Time Complexity Analysis of Company System Methods

Notes:

Llist Methods Time Complexities: KWArrayList Methods Time Complexities:

-size(): T(n)= Θ(1) -setcap(int newCapVal): T(n)= Θ(1)

-getFirst(): T(n)= Θ(1) -reallocate(): T(n)= O(n)

-getLast(): T(n)= Θ(1) -get(int index): T(n)= Θ(1)

-get(int index): T(n)= O(n) -size(): T(n)= Θ(1)

-add(int index, E obj): T(n)=O(n) -set(int index, E element): T(n)= Θ(1)

-addFirst(E obj): T(n)= Θ(1) -add(E element): T(n)= Θ(1)

-addLast(E obj): T(n)= Θ(1) -add(int index, E element): T(n)= O(n)

-remove(E obj): T(n)= O(n) -indexOf(E target): T(n)= O(n)

-remove(int index): T(n)= O(n) -remove(int index): T(n)= O(n)

-iterator(): T(n)= Θ(1)

-iterator(int i): T(n)= O(n)

HybridList Methods Time Complexities:

theList’s size=n , KWArrayLists’ sizes=m

-size(): T(n)= O(n)

-listSize(): T(n)= Θ(1)

-arraySize(): T(n)= O(n)

-get(int listIndex, int arrayIndex): T(n)= O(n)

-get(int index): T(n)= O(n)

-add(E element): T(n)= Θ(1)

-remove(): T(n)= Θ(1)

-remove(int index): T(n)= O(n)

-remove(int listIndex, int arrayIndex): T(n)= O(n)

User Methods Time Complexity:

-public String getName(){return name;} : T(n)= Θ(1)

-public String getSurname(){return surname;} : T(n)= Θ(1)

-public String getEmail(){return email;} : T(n)= Θ(1)

-public String getPassword(){return password;} : T(n)= Θ(1)

-public String getStatus() {return status;} : T(n)= Θ(1)

-protected void setStatus(String newStatus)throws InvalidStatus

{

if(!newStatus.equals("new") && !newStatus.equals("old"))

throw new InvalidStatus("The status of a user can be either new or old.\n");

status=new String(newStatus);

} : **T(n)=O(n)** because equals method for strings take O(n) time. It needs to compare each character in the strings.

-public boolean equals(User other)

{

if(getName().equals(other.getName()) && getSurname().equals(other.getSurname()) && getEmail().equals(other.getEmail()))

return true;

return false;

}: **T(n)=O(n)** because equals method for strings take O(n) time. It needs to compare each character in the strings.

Admin-BranchEmployee-Customer Methods Time Complexity

Note: I will be examining some of the methods that are shared only once because the only difference they have is the file names that their data is stored.

setEmail(String newMail), checkEmail(String mailInfo), save(Admin, Employee or Customer), toString() methods will be analyzed once for all 3 classes for this reason.

delete(Admin, Employee) and clone() for Admin and BranchEmployee will be analyzed once for this reason too.

-protected void setEmail(String newMail)

{

try

{

//Checks if the email was registered from AdminsInfo.txt file

File infoFile = new File("AdminsInfo.txt");

Scanner fileSC= new Scanner(infoFile);

Scanner inSC= new Scanner(System.in);

String mailInput=new String(newMail);

while(fileSC.hasNext())

{

String readWord=new String(fileSC.next());

if(readWord.equals(mailInput))

{

//If it was registered from before, asks for a new email

System.out.println("This email already exists. Enter another email.");

fileSC.close();

mailInput=new String(inSC.next());

fileSC= new Scanner(infoFile);

}

}

//Registers the proper email

email=new String(mailInput);

fileSC.close();

}

catch(FileNotFoundException exc)

{

System.out.println("The file does not exist.");

System.exit(-1);

}

}: **T(n,m)=O(m\*n) , if m\*n=K T(K)=O(K), assuming n is the biggest one the worst case would be T(n)=O(n^2)**  Because fileSC.hasNext() will make the loop run till it reaches end of the file. We can assume it will run m times for m lines. Inside the while loop, there is an equals() method for strings which will run O(n) times in each run of the loop.

-public void saveAdmin()

{

if(getStatus().equals("new"))

{

try

{

FileWriter fw = new FileWriter("AdminsInfo.txt", true);

fw.write("\n"+getEmail()+" "+getName()+" "+getSurname()+" "+getPassword());

fw.close();

}

catch(IOException ioe)

{

System.err.println("IOException: " + ioe.getMessage());

System.exit(-1);

}

}

}: **T(n)=O(n)** Because there is an equals() method for strings which has O(n) time complexity.

-protected void checkEmail(String mailInfo)throws InvalidStatus

{

try

{

File infoFile = new File("AdminsInfo.txt");

Scanner fileSC= new Scanner(infoFile);

while(fileSC.hasNext())

{

String readWord=new String(fileSC.next());

if(readWord.equals(mailInfo))

break;

readWord=new String(fileSC.nextLine());

}

if(!fileSC.hasNext())

{

fileSC.close();

throw new InvalidStatus("There is not an already existing admin with this email address.");

}

fileSC.close();

}

catch(FileNotFoundException exc)

{

System.out.println("The file does not exist.");

System.exit(-1);

}

}: **T(n,m)=O(m\*n) , if m\*n=K T(K)=O(K), assuming n is the biggest one the worst case would be T(n)=O(n^2)** Because fileSC.hasNext() will make the loop run till it reaches end of the file. We can assume it will run m times for m lines. In the loop there is an equals() method for strings which has O(n) time complexity. Equals() will run in each loop therefore it will make O(m\*n) time complexity.

-public void deleteAdmin()

{

try

{

File inputFile = new File("AdminsInfo.txt");

File outputFile = new File("temp.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

fw.write(sc.nextLine()+"\n");

while(sc.hasNextLine())

{

String str = sc.nextLine();

str = str.replaceAll(getEmail()+" "+getName()+" "+getSurname()+" "+getPassword(), "");

if(!str.equals(""))

fw.write("\n"+str);

}

sc.close();

fw.close();

sc=null;

System.gc();

inputFile.delete();

outputFile.renameTo(inputFile);

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

}: **T(n,m)=O(m\*n), if m\*n=K T(K)=O(K), assuming n is the biggest one the worst case would be T(n)=O(n^2)** Because sc.hasNextLine() will make the loop run till it reaches end of the file. We can assume it will run m times for m lines. Inside the while loop there is replaceAll method for a string. This method will search fort he parameter string inside str and replace it if it finds. This requires search in a string also modifying of an immutable string which have both O(n) time complexity. replaceAll will run in each loop therefore it will make O(m\*n) time complexity.

-public String toString()

{

return String.format(getName()+" "+getSurname());

}: **T(n)=Θ(1)**

public Object clone() throws CloneNotSupportedException

{

Admin temp=(Admin)super.clone();

temp.status=new String("old");

return temp;

}: **T(n)=Θ(1)**

-public int getCustomerNo(){return customerNumber;} : **T(n)=Θ(1)**

-public static int getTotalCustomerCount(){return totalCustomerCount;} : **T(n)=Θ(1)**

-public Product getNewOrder(int orderIndex)throws InvalidIndex

{

if(orderIndex<0 || orderIndex>newOrders.size())

throw new InvalidIndex("There is no order with this index.");

return newOrders.get(orderIndex);

}: **T(n)=O(n^2)** because get method of HybridList is being used.

