch.displayProducts();
}
```

T(n,m) -> Θ (n*m)

Branch.display -> Θ(n*m)

N = Number of Branch

M = Number of furniture in branch

```
/**<p>Getting customer using id from database</p>*/
public Customer getCustomer(int id) {
    return empCompanyData.getCustomer(id);
}
```

T(n) -> Θ(n)

DATA.getCustomer -> it is doing linear search comparing one by one for each

N = Number of customer

```
/**<p>getting furniture in working store</p>*/
public Furniture getProduct(int index) {
    return workBranch.getProduct(index);
}
```

T(n,n/m) -> Θ(n + m)

N = Number of node in Hybrid List(big steps) in working branch

M = Max_Number(small steps) in working branch

```
/**<p>It is returning wanted customer index</p>*/
public int indexOfCustomer(Customer customer) {
    return empCompanyData.indexOfCustomer(customer);
}
```

T(n) -> Θ(n)

Data.indexOfCustomer -> Linear Search comparing one by one for each

N = Number of customer

```
/**<p>Make new product to add store</p>*/
public Furniture makeProduct() {
    return empCompanyData.makeProduct();
}
```

T(n) -> Θ(1)

```
/**<p>Query a product in working branch function</p>*/
public boolean queryProduct(Furniture obj){
    return  workBranch.queryProduct(obj);
}
```

T(n,m) -> Θ(n*m)

N = Number of node in Hybrid List(big steps) in branch

M = Max_Number(small steps) in branch

```
/**<p>Remove a furniture in working store</p>*/
public Furniture removeProduct(int index) {
    return workBranch.removeProduct(index);
}
```

Hyrid remove -> Time Complexity -> Θ((n / m) + m)

N = Number of node in Hybrid List(big steps)

M = Max_Number(small steps)

```
/**<p>Selling a product to the parameter customer using branch function</p>*/
public boolean sellProduct(Furniture furniture,Customer customer) {
    return workBranch.sellProduct(furniture, customer);
}
```

T(n) -> Θ(n)

Branch.sell -> Θ(n)

```
/**<p>Sending message to admin about out of stock</p>*/
public void sendMessageToAdmin(String message) {
    Admin.getMessage(message);
}
```

T(n) -> Θ(n)

Admin.get Message -> Θ(n) because there is string summurization

Customer Class Time Complexity :

```
/**<p>Add product customer shopping list</p>*/
public boolean buyProduct(Furniture bF) {
    return shopList.add(bF);
}
```

T(n) -> O(n)

N = number of node in hybrid list

M = max_Number size for arrayList

```
/**<p>Display all product in all store</p>*/
public void displayProducts() {
    customerData.displayProducts();
}
```

T(n) -> Θ(n*m)

-> DATA.displayProduct

N = number of node in hybrid list

M = max_Number size for arrayList

```java
/**<p>Printing  all buying products</p>*/
public void displayShoppingList() {
    System.out.println(getName() + " " + getSurname() + " shopping list :" );
    System.out.println(shopList);
}
```

T(n) -> Θ (n*m)

N = number of node in hybrid list

M = max_Number size for arrayList

```java
/**<p>Customer make online shop entering address and phone</p>*/
public void onlineShopping() {

    Scanner scan = new Scanner(System.in);
    String _adress = new String();
    String _phone = new String();

    System.out.println("Enter adress :");
    _adress = scan.nextLine();
    System.out.println("Enter phone :");
    _phone = scan.nextLine();

    adress = _adress;
    phone = _phone;
}
```

T(n) -> Θ(1)

Scan and print normally take some time but we can accept them as constant

```java
/**<p>Customer using linear search in all branch furniture</p>*/
public Branch queryProduct(Furniture obj) {
    return customerData.queryProduct(obj);
}
```

T(n,m,l) -> Θ(n*m*l)

-> DATA.queryProduct ->

N = Number of Branch

M = Number of node in Hybrid List in branch

L = Max_Number of arrayList in Linked List in HybridList