Depot

```java
public boolean addToDepot(Item item){
    if(item == null){
        throw(new IllegalArgumentException());
    }
    if(item.getQuantity() < 0)
        return false;

    ListIterator<Item> listiter = items.listIterator();
    Item check_item;
    boolean found = false;

    while(listiter.hasNext() && !found){
        check_item = listiter.next();
        if(check_item.equals(item)){
            check_item.setQuantity(check_item.getQuantity() + item.getQuantity());
            found = true;
        }
    }
    if(!found)
        listiter.add(item);

    return true;
}
```

O(n) ⇒ O(n)

O(1) ⇒ O(1)

O(1) ⇒ O(1)

O(n) ⇒ O(n)

O(1) ⇒ O(1)

O(1) ⇒ O(1)

O(1) ⇒ O(1)

O(1) ⇒ O(1)

O(1) ⇒ O(1)

O(n) because if item found loop will end.

```java
public boolean removeFromDepot(Item item){
    if(item == null){
        throw(new IllegalArgumentException());
    }
    ListIterator listIter = items.listIterator();
    while(listIter.hasNext()){
        Item check_item = (Item) listIter.next();
        if(check_item.equals(item)){
            if(check_item.getQuantity() > item.getQuantity()){
                check_item.setQuantity(check_item.getQuantity() - item.getQuantity());
                return true;
            }
            else if(check_item.getQuantity() == item.getQuantity()){
                listIter.remove();
                return true;
            }
            else{
                return false;
            }
        }
    }
    return false;
}
```

=> O(n)

=>O(1)

=>O(n)

O(1)

O(1)

O(1)

O(n)

O(n)

Because loop can break with return statements

m = order size

```java
public boolean removeWithOrder(Order order){
    boolean contains = true;
    for(int i = 0; i < order.getOrderCount() && contains; ++i){
        if(!isContain(order.getOrderfromLatest(i)))
            contains = false;
    }
    if(!contains)
        return false;

    for(int i = 0; i < order.getOrderCount(); ++i){
        removeFromDepot(order.getOrderfromLatest(i));
    }
    return true;
}
```

=> O(mn)

=>O(1)

O(m)

=>O(n)

=>O(1)

=>O(1)

O(m.n)

O(mn)

=>O(n)

O(m.n)

=> O(n)

```
public boolean isContain(Item item){
    if(item == null)
    {
        throw(new IllegalArgumentException());
    }

    ListIterator listIter = items.listIterator();
    while(listIter.hasNext()){
        Item check_item = (Item) listIter.next();
        if(check_item.equals(item) && check_item.getQuantity() >= item.getQuantity())
            return true;
    }
    return false;
}
```

O(n) (whole method); O(n) (while loop); => O(1) (next()); => O(1) (if condition); => O(1) (return true)

```
*/
public HybridList<Item> searchProduct(String name){
    HybridList<Item> founded = new HybridList<>();

    ListIterator listIter = items.listIterator();
    while(listIter.hasNext()) {
        Item check_item = (Item) listIter.next();
        if(check_item.getName().equals(name))
            founded.add(check_item);
    }
    return founded;
}
```

=> O(1) (founded.add); O(n) (while loop); O(n) (whole method)

Customer List (n is cust. count)

```java
public boolean addCustomer(Customer customer){
    if(customer == null){
        throw(new IllegalArgumentException());
    }
    if(!customers.contains(customer)){ => O(n)
        customers.add(customer); => O(1)
        return true;
    }
    return false;
}
```

O(n)

```java
public Customer getCustomerWithID(String ID){
    for(int i = 0; i < customers.size(); ++i){
        if(customers.get(i).getCustomerID().equals(ID)) => O(1)
            return customers.get(i); => O(1)
    }
    return null;
}
```

O(n) O(n)

```java
public Customer getCustomerWithMail(String mail){
    for(int i = 0; i < customers.size(); ++i){
        if(customers.get(i).getEmail().equals(mail)) => O(1)
            return customers.get(i); => O(1)
    }
    return null;
}
```

O(n) O(n)

Order Class

```
public boolean addOrder(Item item){
    if(item == null){
        throw(new IllegalArgumentException());
    }
    if(item.getQuantity() > 0){
        lastOrder.add(item);
        return true;
    }
    else{
        return false;
    }
}
```

lastOrder.add(item); => O(1)

if(item.getQuantity() > 0){ ... } => O(1)

