1. System Requirements

Firstly, most important requirements are Street object, ArrayList object, LinkedList object and LDLinkedList object. We need Street object to keep street properties and fields. We keep buildings in ArrayList, LinkedList and LDLinkedList object respectively. We'll put buildings to the Street and design them.

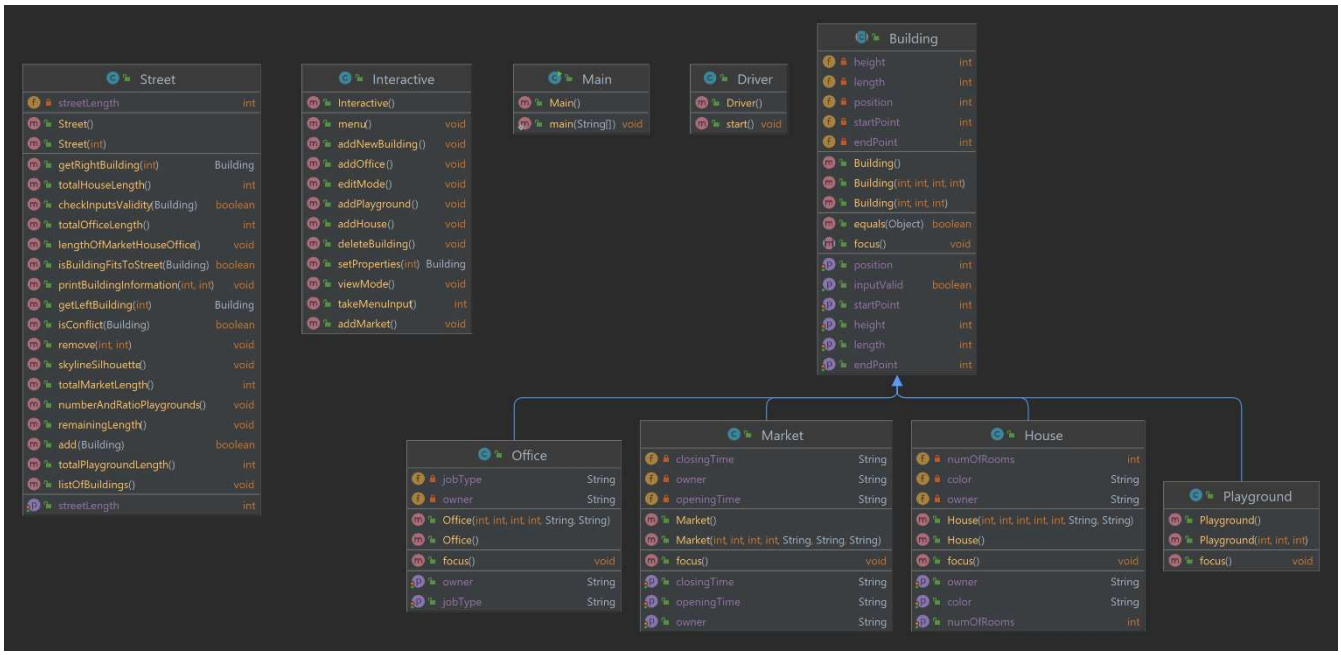
Then we need building object and its subclasses (House, Market, Office, Playground). Because Street, ArrayList, LinkedList and LDLinkedList are meaningless without buildings. We'll put Houses, Markets, Offices, Playgrounds to the ArrayList, LinkedList and LDLinkedList to provide city planning.

We also need Operating system that have jdk 8 and jre 8 for use the program.

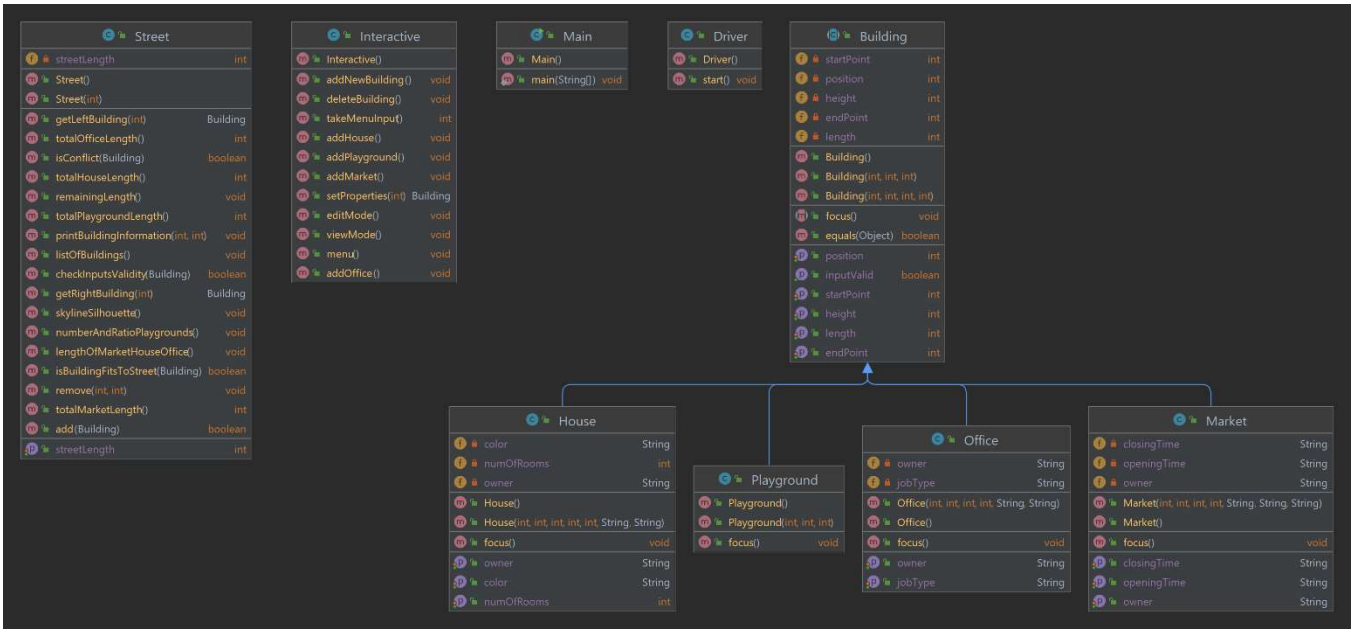
We need some space to store objects of Street, Building, House, Market etc. Each data structure requires different amount of memory. They may hold huge amount of memory according to number of the buildings and streets.