-public int getNewOrderCount()

{

if(newOrders!=null)

return newOrders.size();

return 0;

}: **T(n)=Θ(1)**

-public String getPhoneNumber()

{

if(phoneNumber!=null)

return phoneNumber;

return String.format("NONE");

}: **T(n)=Θ(1)**

-public String getAddress()

{

if(address!=null)

return address;

return String.format("NONE");

}: **T(n)=Θ(1)**

-public void removeCustomer()

{

try

{

File inputFile = new File("CustomersInfo.txt");

File outputFile = new File("temp.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

String temp;

fw.write(sc.nextLine());

sc.next();

while(sc.hasNextLine())

{

temp=sc.next();

if(!temp.equals(getEmail()))

{

fw.write("\n-");

fw.write("\n"+temp);

temp=sc.nextLine();

fw.write(temp);

while(sc.hasNextLine() && !temp.equals("-"))

{

temp=sc.nextLine();

if(!temp.equals("-"))

fw.write("\n"+temp);

}

}

else

while(sc.hasNextLine() && !temp.equals("-"))

temp=sc.nextLine();

}

sc.close();

fw.close();

sc=null;

System.gc();

inputFile.delete();

outputFile.renameTo(inputFile);

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

}: **T(n,m)=O(m^2\*n), assuming n is the biggest one the worst case would be T(n)=O(n^3)** Because the inner while loop will be run till it encounters a “-“ in the file. We can assume it runs m times for m lines. In the inner loop there is a equals method for a string which takes O(n) for a string that has length n. This inner loop will be run till the end of the file in the worst case. For m lines it runs m times. This makes O(m)\*(O(n)\*O(m))

-private void loadOrders()

{

String colourInfo;

int modelInfo;

String Stemp;

try

{

File infoFile = new File("CustomersInfo.txt");

Scanner fileSC= new Scanner(infoFile);

do

{/\*Intentionally left empty\*/}

while(!fileSC.next().equals(getEmail()));

fileSC.nextLine();

fileSC.nextLine();

while(fileSC.hasNextLine())

{

Stemp=new String(fileSC.next());

if(Stemp.equals("NewOrders"))

break;

modelInfo=Integer.parseInt(fileSC.next());

colourInfo=new String(fileSC.next());

Product temp=new Product(Stemp, modelInfo, colourInfo);

Stemp=new String(fileSC.next());

temp.setStoreInfo(Integer.parseInt(Stemp));

storePastOrder(temp);

}

while(fileSC.hasNextLine())

{

Stemp=new String(fileSC.next());

if(Stemp.equals("-"))

break;

modelInfo=Integer.parseInt(fileSC.next());

colourInfo=new String(fileSC.next());

Product temp=new Product(Stemp, modelInfo, colourInfo);

Stemp=new String(fileSC.next());

temp.setStoreInfo(Integer.parseInt(Stemp));

storeNewOrder(temp);

}

fileSC.close();

}

catch(FileNotFoundException exc)

{

System.out.println("The file does not exist.");

System.exit(-1);

}

}: **T(n,m)=O(m\*n) , if m\*n=K T(K)=O(K), assuming n is the biggest one the worst case would be T(n)=O(n^2)** Because fileSC.hasNextLine() will make the loop run till it reaches the end of the file. We can assume it will run m times for m lines. There is equals() method for strings also storePastOrder(temp). Both have O(n) time complexity and they will run in each loop. Therefore O(n)\*O(m)

private void storePastOrder(Product addedProduct)

{

try

{

if(pastOrders==null)

pastOrders=new HybridList<Product>();

pastOrders.add((Product)addedProduct.clone());

}

catch(CloneNotSupportedException exc)

{

System.err.print(exc);

System.exit(-1);

}

}: **T(n)= Θ(1)** Because pastOrders.add() method has Θ(1) time complexity.

-private void storeNewOrder(Product addedProduct)

{

try

{

if(newOrders==null)

newOrders=new HybridList<Product>();

newOrders.add((Product)addedProduct.clone());

}

catch(CloneNotSupportedException exc)

{

System.err.print(exc);

System.exit(-1);

}

}: **T(n)= Θ(1)** Because newOrders.add() method hasΘ(1)time complexity.

-public void addNewOrder(Product addedProduct, int storeNo)

{

try

{

File inputFile = new File("CustomersInfo.txt");

File outputFile = new File("temp.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

String sTemp;

fw.write(sc.nextLine());

do

{

fw.write("\n"+sc.nextLine());

}

while(!sc.hasNext(getEmail()));

fw.write("\n"+sc.nextLine());

do

{

sTemp=new String(sc.nextLine());

fw.write("\n"+sTemp);

}

while(!sTemp.equals("NewOrders"));

fw.write("\n"+addedProduct.toString()+" "+storeNo);

while(sc.hasNextLine())

fw.write("\n"+sc.nextLine());

sc.close();

fw.close();

sc=null;

System.gc();

inputFile.delete();

outputFile.renameTo(inputFile);

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

storeNewOrder(addedProduct);

}: **T(n,m)=O(m\*n) , if m\*n=K T(K)=O(K), assuming n is the biggest one the worst case would be T(n)=O(n^2)** Because sc.hasNextLine(getEmail()) will run m times for m lines and this method compares strings which takes n times for a string with n length. O(n)\*O(m)

public void addPastOrder(Product addedProduct, int storeNo)

{

try

{

File inputFile = new File("CustomersInfo.txt");

File outputFile = new File("temp.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

fw.write(sc.nextLine());

do

{

fw.write("\n"+sc.nextLine());

}

while(!sc.hasNext(getEmail()));

fw.write("\n"+sc.nextLine());

fw.write("\n"+sc.nextLine());

fw.write("\n"+addedProduct.toString()+" "+storeNo);

while(sc.hasNextLine())

fw.write("\n"+sc.nextLine());

sc.close();

fw.close();

sc=null;

System.gc();

inputFile.delete();

outputFile.renameTo(inputFile);

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

storePastOrder(addedProduct);

}: **T(n,m)=O(m\*n) , if m\*n=K T(K)=O(K), assuming n is the biggest one the worst case would be T(n)=O(n^2)** Because sc.hasNextLine(getEmail()) will run m times for m lines and this method compares strings which takes n times for a string with n length. O(n)\*O(m)

- public void removeNewOrder(Product removedProduct)

{

try

{

int counter=0;

//Firstly removes from the newOrders array.

if(newOrders==null || newOrders.size()==0)

{

System.out.println("There is no registered order that can be removed.");

return;

}

else if(isNewOrder(removedProduct)==-1)

return;

else

{

for(int i=0; i<newOrders.size(); ++i)

if(newOrders.get(i).equals(removedProduct))

newOrders.remove(i);

}

//Then removes from the CustomerInfo.txt file

File inputFile = new File("CustomersInfo.txt");

File outputFile = new File("temp.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

String sTemp;

String typeInfo;

int modelInfo;

String colourInfo;

int storeInfo;

counter=0;

fw.write(sc.nextLine());

do

{

fw.write("\n"+sc.nextLine());