O(1)

```
public boolean addFinishedOrder(Item item){
    if(item == null){
        throw(new IllegalArgumentException());
    }
    if(item.getQuantity() > 0){
        shipped.add(item);
        return true;
    }
    else
        return false;
}
```

if(item.getQuantity() > 0){ ... } => O(1)

O(1)

```
public boolean removeOrderfromLatest(int index){
    if(index >= 0 && index < lastOrder.size()){
        ListIterator listIter = lastOrder.listIterator(lastOrder.size());
        for(int i = 0; i < index; ++i)
            listIter.previous();
        Item ord = (Item) listIter.previous();
        shipped.add(ord);
        listIter.remove();
        return true;
    }
    else
        return false;
}
```

O(1) <=

for(int i = 0; i < index; ++i) ... => O(n)

Item ord = (Item) listIter.previous(); => O(1)

shipped.add(ord); => O(1)

listIter.remove(); => O(1)

O(n)

```java
public Item getShippedOrderfromLatest(int index){
    if(index >= 0 && index < shipped.size()){
        ListIterator listIter = shipped.listIterator(shipped.size());
        for(int i = 0; i < index; ++i)
            listIter.previous();
        Item ord = (Item) listIter.previous();
        return ord;
    }
    return null;
}
```

O(n) = O(n)

O(n)

```java
public String toString(){
    StringBuilder strBuild = new StringBuilder();

    strBuild.append("Last order(s):\n");
    ListIterator listIter_last = lastOrder.listIterator(lastOrder.size());
    for(int i = 0; listIter_last.hasPrevious(); ++i)
        strBuild.append(String.valueOf(i+1)).append(") ").append(listIter_last.previous()).append("\n");
    /*for(int i = 0; i < lastOrder.size(); ++i)
        strBuild.append(String.valueOf(i + 1) + ") ").append(getOrderfromLatest(i)).append("\n");*/
    strBuild.append("Shipped order(s):\n");
    ListIterator listIter_shipped = shipped.listIterator(shipped.size());
    for(int i = 0; listIter_shipped.hasPrevious(); ++i)
        strBuild.append(String.valueOf(i+1)).append(") ").append(listIter_shipped.previous()).append("\n");

    return strBuild.toString();
}
```

O(n) { => O(1) } => O(n)

Lastorder size

O(m) { => O(1) } => O(m)

Shipped order size

O(n+m)

Online Depot Class

```java
public void addDepot(Depot depot){
    if(depot == null){
        throw(new IllegalArgumentException());
    }
    depots.add(depot);
}
```

O(1)

k

```java
public void removeDepot(Depot depot){
    for(int i = 0; i < depots.size(); ++i){
        if(depots.get(i) == depot)
            depots.remove(i);
    }
}
```

O(n) ⟸ for

O(n) ⟸ depots.remove(i);

$O(n^2)$ because arraylist shifts it to left,

m => order size

```java
public boolean removeItemFromDepots(Order order){
    boolean found = false;

    for(int i = 0; i < depots.size() && found == false; ++i){ => O(n)
        for(int j = 0; j < order.getOrderCount(); ++j){ => O(m) => O(m)
            if(depots.get(i).isContain(order.getOrderfromLatest(j))){
                found = true;
            }
            else{
                found = false;
                break;
            }
        }
        if(found){
            for(int k = 0; k < order.getOrderCount(); ++k)
                depots.get(i).removeFromDepot(order.getOrderfromLatest(k)); => O(m)

            order.removeAllOrders(); => O(m)
            return true;
        }

    }
    return false;
}
```

$O(m^2)$

$O(m)$

$O(n \cdot m^2)$

```java
public String searchProduct(String name){
    StringBuilder strBuild = new StringBuilder();
    for(int i = 0; i < depots.size(); ++i){
        HybridList<Item> founded = depots.get(i).searchProduct(name);
        if(founded.size() > 0){
            strBuild.append(depots.get(i).getBranch().getName() + ":\n");
        ListIterator listIter = founded.listIterator();
        while(listIter.hasNext()){
            Item item = (Item) listIter.next();
            strBuild.append("   ").append(item).append("\n");
        }
    }
    return strBuild.toString();
}
```

=> $O(n)$

=> $O(m)$

=> $O(1)$

$O(k)$

=> $O(k)$

=> $O(1)$

=> $O(1)$

$O(m+k)$

$O(n(m+k))$

$T_b = O(n \cdot m)$

$T_w = O(n(m+k))$

$T_b = O(m)$

$T_w = O(m+k)$

If it didn't find

Info Box Class

```java
public void addRestockInfo(Branch branch, Item item){
    if(item == null || branch == null){
        throw(new IllegalArgumentException());
    }
    box.add(branch.getName() + item.toString());
}
```