2. Class Diagrams

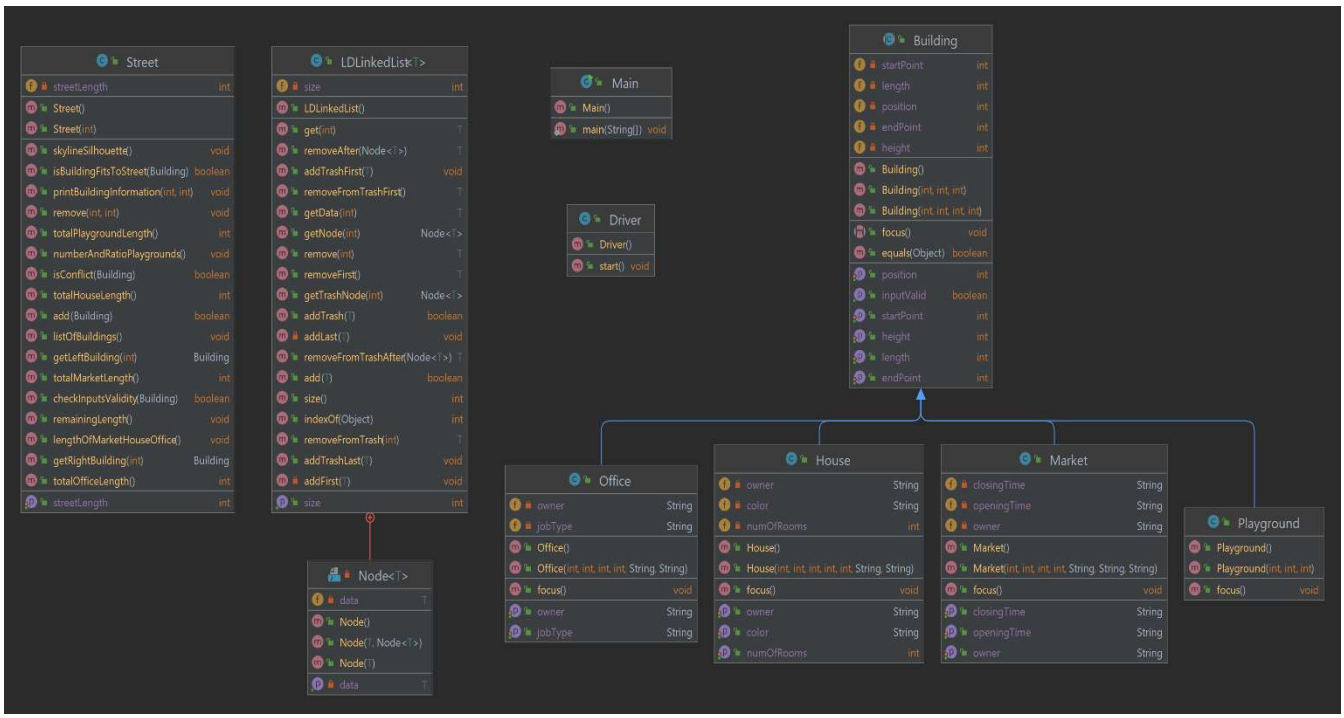
Part-1:



Part-2:


Street
streetLength int
Street()
Street(int)
getLeftBuilding(int) Building
totalOfficeLength() int
isConflict(Building) boolean
totalHouseLength() int
remainingLength() void
totalPlaygroundLength() int
printBuildingInformation(int, int) void
listOfBuildings() void
checkInputsValidity(Building) boolean
getRightBuilding(int) Building
skylineSilhouette() void
numberAndRatioPlaygrounds() void
lengthOfMarketHouseOffice() void
isBuildingFitsToStreet(Building) boolean
remove(int, int) void
totalMarketLength() int
add(Building) boolean
streetLength int
Interactive
Interactive()
addNewBuilding() void
deleteBuilding() void
takeMenuInput() int
addHouse() void
addPlayground() void
addMarket() void
setProperties(int) Building
editMode() void
viewMode() void
menu() void
addOffice() void
Main
Main()
main(String[]) void
Driver
Driver()
start() void
Building
startPoint int
position int
height int
endPoint int
length int
Building()
Building(int, int, int)
Building(int, int, int, int)
focus() void
equals(Object) boolean
position int
inputValid boolean
startPoint int
height int
length int
endPoint int
House
color String
numOfRooms int
owner String
House()
House(int, int, int, int, int, String, String)
focus() void
owner String
color String
numOfRooms int
Playground
Playground()
Playground(int, int, int)
focus() void
Office
owner String
jobType String
Office()
Office(int, int, int, int, String, String)
focus() void
owner String
jobType String
Market
closingTime String
openingTime String
owner String
Market()
Market(int, int, int, int, String, String, String)
focus() void
closingTime String
openingTime String
owner String

Part-3:


Street
streetLength int
Street()
Street(int)
skylineSilhouette() void
isBuildingFitsToStreet(Building) boolean
printBuildingInformation(int, int) void
remove(int, int) void
totalPlaygroundLength() int
numberAndRatioPlaygrounds() void
isConflict(Building) boolean
totalHouseLength() int
add(Building) boolean
listOfBuildings() void
getLeftBuilding(int) Building
totalMarketLength() int
checkInputsValidity(Building) boolean
remainingLength() void
lengthOfMarketHouseOffice() void
getRightBuilding(int) Building
totalOfficeLength() int
streetLength int
LDLinkedList<T>
size int
LDLinkedList()
get(int) T
removeAfter(Node<T>) T
addTrashFirst(T) void
removeFromTrashFirst() T
getData(int) T
getNode(int) Node<T>
remove(int) T
removeFirst() T
getTrashNode(int) Node<T>
addTrash(T) boolean
addLast(T) void
removeFromTrashAfter(Node<T>) T
add(T) boolean
size() int
indexOf(Object) int
removeFromTrash(int) T
addTrashLast(T) void
addFirst(T) void
size int
Node<T>
data T
Node()
Node(T, Node<T>)
Node(T)
data T
Main
Main()
main(String[]) void
Driver
Driver()
start() void
Building
startPoint int
length int
position int
endPoint int
height int
Building()
Building(int, int, int)
Building(int, int, int, int)
focus() void
equals(Object) boolean
position int
inputValid boolean
startPoint int
height int
length int
endPoint int
Office
owner String
jobType String
Office()
Office(int, int, int, int, String, String)
focus() void
owner String
jobType String
House
owner String
color String
numOfRooms int
House()
House(int, int, int, int, int, String, String)
focus() void
owner String
color String
numOfRooms int
Market
closingTime String
openingTime String
owner String
Market()
Market(int, int, int, int, String, String, String)
focus() void
closingTime String
openingTime String
owner String
Playground
Playground()
Playground(int, int, int)
focus() void

3. Problem Solutions Approach

Our problem is make an city planning software that will be used for designing a small one Street town. We have Street and we'll fill this Street with buildings. So we also need building class. In Street class, there should be ArrayList/LinkedList/LDLinkedList to keep all type of buildings.