}

while(!sc.hasNext(getEmail()));

fw.write("\n"+sc.nextLine());

do

{

sTemp=new String(sc.nextLine());

fw.write("\n"+sTemp);

}

while(!sTemp.equals("NewOrders"));

while(sc.hasNextLine() && !sc.hasNext("-"))

{

typeInfo=new String(sc.next());

modelInfo=Integer.parseInt(sc.next());

colourInfo=new String(sc.next());

storeInfo=Integer.parseInt(sc.next());

if(typeInfo.equals(removedProduct.getType())==true && modelInfo==removedProduct.getModel() && colourInfo.equals(removedProduct.getColour())==true &&counter==0)

{

counter++;

}

else

{

fw.write("\n"+typeInfo+" ");

fw.write(" "+modelInfo);

fw.write(" "+colourInfo);

fw.write(" "+storeInfo);

}

}

if(sc.hasNextLine())

sc.nextLine();

while(sc.hasNextLine())

fw.write("\n"+sc.nextLine());

sc.close();

fw.close();

sc=null;

System.gc();

inputFile.delete();

outputFile.renameTo(inputFile);

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

}: **T(n)=O(n^2)** Because the the first loop used HybridList’s size which takes O(n) time also there is equals() and HybridList’s remove() method inside the loop which take O(n) time.

O(n)\*O(n)

-public void addInfo(String addressInfo, String phoneInfo)

{

address=new String(addressInfo);

phoneNumber=new String(phoneInfo);

}: **T(n)= Θ(1)**

-public void saveInfo(String addressInfo, String phoneInfo)

{

addInfo(addressInfo, phoneInfo);

try

{

File inputFile = new File("CustomersInfo.txt");

File outputFile = new File("temp.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

fw.write(sc.nextLine());

do

{

fw.write("\n"+sc.nextLine());

}

while(!sc.hasNext(getEmail()));

fw.write("\n"+sc.nextLine());

fw.write(" "+getPhoneNumber());

fw.write(" "+getAddress());

while(sc.hasNextLine())

fw.write("\n"+sc.nextLine());

sc.close();

fw.close();

sc=null;

System.gc();

inputFile.delete();

outputFile.renameTo(inputFile);

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

}: **T(n,m)=O(m\*n) , if m\*n=K T(K)=O(K), assuming n is the biggest one the worst case would be T(n)=O(n^2)** Because sc.hasNextLine(getEmail()) will run m times for m lines and this method compares strings which takes n times for a string with n length. O(n)\*O(m)

-public void printPastOrders()

{

if(pastOrders!=null)

for(int i=0; i<pastOrders.size(); ++i)

System.out.println(pastOrders.get(i));

else

System.out.println("No Past Orders");

}: **T(n)=O(n^2)** Because pastOrders.size() takes O(n) time and there is also there is pastOrders.get(i) which also takes O(n) time inside the loop. This loop will be repeated n times according to the pastOrder’s size. O(n)\*(O(n)+O(n))

-public void printNewOrders()

{

if(newOrders!=null)

for(int i=0; i<newOrders.size(); ++i)

System.out.println(newOrders.get(i));

else

System.out.println("No New Orders");

}: **T(n)=O(n^2)** Because newOrders.size() takes O(n) time and there is also there is newOrders.get(i) which also takes O(n) time inside the loop. This loop will be repeated n times according to the newOrder’s size. O(n)\*(O(n)+O(n))

-public Object clone() throws CloneNotSupportedException

{

Customer temp=(Customer)super.clone();

temp.status=new String("old");

if(pastOrders!=null)

{

temp.pastOrders=new HybridList<Product>();

for(int i=0; i<pastOrders.size(); ++i)

temp.pastOrders.add((Product)pastOrders.get(i).clone());

}

if(newOrders!=null)

{

temp.newOrders=new HybridList<Product>();

for(int i=0; i<newOrders.size(); ++i)

temp.newOrders.add((Product)newOrders.get(i).clone());

}

return temp;

}: **T(n)=O(n^2)** Because the loops will run n times depending on the pastOrder’s and newOrder’s sizes. Inside the loops there are get() methods which take O(n) time. O(n)\*O(n)

-public int isPastOrder(Product searchedProduct)

{

if(pastOrders!=null)

for(int i=0; i<pastOrders.size(); ++i)

if(pastOrders.get(i).equals(searchedProduct))

return i;

return -1;

}: **T(n)=O(n^2)** Because the loop will run n times depending on the pastOrder’s size. Inside the loop there is HybridList’s get method which take O(n) time. O(n)\*O(n)

-public int isNewOrder(Product searchedProduct)

{

if(newOrders!=null)

for(int i=0; i<newOrders.size(); ++i)

if(newOrders.get(i).equals(searchedProduct))

return i;

return -1;

}: **T(n)=O(n^2)** Because the loop will run n times depending on the newOrder’s size. Inside the loop there is HybridList’s get method which take O(n) time. O(n)\*O(n)

Product Methods Time Complexity

-public int getAvailableModels(){return availableModels;}: **T(n)=Θ(1)**

-public String getAvailableColour(int indexNo) throws IndexOutOfBoundsException

{

if(indexNo<0 || indexNo>colours.length)

throw new IndexOutOfBoundsException();

return colours[indexNo];

}: **T(n)=Θ(1)**

-public String getType() {return type;}: **T(n)=Θ(1)**

-public int getModel(){return model;} : **T(n)=Θ(1)**

-public String getColour(){return colour;} : **T(n)=Θ(1)**

-public int getAmount(){return amount;} : **T(n)=Θ(1)**

-public int getStoreInfo(){return storeInfo;} : **T(n)=Θ(1)**

-public void setStoreInfo(int storeNo)

{

int no=storeNo;

while(no<0 || no>Branch.getTotalBranchCount())

{

System.out.println("Invalid store no. Please enter again.");

Scanner sc=new Scanner(System.in);

no=sc.nextInt();

}

storeInfo=no;

}: **T(n)=Θ(1)**

-private void setType(String newType)throws InvalidType

{

if(Arrays.asList(types).contains(newType))

type=new String(newType);

else

throw new InvalidType("This store does not produce "+newType+".");

}: **T(n)=O(n)** Because contains methd will search the whole array in the worst case.

-private void setModel(int newModel) throws InvalidModel

{

model=newModel;

if(getModel()<1 || getModel()>getAvailableModels())

throw new InvalidModel("There is no Model "+getModel()+" for "+getType()+".");

}: **T(n)=Θ(1)**

-private void setColour(String newColour)throws InvalidColour

{

if(Arrays.asList(colours).contains(newColour))

colour=new String(newColour);

else

throw new InvalidColour(getType()+" cannot be "+newColour+".");

}: **T(n)=O(n)** Because contains method will search the whole array in the worst case.

-private void setInfo()

{

if(getType().equals("Office\_Chair"))

{

String colourlist[]= {"Black", "White", "Red", "Blue", "Green"};

colours=colourlist;

availableModels=7;

}

else if(getType().equals("Office\_Desk"))

{

String colourlist[]= {"Black", "White", "Red", "Blue"};

colours=colourlist;

availableModels=5;

}

else if(getType().equals("Meeting\_Table"))

{

String colourlist[]= {"Black", "White", "Red", "Blue"};

colours=colourlist;

availableModels=10;

}

else if(getType().equals("Bookcase"))

{

String colourlist[]= {"Black"};

colours=colourlist;

availableModels=12;

}

else if(getType().equals("Office\_Cabinet"))

{

String colourlist[]= {"Black"};

colours=colourlist;

availableModels=12;

}

}: **T(n)=O(n)** Because there are equals() methods which take n times for a string that has n length.