=> $O(1)$

$O(1)$

Hybrid

$O(n)$

```java
public String getInfoFromLast(int ind) { return box.get(box.size() - 1 - ind); }
```

```java
public boolean removeInfo(int ind){
    if(ind >= 0 && ind < box.size()){
        box.remove( index: box.size() - 1 - ind);
        return true;
    }
    return false;
}
```

=> $O(n)$

$O(n)$

```java
public String toString(){
    StringBuilder strBuild = new StringBuilder();
    ListIterator<String> box_iter = box.listIterator(box.size());
    for(int i = 0; i < box.size(); ++i){
        strBuild.append("RESTOCK: ");
        strBuild.append(String.valueOf(i+1)).append(") ").append(box_iter.previous()).append('\n');
    }
    return strBuild.toString();
}
```

O(n) { ... } => O(1) } O(n)

=> O(n)

Branch Class

```java
public Employee findEmployee(String ID){
    for(int i = 0; i < employees.size(); ++i){
        if(employees.get(i).getID().equals(ID))
            return employees.get(i);
    }

    return null;
}
```

=> O(1)

O(n) } O(n)

↳ ArrayList

```java
public void addEmployee(Employee emp) { employees.add(emp); }
```

=> O(1)

Amortized

```java
/**
 * Removes employee by ID.
 * @param ID id of the employee will be removed.
 * @return returns true if operation is successful, else false.
 */
public boolean removeEmployee(String ID){
    for(int i = 0; i < employees.size(); ++i){
        if(employees.get(i).getID().equals(ID)){
            employees.remove(i);
            return true;
        }
    }
    return false;
}
```

O(n) → O(n)

=> O(n)

O(n) } O(n²) } O(n²)

$T_b = O(1)$

$T_w = O(n^2)$

ü

```java
public String toString(){
    StringBuilder strBuild = new StringBuilder();
    strBuild.append(branchDepot).append("\n").append("Employees:");
    for(int i = 0; i < employees.size(); ++i){
        strBuild.append("    ").append(employees.get(i).toString()).append("\n");
    }
    return strBuild.toString();
}
```

O(n) => O(n) => O(1) => O(n)

Branch Employee Class

```java
public Customer createNewCustomer(String name, String surname, String email, String password){
    Customer cust = new Customer(name, surname, email, password, branch.getOnlineDep());
    branch.getCustList().addCustomer(cust);
    return cust;
}
```

=> O(n) => O(n)

```java
public Order getOrder(String ID){
    Customer cust = branch.getCustList().getCustomerWithID(ID);
    if(cust != null){
        return cust.getOrder();
    }
    else
        return null;
}
```

O(n) <= O(n)

```java
public boolean addOrder(Order order, Item item) throws IllegalArgumentException{
    if(item == null || order == null){
        throw(new IllegalArgumentException());
    }
    if(item.getQuantity() > 0 && removeItem(item)){   //if it is succesfully remo
        order.addFinishedOrder(item);
        return true;
    }
    else
        return false;
}
```

O(n) => O(n) => O(1) => O(1) => O(1) => O(n)

```java
public boolean removeOrder(Order order, int index) throws IllegalArgumentException{
    if(order == null){
        throw(new IllegalArgumentException());
    }
    if(order.removeOrderfromLatest(index))=>O(n)
        return true; => O(1)
    else
        return false; =>O(1)
}
```

O(n)

```java
public boolean addItem(Item item){
    if(item == null){
        throw(new IllegalArgumentException());
    }
    Depot depot = getBranch().getDepot(); =>O(1)
    if(depot.addToDepot(item))=>O(1)
        return true;
    else
        return false;
}
```

O(1)

```java
public boolean removeItem(Item item){
    Depot depot = getBranch().getDepot(); =>O(1)

    if(depot.removeFromDepot(item)) => O(n)
        return true;
    else
        return false;
}
```

O(n)

```java
public void restockInform(Item item){
    if(item == null){
        throw(new IllegalArgumentException());
    }
O(1)=getBranch().getInfoBox().addRestockInfo(getBranch(), item);
}
```

O(1)