There is Building base class and House, Market, Office, Playground subclasses. Each subclass have common and different properties. I wrote common properties to Building base class, like length, height, start point of building etc. I store all of the Building types in one Building array using polymorphism. I assign subclasses references into building array and they make polymorphic call.

To design a city, I need to be able to add/remove building. I do it in Street class because I'll add this objects into Building ArrayList/LinkedList/LDLinkedList. They have to placed into street properly.

I both create user menu and driver program. In driver program, every possibilities are evaluated. In user menu, I take an input, I evaluate it, if it is valid, action; else take an input again. When I take input from user, i check validity of input. So i prevent misadd objects to the Street. I also prevent misadd objects to the Street in driver program, but with different way. In driver program, I can't ask for a new input from user. So, I don't add wrong building objects to the street.

4. Test Cases

```
Street street1 = new Street( streetLength: 30);

//I create the buildings that are on the left side of the street
House house1 = new House( startPoint: 5, length: 10, height: 8, position: 1, numOfRooms: 5, color: "blue", owner: "owner1");
Playground playground1 = new Playground( startPoint: 15, length: 4, position: 1);
Office office1 = new Office( startPoint: 19, length: 7, height: 12, position: 1, jobType: "Job1", owner: "owner2");

//I create the buildings that are on the right side of the street
Market market1 = new Market( startPoint: 3, length: 15, height: 8, position: 2, openingTime: "08:30", closingTime: "21:00", owner: "owner3");
Office office2 = new Office( startPoint: 18, length: 8, height: 15, position: 2, jobType: "a101", owner: "owner4");
Office office3 = new Office( startPoint: 20, length: 4, height: 5, position: 2, jobType: "bim", owner: "owner5");

street1.add(house1);        //Adding house to left side of the street.
street1.add(playground1);   //Adding playground to left side of the street.
street1.add(office1);       //Adding office to left side of the street.

street1.add(market1);       //Adding market to right side of the street
street1.add(office2);       //Adding office to right side of the street
street1.add(office3);       //Ikinci office (office3) street'e eklenmiyor, cunku koordinatlari diğer office
                            //objesi ile kesisiyor.
street1.listOfBuildings();  //List of Buildings'de de görüldüğü gibi caddenin sağında sadece tek bir office var
street1.remainingLength();
street1.numberAndRatioPlaygrounds();
```

```
street1.listOfBuildings();  //List of Buildings'de de görüldüğü gibi caddenin sağında sadece tek bir office var
street1.remainingLength();
street1.numberAndRatioPlaygrounds();
street1.lengthOfMarketHouseOffice();
```

```
Building's one or more input is invalid, please add valid building

    Left Side Buildings                Right Side Buildings
----------------------------       ----------------------------
        House                                Market

        Playground                               Office

        Office

Total Remanining Length is: 16

Number of Playgrounds in the Street: 1
Ratio of Playgrounds in the Street: 0,07

Total Length of Street Occupied by the Houses: 10
Total Length of Street Occupied by the Offices: 15
Total Length of Street Occupied by the Markets: 15
```