-private void setAmount(int newAmount)throws InvalidAmount

{

if(newAmount<0)

throw new InvalidAmount("There cannot be negative amount of products.");

amount=newAmount;

}: **T(n)=Θ(1)**

-public String toString()

{

return String.format(getType()+" "+getModel()+" "+getColour());

}: **T(n)=Θ(1)**

-public Object clone() throws CloneNotSupportedException

{

return super.clone();

}: **T(n)=Θ(1)**

-public boolean equals(Product other)

{

if(getType().equals(other.getType()) && getColour().equals(other.getColour()) && getModel()==other.getModel())

return true;

return false;

}: **T(n)=O(n)**

-public void changeAmount(int changeValue)throws InvalidAmount

{

setAmount(getAmount()+changeValue);

}: **T(n)=Θ(1)** Because there are equals() methods which take n times for a string that has n length.

OnlineStore and Branch Methods Time Complexity

Note: I will be examining the methods these classes share only once because the only difference they have is file names that they are saved. However I will examine changeNo method separately for both.

-public int getNo() {return no;} : **T(n)=Θ(1)**

-public Product getProduct(int index) throws InvalidIndex

{

if(stock==null || index<0 || index>stock.size()-1)

throw new InvalidIndex("There is not a product that is stored in this index.");

return stock.get(index);

}: **T(n)=O(n)** Because it uses HybridList’s get method which takes O(n) time.

-private void createStock()

{

try

{

FileWriter fw = new FileWriter("OnlineStock.txt");

fw.write("Product\t\t\tAmount");

Product temp=new Product("Office\_Chair", 1, "Black", 1);

for(int i=1; i<=7; ++i)

for(int j=0; j<5; ++j)

fw.write("\nOffice\_Chair "+i+" "+temp.getAvailableColour(j)+"\t5");

temp=new Product("Office\_Desk", 1, "Black", 1);

for(int i=1; i<=5; ++i)

for(int j=0; j<4; ++j)

fw.write("\nOffice\_Desk "+i+" "+temp.getAvailableColour(j)+"\t5");

temp=new Product("Meeting\_Table", 1, "Black", 1);

for(int i=1; i<=10; ++i)

for(int j=0; j<4; ++j)

fw.write("\nMeeting\_Table "+i+" "+temp.getAvailableColour(j)+"\t5");

temp=new Product("Bookcase", 1, "Black", 1);

for(int i=1; i<=12; ++i)

for(int j=0; j<1; ++j)

fw.write("\nBookcase "+i+" "+temp.getAvailableColour(j)+"\t5");

temp=new Product("Office\_Cabinet", 1, "Black", 1);

for(int i=1; i<=12; ++i)

for(int j=0; j<1; ++j)

fw.write("\nOffice\_Cabinet "+i+" "+temp.getAvailableColour(j)+"\t5");

fw.close();

fw=null;

System.gc();

}

catch(IOException ioe)

{

System.err.println("IOException: " + ioe.getMessage());

System.exit(-1);

}

}: **T(n)=Θ(1)** Because each loop has a determined number of runs. The biggest amount of a loop will run is 10\*4=40 which is still a constant.

-private void loadStock()

{

try

{

File filename = new File("OnlineStock.txt");

Scanner reader=new Scanner(filename);

stock=new HybridList<Product>();

String typeInfo=new String(reader.nextLine());

int modelInfo;

String colourInfo;

int amountInfo;

for(int i=0; i<119; ++i)

{

typeInfo=new String(reader.next());

modelInfo=Integer.parseInt(reader.next());

colourInfo=new String(reader.next());

amountInfo=Integer.parseInt(reader.next());

stock.add(new Product(typeInfo, modelInfo, colourInfo, amountInfo));

}

reader.close();

}

catch(FileNotFoundException exc)

{

System.out.println("This branch does not exist.");

System.exit(-1);

}

}: **T(n)=Θ(1)** Because the loop will run exactly 119 times and the add method of HybridList inside the loop takes Θ(1) time too.

- public void printStock()

{

for(int i=0; i<stock.size(); ++i)

System.out.println(stock.get(i)+"\t"+stock.get(i).getAmount());

}: **T(n)=O(n^2)** Because the loop will run n times depending on the stock’s size. Also the get method inside the loop has O(n) time complexity. O(n)\*O(n)

- public void changeStock(Product addedProduct, int amount)

{

int i;

try

{

//Firstly changes the array

for(i=0; i<stock.size(); ++i)

{

if(stock.get(i).equals(addedProduct))

{

stock.get(i).changeAmount(amount);

break;

}

}

//Then edits the online store's stock file.

File inputFile = new File("OnlineStock.txt");

File outputFile = new File("temp.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

fw.write(sc.nextLine()+"\n");

for(int j=0; j<i; ++j)

fw.write(sc.nextLine()+"\n");

for(int j=0; j<3; ++j)

{

if(j!=2)

fw.write(sc.next()+" ");

else

fw.write(sc.next()+"\t");

}

fw.write(stock.get(i).getAmount()+"");

sc.next();

if(sc.hasNextLine())

{

fw.write(sc.nextLine());

for(int j=0; j<stock.size()-i-1; ++j)

fw.write("\n"+sc.nextLine());

}

sc.close();

fw.close();

sc=null;

System.gc();

inputFile.delete();

outputFile.renameTo(inputFile);

}

catch(InvalidAmount exc)

{

System.err.print(exc);

System.exit(-1);

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

}: **T(n)=O(n^2)** Because the loop will run n times depending on the stock’s size. Also the get method inside the loop has O(n) time complexity. O(n)\*O(n)

changeNo method of OnlineStore

-public void changeNo(int newNo)

{

System.err.print("You cannot change Online Store's number. It is always 0.");

System.exit(-1);

}: **T(n)=Θ(1)**

-public int isInStock(Product searchedProduct)

{

for(int i=0; i<stock.size(); ++i)

if(stock.get(i).equals(searchedProduct) && stock.get(i).getAmount()>0)

return i;

return -1;

}: **T(n)=O(n^2)** Because the loop will run n times depending on the stock’s size. Also the get method inside the loop has O(n) time complexity. O(n)\*O(n)

-public Object clone() throws CloneNotSupportedException

{

OnlineStore temp=(OnlineStore)super.clone();

if(stock!=null)

{

temp.stock=new HybridList<Product>();

for(int i=0; i<stock.size(); ++i)

temp.stock.add((Product)stock.get(i).clone());

}

return temp;

}: **T(n)=O(n^2)** Because the loop will run n times depending on the stock’s size. Also the get method inside the loop has O(n) time complexity. O(n)\*O(n)

-public boolean equals(Store other)

{

if(getNo()==other.getNo())

return true;

return false;

}: **T(n)=Θ(1)**

changeNo method of Branch

- public void changeNo(int newNo)

{

try

{

File outputFile = new File("Branch"+newNo+"Stock.txt");

if(outputFile.exists())

{

System.err.print("There is already a branch with this number."+newNo);

System.exit(-1);

}

File inputFile = new File("Branch"+getNo()+"Stock.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

fw.write(sc.nextLine());

while(sc.hasNextLine())

fw.write("\n"+sc.nextLine());

sc.close();

fw.close();

inputFile.delete();

branchNumber=newNo;

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

}: **T(n)=O(n)** Because there is a loop that will run n times for n lines.