Customer Class

```java
public String searchProduct(String name) { return onlineDep.searchProduct(name); }
```
=> O(n(m+k))

```java
/**
 * Removes item from the order list.
 * @param ind index of the item which will be removed from Order List.
 * @return returns true if Order successfully removed.
 */
public boolean removeItemFromOrder(int ind) { return getOrder().removeOrderfromLatest(ind); }
```
=> O(n)

```java
/**
 * Adds item to the order list.
 * @param item Item information which will be added to the order list.
 * @return returns true if Order successfully added to the order list.
 */
public boolean addToOrder(Item item) { return(getOrder().addOrder(item)); }
```
=> O(1)

O(n m²)

```java
public boolean makeOnlineOrder(){
    if(getAddress() == null || getPhone() == null){
        //exception
    }
    return onlineDep.removeItemFromDepots(this.getOrder());
}
```
=> O(n . m²)

Company Class

```
public Customer createCustomerAccount(String name, String surname, String email, String password){
    Customer returnVal = new Customer(name, surname, email, password, onlineDepot);
    if(!customers.addCustomer(returnVal)){ => O(n)
        return null;
    }
    return returnVal;
}
```

O(n)

```
public boolean addBranch(String name){
    Branch branch = new Branch(name, customers, onlineDepot);
    if(!branches.contains(branch)){ => O(n)
        branches.add(branch); => O(1) => adding to last is constant
        onlineDepot.addDepot(branch.getDepot()); => O(1)
        return true;
    }
    else
        return false;
}
```

O(n)

```
public Branch getBranch(String name){
    ListIterator<Branch> listIter = branches.listIterator();
    while(listIter.hasNext()) { => O(n)
        Branch branch = (Branch) listIter.next(); => O(1)
        if (branch.getName().equals(name)) => O(1)
            return branch; => O(1)
    }
    return null;
}
```

O(n) O(n)

```
public boolean removeBranch(String name){
    ListIterator<Branch> listIter = branches.listIterator();
    while(listIter.hasNext()) { => O(n)
        Branch branch = (Branch) listIter.next();
        if (branch.getName().equals(name)) {
            onlineDepot.removeDepot(branch.getDepot()); => O(n^2)
            listIter.remove(); => O(1)
            return true;
        }
    }
    return false;
}
```

$O(n^3)$

$n = n$

depot = branch

Admin Class

```java
public boolean addBranch(String name) { return comp.addBranch(name); }
```
=> O(n)

```java
/**
 * Removes new branch from the company.
 * @param name unique name of the branch.
 * @return returns true if branch name is unique and removing is success
 */
public boolean removeBranch(String name){
    Branch branch = getBranch(name);
    if(branch != null)
        comp.getOnlineDepot().removeDepot(branch.getDepot());
    return comp.removeBranch(name);
}
```
=> O(n²)

O(n²)

```java
public Branch getBranch(String name) { return comp.getBranch(name); }
```
=> O(n)

```java
/**
 * Adds branch employee to a branch.
 * @param branch branch object of the branch that will add employee.
 * @param name name of the employee.
 * @param surname surname of the employee.
 * @return returns Employee object.
 * @throws IllegalArgumentException if branch or name or surname is null.
 */
public Employee addBranchEmployee(Branch branch, String name, String surname)
    if(branch == null || name == null || surname == null){
        throw(new IllegalArgumentException());
    }
    BranchEmployee emp = new BranchEmployee(name, surname, branch);
    branch.addEmployee(emp);
    comp.increaseEmployeeCount();
    return emp;
}
```
=> O(1)
=> O(1)
=> O(1)

O(1)

```java
public boolean removeBranchEmployee(Branch branch, String ID) throws IllegalArgumentException{
    if(branch == null || ID == null){
        throw(new IllegalArgumentException());
    }
    return(branch.removeEmployee(ID));
}
```

=> O(n²)

O(n²)

```java
/**
 * It gives texts in information box, for inform admin.
 * @param branch branch object of the branch.
 * @return returns String of texts in information box.
 * @throws IllegalArgumentException if branch is null.
 */
public String lookInfoBoxes(Branch branch) throws IllegalArgumentException{
    if(branch == null){
        throw(new IllegalArgumentException());
    }
    return branch.getInfoBox().toString();
}
```

=> O(n)

O(n)

```java
public boolean removeInfo(Branch branch, int ind)
    if(branch == null){
        throw(new IllegalArgumentException());
    }
    return branch.getInfoBox().removeInfo(ind);
}
```

=> O(n)

O(n)