It won't add 2nd office object to the street, because its coordinates conflict with other building's coordinate.

```
//I create the buildings that are on the left side
House house2 = new House( startPoint: 0, length: 6, height: 10, position: 1, numOfRooms: 4, color: "red", owner: "owner1");
House house3 = new House( startPoint: 7, length: 6, height: 7, position: 1, numOfRooms: 5, color: "purple", owner: "owner2");
Playground playground2 = new Playground( startPoint: 13, length: 4, position: 1);
Market market2 = new Market( startPoint: 17, length: 15, height: 15, position: 1, openingTime: "10:00", closingTime: "23:00", owner: "owner3");
Playground playground3 = new Playground( startPoint: 32, length: 6, position: 1);
House house4 = new House( startPoint: 38, length: 12, height: 20, position: 1, numOfRooms: 7, color: "brown", owner: "owner4");

//I create the buildings that are on the right side of the street
Office office4 = new Office( startPoint: 0, length: 20, height: 10, position: 2, jobType: "Mall", owner: "owner5");
Office office5 = new Office( startPoint: 25, length: 7, height: 20, position: 2, jobType: "Business Center", owner: "owner6");
Playground playground4 = new Playground( startPoint: 32, length: 10, position: 2);
Market market3 = new Market( startPoint: 42, length: 8, height: 10, position: 2, openingTime: "07:00", closingTime: "22:00", owner: "owner7");

street2.add(house2);            //Adding house to left side of the street.
street2.add(house3);            //Adding house to left side of the street.
street2.add(playground2);       //Adding playground to left side of the street.
street2.add(market2);           //Adding market to left side of the street
street2.add(playground3);       //Adding playground to left side of the street.
street2.add(house4);            //Adding house to left side of the street.
```

```
street2.add(office4);           //Adding office to right side of the street.
street2.add(office5);           //Adding office to right side of the street.
street2.add(playground4);       //Adding playground to right side of the street.
street2.add(market3);           //Adding market to right side of the street.

street2.listOfBuildings();        //Listing all buildings
street2.remainingLength();
street2.numberAndRatioPlaygrounds();
street2.lengthOfMarketHouseOffice();
```

```
    Left Side Buildings                 Right Side Buildings
----------------------------        ----------------------------
        House                               Office

        House                               Office

        Playground                              Playground

        Market                              Market

        Playground

        House

Total Remanining Length is: 6

Number of Playgrounds in the Street: 3
Ratio of Playgrounds in the Street: 0,20

Total Length of Street Occupied by the Houses: 24
Total Length of Street Occupied by the Offices: 27
Total Length of Street Occupied by the Markets: 23
```

```
                                //I remove playgrounds from street.
street2.remove( pos: 1, index: 2);    //Normally, second playground was at 4th index,
street2.remove( pos: 1, index: 3);    //but when I delete first playground, it
street2.remove( pos: 2, index: 2);    //moves to 3rd index. So, to delete playgrounds
                                //from my street, i remove 3rd index.
street2.listOfBuildings();        //Listing buildings except removing ones.
street2.remainingLength();
street2.numberAndRatioPlaygrounds();
street2.lengthOfMarketHouseOffice();
```

```
    Left Side Buildings                Right Side Buildings
----------------------------        ----------------------------
        House                               Office

        House                               Office

        Market                              Market

        House

Total Remanining Length is: 26

Number of Playgrounds in the Street: 0
Ratio of Playgrounds in the Street: 0,00

Total Length of Street Occupied by the Houses: 24
Total Length of Street Occupied by the Offices: 27
Total Length of Street Occupied by the Markets: 23
```

```
street2.remove( pos: 1, index: 0);    //I'm removing all buildings from street.
street2.remove( pos: 1, index: 0);    //I just remove 0th indexes, because when i remove an element from
street2.remove( pos: 1, index: 0);    //street, next elements shifts automatically to the left.
street2.remove( pos: 1, index: 0);

street2.remove( pos: 2, index: 0);
street2.remove( pos: 2, index: 0);
street2.remove( pos: 2, index: 0);

street2.listOfBuildings();        //Listing empty buildings.
street2.remainingLength();
street2.numberAndRatioPlaygrounds();
street2.lengthOfMarketHouseOffice();
```

```
    Left Side Buildings                Right Side Buildings
----------------------------        ----------------------------
Total Remanining Length is: 100

Number of Playgrounds in the Street: 0
Ratio of Playgrounds in the Street: 0,00

Total Length of Street Occupied by the Houses: 0
Total Length of Street Occupied by the Offices: 0
Total Length of Street Occupied by the Markets: 0
```

```
Enter the position of the building(1 for left, 2 for right
asfga
Entered input is invalid, try again.
-------------------------------------------
```

```
Enter the number of rooms of the house:
wrongInput
Entered input is invalid, try again.
```

```
Select an option:
1) Run Interactive Program
2) Run Drive Code
3) Exit
5
Entered input is out of range, try again.
```

```
Enter the side of building you want to delete(1 For Left, 2 For Right):
2

Enter the index of building you want to delete:
5
Entered input is out of range, try again.
```

```
Select an option:
1) Run Interactive Program
2) Run Drive Code
3) Exit
asdf
Exception handled. Entered input is invalid, try again.
```

```
Choose one of the followings:

 1) Edit Mode
 2) View Mode
 3) Return to Main Menu...
deneme
Exception handled. Entered input is invalid, try again.
```

```
Choose one of the followings:
1) Display the total remaining length of lands on the street
2) Display the list of buildings on the street
3) Display the number and ratio of length of playgrounds in the street
4) Calculate the total length of street occupied by the markets, houses or offices
5) Display the skyline silhouette of the street
6) Display focused informations of buildings
7) Return to Menu
-5
Entered input is out of range, try again.
```

```
Choose one of the followings:
1) Display the total remaining length of lands on the street
2) Display the list of buildings on the street
3) Display the number and ratio of length of playgrounds in the street
4) Calculate the total length of street occupied by the markets, houses or offices
5) Display the skyline silhouette of the street
6) Display focused informations of buildings
7) Return to Menu
asdf
Entered input is invalid, try again.
```

```
Choose one of the followings:
1) Display the total remaining length of lands on the street
2) Display the list of buildings on the street
3) Display the number and ratio of length of playgrounds in the street
4) Calculate the total length of street occupied by the markets, houses or offices
5) Display the skyline silhouette of the street
6) Display focused informations of buildings
7) Return to Menu
7

Choose one of the followings:

 1) Edit Mode
 2) View Mode
 3) Return to Main Menu...
```

```
Enter Street Length =
wrong Input
Exception handled. Entered input is invalid, try again.
Enter Street Length =
```

5. Running Command and Results

The file should be unzipped and opened as a Project in IntellijIdea. All the user methods are accessible on driver code.

I do every part of homework properly.

6. Time Complexities

a) Homework-1 Time Complexity

- All getter/setter's time complexity of Homework-1 is θ(1)

```
public boolean isConflict(Building b){  //This method checks if two building is conflict. If so, returns true, otherwise return false.
    if(b.getPosition() == 1){       //If getPosition() == 1 --> Buildings are on the left
        for(int i = 0 ; i < this.leftSize ; i++){
            if((this.leftBuildings[i].getStartPoint() > b.getStartPoint() &&    //I checked all situations that buildings are conflict.
                this.leftBuildings[i].getStartPoint() < b.getEndPoint()) ||
                (this.leftBuildings[i].getEndPoint() > b.getStartPoint() &&
                this.leftBuildings[i].getEndPoint() < b.getEndPoint()) ||
                (this.leftBuildings[i].getStartPoint() < b.getStartPoint() &&
                this.leftBuildings[i].getEndPoint() > b.getEndPoint()) ||
                (this.leftBuildings[i].getStartPoint() > b.getStartPoint() &&
                this.leftBuildings[i].getEndPoint() < b.getEndPoint())){
                    return true;
            }
        }
    }
    else{       //Else --> Buildings are on the left
        for(int i = 0 ; i < this.rightSize ; i++){
            if((this.rightBuildings[i].getStartPoint() > b.getStartPoint() &&   //I checked all situations that buildings are conflict.
                this.rightBuildings[i].getStartPoint() < b.getEndPoint()) ||
                (this.rightBuildings[i].getEndPoint() > b.getStartPoint() &&
                this.rightBuildings[i].getEndPoint() < b.getEndPoint()) ||
                (this.rightBuildings[i].getStartPoint() < b.getStartPoint() &&
                this.rightBuildings[i].getEndPoint() > b.getEndPoint()) ||
                (this.rightBuildings[i].getStartPoint() > b.getStartPoint() &&
                this.rightBuildings[i].getEndPoint() < b.getEndPoint())){
                    return true;
            }
        }
    }
    return false;
}
```

- isConflict(Building b) →Time Complexity is **θ(n)**

```
public boolean isBuildingFitsToStreet(Building b){  //Checks if building fits to street.
    return (b.getStartPoint() + b.getLength() <= this.getStreetLength());
}
```