CompanySystem Methods Time Complexity

-public String getLoggedInStatus() {return loggedInStatus; }: **T(n)=Θ(1)**

-public int getStoreAmount() {return StoreList.size();} : **T(n)=Θ(1)**

-public int getSupplyAmount()

{

if(SupplyList==null)

return 0;

return SupplyList.size();

}: **T(n)=O(n)** Because uses HybridList’s size method.

-public Product getSupplyList(int index)throws InvalidIndex

{

if(index<0 || index>SupplyList.size()-1)

throw new InvalidIndex("There is no supply with this index.");

return SupplyList.get(index);

}: **T(n)=O(n)** Because uses HybridList’s get method.

-public Store getStore(int index)throws InvalidIndex

{

if(index<0 || index>StoreList.size()-1)

throw new InvalidIndex("There is no store with this index.");

return StoreList.get(index);

}: **T(n)=O(n)** Because uses LList’s get method.

-public int getCustomerAmount() {return CustomerList.size();}: **T(n)=Θ(1)**

-public Customer getCustomer(int index)throws InvalidIndex

{

if(index<0 || index>CustomerList.size()-1)

throw new InvalidIndex("There is no customer with this index.");

return CustomerList.get(index);

}: **T(n)=Θ(1)**

-public void enterSystem()

{

String mailInfo;

String nameInfo;

String surnameInfo;

String passInfo;

Scanner sc=new Scanner(System.in);

System.out.println("Enter Email:");

mailInfo=new String(sc.next());

if(isAdmin(mailInfo)!=-1)

loggedInStatus=new String("Admin");

else if(isEmployee(mailInfo)!=-1)

loggedInStatus=new String("Employee");

else if(isCustomer(mailInfo)!=-1)

loggedInStatus=new String("Customer");

else

{

System.out.println("You can register the system by entering a password, your name and surname.");

System.out.println("Enter your password:");

passInfo=new String(sc.next());

System.out.println("Enter your name:");

nameInfo=new String(sc.next());

System.out.println("Enter your surname:");

surnameInfo=new String(sc.next());

Customer newCostumer=new Customer(nameInfo, surnameInfo, mailInfo, passInfo, "new");

storeCustomer(newCostumer);

loggedInStatus=new String("Customer");

System.out.println("The registration is completed. Your customer number is: "+String.valueOf(newCostumer.getCustomerNo()));

}

System.out.println("Enter your password:");

passInfo=new String(sc.next());

while(loggedInStatus.equals("Admin") && !AdminList.get(isAdmin(mailInfo)).getPassword().equals(passInfo))

{

System.out.println("You have entered a wrong password. Try again:");

passInfo=new String(sc.next());

}

while(loggedInStatus.equals("Employee") && !EmployeeList.get(isEmployee(mailInfo)).getPassword().equals(passInfo))

{

System.out.println("You have entered a wrong password. Try again:");

passInfo=new String(sc.next());

}

while(loggedInStatus.equals("Customer") && !CustomerList.get(isCustomer(mailInfo)).getPassword().equals(passInfo))

{

System.out.println("You have entered a wrong password. Try again:");

passInfo=new String(sc.next());

}

loggedInEmail=new String(mailInfo);

}: **T(n)=O(n^2) (Amortized)** Because isAdmin(), isEmployee(), isCustomer() methods all take O(n^2) time. The while loops also have these methods inside. How many times these loops run is important. The loops will run till the user enters a proper input. Generally this loop should not run more than a few times which is constant time however if the user keeps entering wrong inputs, this will cause an infinite loop. I will be ignoring this situation because normally it should not happen. The time complexity is O(n^2) however it is amortized because of the ingored scenario.

-public void makeSale(int no)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the type of the product customer wants to buy:");

String typeInfo=sc.next();

System.out.println("Enter the model of the product customer wants to buy:");

int modelInfo=sc.nextInt();

System.out.println("Enter the colour of the product customer wants to buy:");

String colourInfo=sc.next();

System.out.println("Enter the branch number customer wants to buy from.");

int storeNo=sc.nextInt();

while(storeNo<1 || storeNo>=StoreList.size())

{

System.out.println("There is no branch with this number, please enter again.");

storeNo=sc.nextInt();

}

Product boughtProduct=new Product(typeInfo, modelInfo, colourInfo);

boughtProduct.setStoreInfo(storeNo);

for(int i=0; i<CustomerList.size(); ++i)

if(CustomerList.get(i).getCustomerNo()==no)

{

if(StoreList.get(storeNo).isInStock(boughtProduct)==-1)

{

CustomerList.get(i).addNewOrder(boughtProduct, storeNo);

addSupply(boughtProduct);

System.out.println("You have successfuly done your order from Branch "+storeNo+". Your order will be delivered as soon as it gets into stock.");

}

else

{

CustomerList.get(i).addPastOrder(boughtProduct, storeNo);

StoreList.get(storeNo).changeStock(boughtProduct, -1);

System.out.println("You have successfuly done your purchase from Branch "+storeNo+".");

}

}

}: **T(n,c)=O(n\*c^2) assuming n is the biggest one the worst case would be T(n)=O(n^3)** Because there is a loop which runs n times depending on how many customer there are. isInStock method takes O(n^2) time depending on how many supplies there are inside the stock. There are also methods like changeStock, get method of LList which have also O(n^2) time complexity inside the loop.

Inside the loop: “addNewOrder = O(n\*m) n=string lenght, m=line number” + “StoreList.get = O(k) k=store number” + “isInStock=O(c^2) c=stock size”, “changeStock=O(c^2) c=stock size” = O(n\*m)+O(k)+O(c^2)+O(c^2). I will represent this as O(c^2).

- public void shopMenu()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the type of the product you want to buy:");

String typeInfo=sc.next();

System.out.println("Enter the model of the product you want to buy:");

int modelInfo=sc.nextInt();

System.out.println("Enter the colour of the product you want to buy:");

String colourInfo=sc.next();

System.out.println("Enter the store number you want to buy from. For Online Store enter 0.");

int storeNo=sc.nextInt();

while(storeNo<0 || storeNo>StoreList.size()-1)

{

System.out.println("There is no store with this number, please enter again.");

storeNo=sc.nextInt();

}

int temp=-1;

Product boughtProduct=new Product(typeInfo, modelInfo, colourInfo);

while(StoreList.get(storeNo).isInStock(boughtProduct)==-1 && temp!=storeNo)

{

temp=storeNo;

System.out.println("\nCurrently this product is not in the stock of the store you want to buy from.");

System.out.println("You can make your order and wait for the stock to be updated or you can purchase it from other stores.");

System.out.println("This product can be found in these stores' stocks:");

searchProduct(boughtProduct);

System.out.println("\nPlease enter the store number you want to buy from again. You can also enter the same number to proceed with your order.");

storeNo=sc.nextInt();

if(temp==storeNo)