- isBuildingFitsToStreet(Building b) → **θ(1)**

- add(Building var) → O(n)

```
public boolean add(Building var){

    if(checkInputsValidity(var)){   //If inputs are valid, continues. Then, returns false.
        if(var.getPosition() == 1){
            if(leftSize == leftCapacity){   //If capacity is full, I reallocate
                reallocate( var: 1);          //and expand my building array
            }
            leftBuildings[leftSize] = var;  //New element is added to array.
            leftSize++;     //Size increased.
        }
        else{
            if(rightSize == rightCapacity){
                reallocate( var: 2);
            }
            rightBuildings[rightSize] = var;
            rightSize++;
        }
        return true;
    }
    return false;
}
```

- Remove(int pos, int index) --> θ(n)

```
public boolean remove(int pos, int index){
    int i;
    if(pos == 1){   //If pos == 1 --> building that will be removed is on the left.
        for(i=index+1;i<leftSize;i++){
            leftBuildings[i-1] = leftBuildings[i];  //I removed that element with shifting
        }
        leftBuildings[leftSize-1] = null;
        leftSize--;     //Size decreased
    }
    else{
        for(i=index+1;i<rightSize;i++){
            rightBuildings[i-1] = rightBuildings[i];
        }
        rightBuildings[rightSize-1] = null;
        rightSize--;
    }
    return true;
}
```

- Reallocate(int var) → θ(n)

```
public void reallocate(int var){        //I create a temp building array to keep original array.
    Building[] tempBuildings;           //I assigned building array to temp array and reinitialize
    if(var == 1){                       //building array with double capacity. And save old datas,
        tempBuildings = leftBuildings;  //I copied temp array datas to building array.
        this.leftBuildings = new Building[leftCapacity*2];
        for(int i = 0 ; i < leftSize ; i++){
            leftBuildings[i] = tempBuildings[i];
        }
        leftCapacity = leftCapacity * 2;
    }
    else{
        tempBuildings = rightBuildings;
        this.rightBuildings = (Building[]) new Object[rightCapacity*2];
        for(int i = 0 ; i < rightSize ; i++){
            rightBuildings[i] = tempBuildings[i];
        }
        rightCapacity = rightCapacity * 2;
    }
}
```

- totalHouseLength() → θ(n)

```java
public int totalHouseLength(){
    int total = 0;
    Building right, left;
    for(int i = 0 ; i < this.getRightSize() ; i++) {    //Searched rightBuilding array
        right = this.getRightBuilding(i);
        if (right instanceof House) {                   //if it is instance of House object,
            total = total + right.getLength();          //total = total + object.getLength()
        }
    }
    for(int j = 0 ; j < this.getLeftSize() ; j++){      //Same for leftBuilding array
        left = this.getLeftBuilding(j);
        if(left instanceof House){
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalOfficeLength() → θ(n)

```java
public int totalOfficeLength(){     //I do the same things as totalHouseLength method
    int total = 0;
    Building right, left;
    for(int i = 0 ; i < this.getRightSize() ; i++) {
        right = this.getRightBuilding(i);
        if (right instanceof Office) {
            total = total + right.getLength();
        }
    }
    for(int j = 0 ; j < this.getLeftSize() ; j++){
        left = this.getLeftBuilding(j);
        if(left instanceof Office){
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalMarketLength() → θ(n)

```java
public int totalMarketLength(){     //I do the same things as totalHouseLength method
    int total = 0;
    Building right, left;
    for(int i = 0 ; i < this.getRightSize() ; i++) {
        right = this.getRightBuilding(i);
        if (right instanceof Market) {
            total = total + right.getLength();
        }
    }
    for(int j = 0 ; j < this.getLeftSize() ; j++){
        left = this.getLeftBuilding(j);
        if(left instanceof Market){
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalPlaygroundLength() → θ(n)

```
public int totalPlaygroundLength(){     //I do the same things as totalHouseLength method
    int total = 0;
    Building right, left;
    for(int i = 0 ; i < this.getRightSize() ; i++) {
        right = this.getRightBuilding(i);
        if (right instanceof Playground) {
            total = total + right.getLength();
        }
    }
    for(int j = 0 ; j < this.getLeftSize() ; j++){
        left = this.getLeftBuilding(j);
        if(left instanceof Playground){
            total = total + left.getLength();
        }
    }
    return total;
}
```

- remainingLength() → θ(n)

```
public void remainingLength(){      //Street has two side. Thus, i multiply the street length by 2. Then, i extracted building lengths.
    int remaining = (this.getStreetLength()*2) - (totalHouseLength() + totalOfficeLength() +
                    totalMarketLength() + totalPlaygroundLength());
    System.out.println("Total Remanining Length is: " + remaining);
}
```

- listOfBuildings() → θ(n)

```
public void listOfBuildings(){      //I printed list of buildings here
    Building right, left;
    Formatter formatter = new Formatter();
    int biggerSize;
    if(this.getLeftSize() > this.getRightSize())
        biggerSize = leftSize;
    else{
        biggerSize = rightSize;
    }
    System.out.println("\tLeft Side Buildings\t\t\t\t\tRight Side Buildings");
    System.out.println("---------------------------\t\t---------------------------");

    for(int i = 0 ; i < biggerSize ; i++){
        System.out.print("\t\t");
        if(!(i >= this.getLeftSize())){
            left = this.getLeftBuilding(i);
            System.out.print(left.getClass().getName());
        }
        System.out.print("\t\t\t\t\t\t\t\t");
        if(!(i >= this.getRightSize())){
            right = this.getRightBuilding(i);
            System.out.print(right.getClass().getName());
        }
        System.out.print("\n\n");
    }
}
```

- numberAndRatioPlaygrounds() → θ(n)

```
public void numberAndRatioPlaygrounds(){
    int number = 0;
    double ratioOfPlayground = 0.0;
    Building right, left;
    for(int i = 0 ; i < this.getRightSize() ; i++) {
        right = this.getRightBuilding(i);
        if (right instanceof Playground) {
            number++;
        }
    }
    for(int j = 0 ; j < this.getLeftSize() ; j++){
        left = this.getLeftBuilding(j);
        if(left instanceof Playground){
            number++;
        }
    }
    ratioOfPlayground = (float) totalPlaygroundLength()/(this.streetLength*2);
    System.out.println("\nNumber of Playgrounds in the Street: " + number);
    System.out.println("Ratio of Playgrounds in the Street: " + (String.format("%.2f", ratioOfPlayground)));
}
```

- lengthOfMarketHouseOffice() → θ(n)