{

boughtProduct.setStoreInfo(storeNo);

for(int i=0; i<CustomerList.size(); ++i)

if(CustomerList.get(i).getEmail().equals(loggedInEmail))

CustomerList.get(i).addNewOrder(boughtProduct, storeNo);

System.out.println("You have successfuly done your order from Branch "+storeNo+". Your order will be delivered as soon as it gets into stock.");

}

}

boughtProduct.setStoreInfo(storeNo);

if(storeNo!=0 && temp!=storeNo)

{

for(int i=0; i<CustomerList.size(); ++i)

if(CustomerList.get(i).getEmail().equals(loggedInEmail))

CustomerList.get(i).addPastOrder(boughtProduct, storeNo);

StoreList.get(storeNo).changeStock(boughtProduct, -1);

System.out.println("You have successfuly done your purchase from Branch "+storeNo+".");

}

else if(temp!=storeNo)

{

for(int i=0; i<CustomerList.size(); ++i)

{

if(CustomerList.get(i).getEmail().equals(loggedInEmail) && CustomerList.get(i).getPhoneNumber()=="NONE")

{

System.out.println("You need to register your address and phone number to do shopping.");

String phoneInfo=sc.next();

String addressInfo=sc.next();

CustomerList.get(i).addInfo(addressInfo, phoneInfo);

}

}

for(int i=0; i<CustomerList.size(); ++i)

if(CustomerList.get(i).getEmail().equals(loggedInEmail))

CustomerList.get(i).addNewOrder(boughtProduct, storeNo);

StoreList.get(storeNo).changeStock(boughtProduct, -1);

System.out.println("You have successfuly done your purchase from the online store. Your purchase will be delievered as soon as possible.");

}

} **T(n,m,c)=O(n\*m\*c) assuming n is the biggest one the worst case would be T(n)=O(n^3)** Because in the while loop there is isInStock method which has O(n^2) time complexity when n is the product amount in the stock. The while loop will never run more than a few times therefore there being a while loop will not affect the time complexity. There is also another for loop inside the while loop. This loop will run n times depending on the customer amount. Also inside this loop there is addNewOrder method which had O(m\*c) time complexity when m is the lines in the file and c is the string length. Therefore O(n)\*O(m\*c)

-public void salesMenu()

{

try

{

for(int i=0; i<CustomerList.size(); ++i)

{

for(int j=0; j<CustomerList.get(i).getNewOrderCount(); ++j)

{

Product sale=(Product)CustomerList.get(i).getNewOrder(j).clone();

if(StoreList.get(sale.getStoreInfo()).isInStock(sale)==-1)

addSupply(sale);

else

{

CustomerList.get(i).removeNewOrder(sale);

CustomerList.get(i).addPastOrder(sale, sale.getStoreInfo());

StoreList.get(sale.getStoreInfo()).changeStock(sale, -1);

j--;

}

}

}

}

catch(InvalidIndex exc)

{

System.err.print(exc);

System.exit(-1);

}

catch(CloneNotSupportedException exc)

{

System.err.print(exc);

System.exit(-1);

}

} **T(n)=O(n^4)** Because there is a nested loop which will run n^2 times and inside the loops, there are methods which have O(n^2) time like removeNewOrder, isInStock.

-private void loadAdmins()

{

try

{

String mailInfo;

String nameInfo;

String surnameInfo;

String passInfo;

Admin atemp;

File inputFile = new File("AdminsInfo.txt");

Scanner sc=new Scanner(inputFile);

sc.nextLine();

sc.nextLine();

while(sc.hasNextLine())

{

mailInfo=new String(sc.next());

nameInfo=new String(sc.next());

surnameInfo=new String(sc.next());

passInfo=new String(sc.next());

atemp=new Admin(nameInfo, surnameInfo, mailInfo, passInfo, "old");

storeAdmin(atemp);

}

sc.close();

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

}: **T(n, m)=O(n^2\*m) assuming n is the biggest one the worst case would be T(n)=O(n^3)** Because there is a while loop which will run m times for m lines. Inside the while loop there is storeAdmin method which takes O(n^2) times.

-private void loadCustomers()

{

try

{

String mailInfo;

String nameInfo;

String surnameInfo;

String passInfo;

String phoneInfo;

String addressInfo;

String stemp;

Customer ctemp;

File inputFile = new File("CustomersInfo.txt");

Scanner sc=new Scanner(inputFile);

sc.nextLine();

sc.nextLine();

while(sc.hasNextLine())

{

mailInfo=new String(sc.next());

nameInfo=new String(sc.next());

surnameInfo=new String(sc.next());

passInfo=new String(sc.next());

ctemp=new Customer(nameInfo, surnameInfo, mailInfo, passInfo, "old");

sc.next();

stemp=new String(sc.next());

if(!stemp.equals("PastOrders"))

{

phoneInfo=new String(stemp);

addressInfo=new String(sc.next());

ctemp.addInfo(phoneInfo, addressInfo);

}

storeCustomer(ctemp);

do

{

if(sc.hasNextLine())

stemp=new String(sc.next());

}

while(!stemp.equals("-") && sc.hasNextLine());

if(sc.hasNextLine())

sc.nextLine();

}

sc.close();

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

}: **T(n, m)=O(n^2\*m) assuming n is the biggest one the worst case would be T(n)=O(n^3)** Because there is a while loop which will run m times for m lines. Inside the while loop there is storeCustomer method which takes O(n^2) times.

- private void loadEmployees()

{

try

{

String mailInfo;

String nameInfo;

String surnameInfo;

String passInfo;

BranchEmployee etemp;

File inputFile = new File("BranchEmployeesInfo.txt");

Scanner sc=new Scanner(inputFile);

sc.nextLine();

sc.nextLine();

while(sc.hasNextLine())

{

mailInfo=new String(sc.next());

nameInfo=new String(sc.next());

surnameInfo=new String(sc.next());

passInfo=new String(sc.next());

etemp=new BranchEmployee(nameInfo, surnameInfo, mailInfo, passInfo, "old");

storeEmployee(etemp);

}

sc.close();

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

}: **T(n, m)=O(n^2\*m) assuming n is the biggest one the worst case would be T(n)=O(n^3)** Because there is a while loop which will run m times for m lines. Inside the while loop there is storeEmployee method which takes O(n^2) times.

- private void loadSupplyList()

{

try

{

File inputFile = new File("SupplyList.txt");

Scanner sc=new Scanner(inputFile);

String typeInfo;

int modelInfo;

String colourInfo;

int storeInfo;

Product temp;

sc.nextLine();

while(sc.hasNextLine())

{

typeInfo=new String(sc.next());

modelInfo=Integer.parseInt(sc.next());

colourInfo=new String(sc.next());

temp=new Product(typeInfo, modelInfo, colourInfo);

storeInfo=Integer.parseInt(sc.next());

temp.setStoreInfo(storeInfo);

storeSupply(temp);

}

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

}:**T(n)=O(n)** Because there is a while loop that will run n times however everything inside it has Θ (1) time complexity.

-public void storeAdmin(Admin addedAdmin)

{

if(AdminList==null)

AdminList=new KWArrayList<Admin>();

else if(isAdmin(addedAdmin.getEmail())!=-1)

return;

AdminList.add(addedAdmin);

}: **T(n)=O(n^2)** Because there is isAdmin method which has O(n^2) time complexity. Add method is Θ (1).