```
public void lengthOfMarketHouseOffice(){
    System.out.println("\nTotal Length of Street Occupied by the Houses: "+totalHouseLength());
    System.out.println("Total Length of Street Occupied by the Offices: "+totalOfficeLength());
    System.out.println("Total Length of Street Occupied by the Markets: "+totalMarketLength()+"\n");
}
```

- checkInputsValidity → θ(n)

```
public boolean checkInputsValidity(Building b){
    return (this.isBuildingFitsToStreet(b) &&          //I concatenate all validity methods in here.
            b.isInputValid() && !this.isConflict(b));
}
```

b) Homework-3 Part-1 Time Complexity

- All getter/setter's time complexity of Homework-3 Part-1 is θ(1)

```
public boolean isConflict(Building b){  //This method checks if two building is conflict. If so, returns true, otherwise return false.
    if(b.getPosition() == 1){       //If getPosition() == 1 --> Buildings are on the left
        for(int i = 0 ; i < this.leftSize ; i++){
            if((this.leftBuildings[i].getStartPoint() > b.getStartPoint() &&    //I checked all situations that buildings are conflict.
                this.leftBuildings[i].getStartPoint() < b.getEndPoint()) ||
                (this.leftBuildings[i].getEndPoint() > b.getStartPoint() &&
                this.leftBuildings[i].getEndPoint() < b.getEndPoint()) ||
                (this.leftBuildings[i].getStartPoint() < b.getStartPoint() &&
                this.leftBuildings[i].getEndPoint() > b.getEndPoint()) ||
                (this.leftBuildings[i].getStartPoint() > b.getStartPoint() &&
                this.leftBuildings[i].getEndPoint() < b.getEndPoint())){
                    return true;
            }
        }
    }
    else{       //Else --> Buildings are on the left
        for(int i = 0 ; i < this.rightSize ; i++){
            if((this.rightBuildings[i].getStartPoint() > b.getStartPoint() &&   //I checked all situations that buildings are conflict.
                this.rightBuildings[i].getStartPoint() < b.getEndPoint()) ||
                (this.rightBuildings[i].getEndPoint() > b.getStartPoint() &&
                this.rightBuildings[i].getEndPoint() < b.getEndPoint()) ||
                (this.rightBuildings[i].getStartPoint() < b.getStartPoint() &&
                this.rightBuildings[i].getEndPoint() > b.getEndPoint()) ||
                (this.rightBuildings[i].getStartPoint() > b.getStartPoint() &&
                this.rightBuildings[i].getEndPoint() < b.getEndPoint())){
                    return true;
            }
        }
    }
    return false;
}
```

- isConflict(Building b) →Time Complexity is **θ(n)**

```
public boolean isBuildingFitsToStreet(Building b){  //Checks if building fits to street.
    return (b.getStartPoint() + b.getLength() <= this.getStreetLength());
}
```

- isBuildingFitsToStreet(Building b) → θ(1)

- add(Building var) → θ(n)

```
public boolean add(Building var){
    if(checkInputsValidity(var)){   //If inputs are valid, continues. Then, returns false.
        if(var.getPosition() == 1){
            leftBuildings.add(var);  //New element is added to ArrayList.
        }
        else{
            rightBuildings.add(var);
        }
        return true;
    }
    else{
        System.out.println("Building's one or more input is invalid, please add valid building");
    }
    return false;
}
```

- Remove(int pos, int index) → O(n)

```
public void remove(int pos, int index) throws IndexOutOfBoundsException{
    if(pos == 1){   //If pos == 1 --> building that will be removed is on
        leftBuildings.remove(index);
    }
    else{
        rightBuildings.remove(index);
    }
}
```

- totalHouseLength() → θ(n)

```
public int totalOfficeLength(){     //I do the same things
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Office) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Office) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalOfficeLenght() → θ(n)

```java
public int totalOfficeLength(){     //I do the same things
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Office) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Office) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalMarketLength() → θ(n)

```java
public int totalMarketLength(){     //I do the same things as
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Market) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Market) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalPlaygroundLength() → θ(n)

```
public int totalPlaygroundLength(){     //I do the same thi
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Playground) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Playground) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- remainingLength() → θ(n)

```
public void remainingLength(){      //Street has two side. Thus, I multiply the street length
    int remaining = (this.getStreetLength()*2) - (totalHouseLength() + totalOfficeLength() +
            totalMarketLength() + totalPlaygroundLength());
    System.out.println("Total Remanining Length is: " + remaining);
}
```

- listOfBuildings() → θ(n)

```
public void listOfBuildings(){
    Building right, left;
    int biggerSize = Math.max(this.leftBuildings.size(), this.rightBuildings.size());
    System.out.println("\tLeft Side Buildings\t\t\t\t\tRight Side Buildings");
    System.out.println("--------------------------\t\t--------------------------");

    for(int i = 0 ; i < biggerSize ; i++){
        System.out.print("\t\t");
        if(!(i >= this.leftBuildings.size())){
            left = this.leftBuildings.get(i);
            System.out.print(left.getClass().getName());
        }
        System.out.print("\t\t\t\t\t\t\t\t");
        if(!(i >= this.rightBuildings.size())){
            right = this.rightBuildings.get(i);
            System.out.print(right.getClass().getName());
        }
        System.out.print("\n\n");
    }
}
```

- numberAndRatioPlaygrounds() → θ(n)

```
public void numberAndRatioPlaygrounds(){
    int number = 0;
    double ratioOfPlayground = 0.0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Playground) {
            number++;
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Playground) {
            number++;
        }
    }
    ratioOfPlayground = (float) totalPlaygroundLength()/(this.streetLength*2);
    System.out.println("\nNumber of Playgrounds in the Street: " + number);
    System.out.println("Ratio of Playgrounds in the Street: " + (String.format("%.2f", ratioOfPlayground)));
}
```

- lengthOfMarketHouseOffice() → θ(n)

```
public void lengthOfMarketHouseOffice(){
    System.out.println("\nTotal Length of Street Occupied by the Houses: "+totalHouseLength());
    System.out.println("Total Length of Street Occupied by the Offices: "+totalOfficeLength());
    System.out.println("Total Length of Street Occupied by the Markets: "+totalMarketLength()+"\n");
}
```

- checkInputsValidity() → θ(n)

```
public boolean checkInputsValidity(Building b){
    return (this.isBuildingFitsToStreet(b) &&
            b.isInputValid() && !this.isConflict(b));
}
```

c) Homework-3 Part-2 Time Complexity

- All getter/setter's time complexity of Homework-3 Part-2 is θ(1)

```
public boolean isConflict(Building b){  //This method checks if two building is conflict. If so, returns true, otherwise return false.
    if(b.getPosition() == 1){       //If getPosition() == 1 --> Buildings are on the left
        for (Building leftBuilding : this.leftBuildings) {
            if ((leftBuilding.getStartPoint() > b.getStartPoint() &&    //I checked all situations that buildings are conflict.
                    leftBuilding.getStartPoint() < b.getEndPoint()) ||
                    (leftBuilding.getEndPoint() > b.getStartPoint() &&
                            leftBuilding.getEndPoint() < b.getEndPoint()) ||
                    (leftBuilding.getStartPoint() < b.getStartPoint() &&
                            leftBuilding.getEndPoint() > b.getEndPoint()) ||
                    (leftBuilding.getStartPoint() > b.getStartPoint() &&
                            leftBuilding.getEndPoint() < b.getEndPoint())) {
                return true;
            }
        }
    }
    else{       //Else --> Buildings are on the left
        for (Building rightBuilding : this.rightBuildings) {
            if ((rightBuilding.getStartPoint() > b.getStartPoint() &&   //I checked all situations that buildings are conflict.
                    rightBuilding.getStartPoint() < b.getEndPoint()) ||
                    (rightBuilding.getEndPoint() > b.getStartPoint() &&
                            rightBuilding.getEndPoint() < b.getEndPoint()) ||
                    (rightBuilding.getStartPoint() < b.getStartPoint() &&
                            rightBuilding.getEndPoint() > b.getEndPoint()) ||
                    (rightBuilding.getStartPoint() > b.getStartPoint() &&
                            rightBuilding.getEndPoint() < b.getEndPoint())) {
                return true;
            }
        }
    }
    return false;
```

- isConflict(Building b) →Time Complexity is O**(n)**

```
public boolean isBuildingFitsToStreet(Building b){  //Checks if building fits to street.
    return (b.getStartPoint() + b.getLength() <= this.getStreetLength());
}
```

- isBuildingFitsToStreet(Building b) → θ(1)

- add(Building var) → θ(n)

```
public boolean add(Building var){
    if(checkInputsValidity(var)){   //If inputs are valid, continues. Then, returns false.
        if(var.getPosition() == 1){
            leftBuildings.add(var);  //New element is added to ArrayList.
        }
        else{
            rightBuildings.add(var);
        }
        return true;
    }
    else{
        System.out.println("Building's one or more input is invalid, please add valid building");
    }
    return false;
}
```

- Remove(int pos, int index) → O(n)

```
public void remove(int pos, int index) throws IndexOutOfBoundsException{
    if(pos == 1){   //If pos == 1 --> building that will be removed is on
        leftBuildings.remove(index);
    }
    else{
        rightBuildings.remove(index);
    }
}
```

- totalHouseLength() → θ(n)

```
public int totalOfficeLength(){     //I do the same things
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Office) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Office) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalOfficeLenght() → θ(n)

```
public int totalOfficeLength(){     //I do the same things
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Office) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Office) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalMarketLength() → θ(n)

```
public int totalMarketLength(){     //I do the same things as
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Market) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Market) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalPlaygroundLength() → θ(n)

```
public int totalPlaygroundLength(){     //I do the same thi
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Playground) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Playground) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- remainingLength() → θ(n)

```
public void remainingLength(){      //Street has two side. Thus, I multiply the street length
    int remaining = (this.getStreetLength()*2) - (totalHouseLength() + totalOfficeLength() +
            totalMarketLength() + totalPlaygroundLength());
    System.out.println("Total Remanining Length is: " + remaining);
}
```

- listOfBuildings() → O(n^2)

```
public void listOfBuildings(){
    Building right, left;
    int biggerSize = Math.max(this.leftBuildings.size(), this.rightBuildings.size());
    System.out.println("\tLeft Side Buildings\t\t\t\t\tRight Side Buildings");
    System.out.println("--------------------------\t\t--------------------------");

    for(int i = 0 ; i < biggerSize ; i++){
        System.out.print("\t\t");
        if(!(i >= this.leftBuildings.size())){
            left = this.leftBuildings.get(i);
            System.out.print(left.getClass().getName());
        }
        System.out.print("\t\t\t\t\t\t\t\t");
        if(!(i >= this.rightBuildings.size())){
            right = this.rightBuildings.get(i);
            System.out.print(right.getClass().getName());
        }
        System.out.print("\n\n");
    }
}
```

- numberAndRatioPlaygrounds() → θ(n)

```
public void numberAndRatioPlaygrounds(){
    int number = 0;
    double ratioOfPlayground = 0.0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Playground) {
            number++;
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Playground) {
            number++;
        }
    }
    ratioOfPlayground = (float) totalPlaygroundLength()/(this.streetLength*2);
    System.out.println("\nNumber of Playgrounds in the Street: " + number);
    System.out.println("Ratio of Playgrounds in the Street: " + (String.format("%.2f", ratioOfPlayground)));
}
```

- lengthOfMarketHouseOffice() → θ(n)

```
public void lengthOfMarketHouseOffice(){
    System.out.println("\nTotal Length of Street Occupied by the Houses: "+totalHouseLength());
    System.out.println("Total Length of Street Occupied by the Offices: "+totalOfficeLength());
    System.out.println("Total Length of Street Occupied by the Markets: "+totalMarketLength()+"\n");
}
```

- checkInputsValidity() → θ(n)

```
public boolean checkInputsValidity(Building b){
    return (this.isBuildingFitsToStreet(b) &&
            b.isInputValid() && !this.isConflict(b));
}
```

d) Homework-3 Part-3 Time Complexity