-public int isAdmin(String searchedAdmin)

{

if(AdminList!=null)

for(int i=0; i<AdminList.size(); ++i)

if(AdminList.get(i).getEmail().equals(searchedAdmin))

return i;

return -1;

}: **T(n)=O(n^2)** Because there is a loop that will run n times depending on the admin amount. Inside this loop there is equals method that also has O(n) time complexity. Get method is Θ (1). O(n)\*O(n)

-public void printAdmins()

{

if(AdminList!=null)

for(int i=0; i<AdminList.size(); ++i)

System.out.println(AdminList.get(i));

}: **T(n)=O(n)** Because there is a loop that will run n times depending on admin amount. Inside the loop there is KWArrayList get method which is constant time.

-public void storeCustomer(Customer addedCustomer)

{

if(CustomerList==null)

CustomerList=new KWArrayList<Customer>();

else if(isCustomer(addedCustomer.getEmail())!=-1)

return;

CustomerList.add(addedCustomer);

}: **T(n)=O(n^2)** Because there is isCustomer method which has O(n^2) time complexity. Add method is Θ (1).

-public int isCustomer(String addedCustomer)

{

if(CustomerList!=null)

for(int i=0; i<CustomerList.size(); ++i)

if(CustomerList.get(i).getEmail().equals(addedCustomer))

return i;

return -1;

}: **T(n)=O(n^2)** Because there is a loop that will run n times depending on the customer amount. Inside this loop there is equals method that also has O(n) time complexity. Get method is Θ (1). O(n)\*O(n)

-public void printCustomers()

{

if(CustomerList!=null)

for(int i=0; i<CustomerList.size(); ++i)

System.out.println(CustomerList.get(i));

}: **T(n)=O(n)** Because there is a loop that will run n times depending on customer amount. Inside the loop there is KWArrayList get method which is constant time.

-public void storeEmployee(BranchEmployee addedEmployee)

{

if(EmployeeList==null)

EmployeeList=new KWArrayList<BranchEmployee>();

else if(isEmployee(addedEmployee.getEmail())!=-1)

return;

EmployeeList.add(addedEmployee);

}: **T(n)=O(n^2)** Because there is isEmployee method which has O(n^2) time complexity. Add method is Θ (1).

-public void removeEmployee(BranchEmployee removedEmployee)

{

try

{

if(EmployeeList==null || EmployeeList.size()==0)

{

System.out.println("There is no registered employee that can be removed.");

return;

}

else if(isEmployee(removedEmployee.getEmail())==-1)

return;

else

{

removedEmployee.deleteEmployee();

for(int i=0; i<EmployeeList.size(); ++i)

if(EmployeeList.get(i).getEmail().equals(removedEmployee.getEmail()))

{

EmployeeList.remove(i);

break;

}

}

}

catch(InvalidIndex exc)

{

System.err.print(exc);

System.exit(-1);

}

}: **T(n)=O(n^2)** Because there is isEmployee, deleteEmployee methods which are both O(n^2). There is also a loop which runs n times and has another method which has O(n) time complexity inside. O(n^2)+O(n^2)+O(n^2)

-public int isEmployee(String searchedEmployee)

{

if(EmployeeList!=null)

for(int i=0; i<EmployeeList.size(); ++i)

if(EmployeeList.get(i).getEmail().equals(searchedEmployee))

return i;

return -1;

}: **T(n)=O(n^2)** Because there is a loop that will run n times depending on the employee amount. Inside this loop there is equals method that also has O(n) time complexity. Get method is Θ (1). O(n)\*O(n)

-public void printEmployees()

{

if(EmployeeList!=null)

for(int i=0; i<EmployeeList.size(); ++i)

System.out.println(EmployeeList.get(i));

}: **T(n)=O(n)** Because there is a loop that will run n times depending on employee amount. Inside the loop there is KWArrayList get method which is constant time.

-public void storeSupply(Product addedSupply)

{

if(SupplyList==null)

SupplyList=new HybridList<Product>();

SupplyList.add(addedSupply);

}: **T(n)= Θ (1)** Because the add method of HybridList has constant time complexity.

-public void addSupply(Product addedSupply)

{

try

{

FileWriter fw = new FileWriter("SupplyList.txt", true);

fw.write("\n"+addedSupply.getType()+" "+addedSupply.getModel()+" "+addedSupply.getColour()+" "+addedSupply.getStoreInfo());

fw.close();

storeSupply(addedSupply);

System.out.println("Product has been added to the Supply List to inform the manager: "+addedSupply.getType()+" "+addedSupply.getModel()+" "+addedSupply.getColour());

}

catch(IOException ioe)

{

System.err.println("IOException: " + ioe.getMessage());

System.exit(-1);

}

} **T(n)= Θ (1)**

-public void removeSupply(Product removedSupply)

{

if(SupplyList==null || SupplyList.size()==0)

{

System.out.println("There is no registered supply that can be removed.");

return;

}

else if(isSupply(removedSupply)==-1)

{

System.out.println("This product is not in the supply list.");

return;

}

else

{

String typeInfo;

int modelInfo;

int storeInfo;

String colourInfo;

for(int i=0; i<SupplyList.size(); ++i)

if(SupplyList.get(i).equals(removedSupply))

SupplyList.remove(i);

try

{

int counter=0;

File inputFile = new File("SupplyList.txt");

File outputFile = new File("temp.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

while(sc.hasNextLine())

{

typeInfo=new String(sc.next());

modelInfo=Integer.parseInt(sc.next());

colourInfo=new String(sc.next());

storeInfo=Integer.parseInt(sc.next());

if(typeInfo.equals(removedSupply.getType())==true && modelInfo==removedSupply.getModel() && colourInfo.equals(removedSupply.getColour())==true && counter==0)

{

counter++;

}

else

{

fw.write("\n"+typeInfo);

fw.write(" "+modelInfo);

fw.write(" "+colourInfo);

fw.write(" "+storeInfo);

}

}

sc.close();

fw.close();

sc=null;

System.gc();

inputFile.delete();

outputFile.renameTo(inputFile);

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

}

}}: **T(n)=O(n^2)** Because there is a loop that will run n times depending on the supply amount and there is removemethod of HybridList inside the loop. It has O(n) time complexity. There is also methods like isSupply which had O(n^2) time complexity. O(n)\*O(n)+O(n^2)

-public int isSupply(Product searchedProduct)

{

if(SupplyList!=null)

for(int i=0; i<SupplyList.size(); ++i)

if(SupplyList.get(i).equals(searchedProduct))

return i;

return -1;

}: **T(n)=O(n^2)** Because there is a loop that will run n times depending on the supply amount and there is get method of HybridList inside the loop. It has O(n) time complexity. O(n)\*O(n)

-public void printSupplyList()

{

if(SupplyList!=null)

for(int i=0; i<SupplyList.size(); ++i)

System.out.println(SupplyList.get(i)+" to Store "+SupplyList.get(i).getStoreInfo());

}: **T(n)=O(n^2)** Because there is a loop that will run n times depending on the supply amount and there is get method of HybridList inside the loop. It has O(n) time complexity. O(n)\*O(n)

-public void addBranch(Store addedBranch)

{

try

{

StoreList.addLast(addedBranch);

File inputFile = new File("BranchCount.txt");

File outputFile = new File("temp.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

fw.write(String.valueOf(StoreList.size()-1));

sc.close();

fw.close();

sc=null;

System.gc();

inputFile.delete();

outputFile.renameTo(inputFile);

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

}: **T(n)= Θ (1)** Because addLast of LList is constant time.

- public void removeBranch(int branchNo)throws InvalidIndex

{

try

{

if(branchNo<1 || branchNo>StoreList.size()-1)

throw new InvalidIndex("There is no branch with this number");

else

{

File removedStock = new File("Branch"+branchNo+"Stock.txt");

removedStock.delete();

StoreList.remove(branchNo);

for(int i=0; i<StoreList.size(); ++i)

if(StoreList.get(i).getNo()>branchNo)

StoreList.get(i).changeNo(StoreList.get(i).getNo()-1);

Branch.totalBranchCount--;

File inputFile = new File("BranchCount.txt");

File outputFile = new File("temp.txt");

Scanner sc = new Scanner(inputFile);

FileWriter fw = new FileWriter(outputFile);

fw.write(String.valueOf(StoreList.size()-1));

sc.close();

fw.close();

sc=null;

System.gc();

inputFile.delete();

outputFile.renameTo(inputFile);

File inputFile2 = new File("SupplyList.txt");

if(inputFile2.exists())

{

SupplyList=null;

inputFile2.delete();

}

File inputFile3 = new File("CustomersInfo.txt");

File outputFile3 = new File("temp3.txt");

Scanner sc3 = new Scanner(inputFile3);

FileWriter fw3 = new FileWriter(outputFile3);

fw3.write(sc3.nextLine());

String stemp;

while(sc3.hasNext())

{

do

{

stemp=sc3.nextLine();

fw3.write("\n"+stemp);

}

while(!stemp.equals("PastOrders"));

while(!sc3.hasNext("NewOrders"))

{

String typeInfo=new String(sc3.next());

int modelInfo=Integer.parseInt(sc3.next());

String colourInfo=new String(sc3.next());

int storeInfo=Integer.parseInt(sc3.next());

fw3.write("\n"+typeInfo);

fw3.write(" "+modelInfo);

fw3.write(" "+colourInfo);

if(storeInfo==StoreList.size())

fw3.write(" 0");

else

fw3.write(" "+storeInfo);

}

fw3.write("\n"+sc3.next());

while(sc3.hasNext() && !sc3.hasNext("-"))

{

String typeInfo=new String(sc3.next());

int modelInfo=Integer.parseInt(sc3.next());

String colourInfo=new String(sc3.next());

int storeInfo=Integer.parseInt(sc3.next());

fw3.write("\n"+typeInfo);

fw3.write(" "+modelInfo);

fw3.write(" "+colourInfo);

if(storeInfo==StoreList.size())

fw3.write(" 0");

else

fw3.write(" "+storeInfo);

}

if(sc3.hasNextLine())

sc3.nextLine();

}

sc3.close();

fw3.close();

sc3=null;

System.gc();

inputFile3.delete();

outputFile3.renameTo(inputFile3);

}

}

catch (FileNotFoundException e2)

{

e2.printStackTrace();

}

catch (IOException e1)

{

e1.printStackTrace();

}

}: **T(n)=O(n^2)** Because there are loops that will run n times depending on how long the file is or when they encounter a specific string in the file. In these loops there are methods like equals(), hasNext(string) methods which have O(n) time complexity. O(n)\*O(n)

-public void querySupplyList()

{

if(SupplyList==null || SupplyList.size()==0)

{

System.out.println("There are no products that need to be supplied.");

}

else

{

System.out.println("These products should be supplied:");

printSupplyList();

}

}: **T(n)= Θ (1)** Because printSupplyList takes constant time.

-public void printAllProducts()

{

System.out.println("All Products:");

Product temp=new Product("Office\_Chair", 1, "Black", 1);

for(int i=1; i<=7; ++i)

for(int j=0; j<5; ++j)

System.out.println("Office\_Chair "+i+" "+temp.getAvailableColour(j));

temp=new Product("Office\_Desk", 1, "Black", 1);

for(int i=1; i<=5; ++i)

for(int j=0; j<4; ++j)

System.out.println("Office\_Desk "+i+" "+temp.getAvailableColour(j));

temp=new Product("Meeting\_Table", 1, "Black", 1);

for(int i=1; i<=10; ++i)

for(int j=0; j<4; ++j)

System.out.println("Meeting\_Table "+i+" "+temp.getAvailableColour(j));

temp=new Product("Bookcase", 1, "Black", 1);

for(int i=1; i<=12; ++i)

for(int j=0; j<1; ++j)

System.out.println("Bookcase "+i+" "+temp.getAvailableColour(j));

temp=new Product("Office\_Cabinet", 1, "Black", 1);

for(int i=1; i<=12; ++i)

for(int j=0; j<1; ++j)

System.out.println("Office\_Cabinet "+i+" "+temp.getAvailableColour(j));

}: **T(n)= Θ (1)** Because there are loops that will run for specified times. The biggest time a loop is run for is 10\*4=40 which is a constant.

-public void searchProduct(Product searchedProduct)

{

try

{

for(int i=0; i<StoreList.size(); ++i)

{

if(StoreList.get(i).isInStock(searchedProduct)!=-1 && i!=0)

{

System.out.println(searchedProduct+" can be found in Branch "+StoreList.get(i).getNo()+".");

System.out.println("There are "+(StoreList.get(i).getProduct(StoreList.get(i).isInStock(searchedProduct))).getAmount());

}

else if(StoreList.get(i).isInStock(searchedProduct)!=-1)

{

System.out.println(searchedProduct+" can be found in the Online Store.");

System.out.println("There are "+(StoreList.get(i).getProduct(StoreList.get(i).isInStock(searchedProduct))).getAmount());

}

}

}

catch(InvalidIndex exc)

{

System.err.print(exc);

System.exit(-1);

}

}: **T(n)= O(n^3)** Because there is a loop that will run n times depending on store amount. In this loop there is isInStock method which has O(n^2) time complexity. O(n^2)\*O(n)

-public void seePastOrders()

{

for(int i=0; i<CustomerList.size(); ++i)

{

if(CustomerList.get(i).getEmail().equals(loggedInEmail))

{

System.out.println("Your Past Orders:");

CustomerList.get(i).printPastOrders();

break;

}

}

}: **T(n)= O(n^3)** Because there is a loop that will run n times depending on how many customers there are and inside the loop there is printPastOrders.This method had O(n^2) time complexity. O(n)\*O(n^2)

-public void seePastOrders(int customerNo)

{

for(int i=0; i<CustomerList.size(); ++i)

{

if(CustomerList.get(i).getCustomerNo()==customerNo)

{

System.out.println("The Customer's Past Orders:");

CustomerList.get(i).printPastOrders();

break;

}

else if(i==CustomerList.size()-1)

System.out.println("There is no customer with this customer number.");

}

}: **T(n)= O(n^3)** Because there is a loop that will run n times depending on how many customers there are and when it finds the searched customer printPastOrders method will be run. This method had O(n^2) time complexity. O(n)\*O(n^2)