```
public boolean isConflict(Building b){  //This method checks if two building is conflict. If so, returns true, otherwise return false.
    if(b.getPosition() == 1){       //If getPosition() == 1 --> Buildings are on the left
        for (Building leftBuilding : this.leftBuildings) {
            if ((leftBuilding.getStartPoint() > b.getStartPoint() &&    //I checked all situations that buildings are conflict.
                    leftBuilding.getStartPoint() < b.getEndPoint()) ||
                    (leftBuilding.getEndPoint() > b.getStartPoint() &&
                            leftBuilding.getEndPoint() < b.getEndPoint()) ||
                    (leftBuilding.getStartPoint() < b.getStartPoint() &&
                            leftBuilding.getEndPoint() > b.getEndPoint()) ||
                    (leftBuilding.getStartPoint() > b.getStartPoint() &&
                            leftBuilding.getEndPoint() < b.getEndPoint())) {
                return true;
            }
        }
    }
    else{       //Else --> Buildings are on the left
        for (Building rightBuilding : this.rightBuildings) {
            if ((rightBuilding.getStartPoint() > b.getStartPoint() &&    //I checked all situations that buildings are conflict.
                    rightBuilding.getStartPoint() < b.getEndPoint()) ||
                    (rightBuilding.getEndPoint() > b.getStartPoint() &&
                            rightBuilding.getEndPoint() < b.getEndPoint()) ||
                    (rightBuilding.getStartPoint() < b.getStartPoint() &&
                            rightBuilding.getEndPoint() > b.getEndPoint()) ||
                    (rightBuilding.getStartPoint() > b.getStartPoint() &&
                            rightBuilding.getEndPoint() < b.getEndPoint())) {
                return true;
            }
        }
    }
    return false;
```

- isConflict(Building b) →Time Complexity is **O(n)**

```
public boolean isBuildingFitsToStreet(Building b){  //Checks if building fits to street.
    return (b.getStartPoint() + b.getLength() <= this.getStreetLength());
}
```

- isBuildingFitsToStreet(Building b) → θ(1)

- add(Building var) → θ(n)

```
public boolean add(Building var){
    if(checkInputsValidity(var)){   //If inputs are valid, continues. Then, returns false.
        if(var.getPosition() == 1){
            leftBuildings.add(var);  //New element is added to ArrayList.
        }
        else{
            rightBuildings.add(var);
        }
        return true;
    }
    else{
        System.out.println("Building's one or more input is invalid, please add valid building");
    }
    return false;
}
```

- Remove(int pos, int index) → O(n)

```
public void remove(int pos, int index) throws IndexOutOfBoundsException{
    if(pos == 1){   //If pos == 1 --> building that will be removed is on
        leftBuildings.remove(index);
    }
    else{
        rightBuildings.remove(index);
    }
}
```

- totalHouseLength() → θ(n)

```
public int totalOfficeLength(){     //I do the same things
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Office) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Office) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalOfficeLenght() → θ(n)

```
public int totalOfficeLength(){     //I do the same things
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Office) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Office) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalMarketLength() → θ(n)

```
public int totalMarketLength(){     //I do the same things as
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Market) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Market) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- totalPlaygroundLength() → θ(n)

```java
public int totalPlaygroundLength(){     //I do the same thi
    int total = 0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Playground) {
            total = total + right.getLength();
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Playground) {
            total = total + left.getLength();
        }
    }
    return total;
}
```

- remainingLength() → θ(n)

```java
public void remainingLength(){      //Street has two side. Thus, I multiply the street length
    int remaining = (this.getStreetLength()*2) - (totalHouseLength() + totalOfficeLength() +
            totalMarketLength() + totalPlaygroundLength());
    System.out.println("Total Remanining Length is: " + remaining);
}
```

- listOfBuildings() → O(n^2)

```java
public void listOfBuildings(){
    Building right, left;
    int biggerSize = Math.max(this.leftBuildings.size(), this.rightBuildings.size());
    System.out.println("\tLeft Side Buildings\t\t\t\t\tRight Side Buildings");
    System.out.println("--------------------------\t\t--------------------------");

    for(int i = 0 ; i < biggerSize ; i++){
        System.out.print("\t\t");
        if(!(i >= this.leftBuildings.size())){
            left = this.leftBuildings.get(i);
            System.out.print(left.getClass().getName());
        }
        System.out.print("\t\t\t\t\t\t\t\t");
        if(!(i >= this.rightBuildings.size())){
            right = this.rightBuildings.get(i);
            System.out.print(right.getClass().getName());
        }
        System.out.print("\n\n");
    }
}
```

- numberAndRatioPlaygrounds() → θ(n)

```
public void numberAndRatioPlaygrounds(){
    int number = 0;
    double ratioOfPlayground = 0.0;
    Building right, left;
    for (Building rightBuilding : this.rightBuildings) {
        right = rightBuilding;
        if (right instanceof Playground) {
            number++;
        }
    }
    for (Building leftBuilding : this.leftBuildings) {
        left = leftBuilding;
        if (left instanceof Playground) {
            number++;
        }
    }
    ratioOfPlayground = (float) totalPlaygroundLength()/(this.streetLength*2);
    System.out.println("\nNumber of Playgrounds in the Street: " + number);
    System.out.println("Ratio of Playgrounds in the Street: " + (String.format("%.2f", ratioOfPlayground)));
}
```

- lengthOfMarketHouseOffice() → θ(n)

```
public void lengthOfMarketHouseOffice(){
    System.out.println("\nTotal Length of Street Occupied by the Houses: "+totalHouseLength());
    System.out.println("Total Length of Street Occupied by the Offices: "+totalOfficeLength());
    System.out.println("Total Length of Street Occupied by the Markets: "+totalMarketLength()+"\n");
}
```

- checkInputsValidity() → θ(n)

```
public boolean checkInputsValidity(Building b){
    return (this.isBuildingFitsToStreet(b) &&
            b.isInputValid() && !this.isConflict(b));
}
```

- Get(index) → O(n)

```
@Override
public T get(int index) {
    return getNode(index).getData();
}
```

- addTrash(T t) →O(n)

```
public boolean addTrash(T t){
    Node<T> temp;
    if(size == 0){
        addTrashFirst(t);
    }
    else{
        addTrashLast(t);
    }
    return true;
}
```

- addTrashFirst(T t) → θ(1)

```
public void addTrashFirst(T t){
    root = new Node<>(t);
    tail = root;
    size++;
}
```

- addTrashLast(T t) → θ(1)

```
public void addTrashLast(T t){
    tail.next = new Node<>(t);
    tail = tail.next;
    size++;
}
```

- add(T t) → O(n)

```
public boolean add(T t){
    boolean founded = false;
    Node<T> temp;
    int i = 0;
    if(trash.size != 0){
        for (; i < size ; i++){
            if(t.equals(trash.getTrashNode(i).getData())){
                founded = true;
                break;
            }
        }
    }

    if(size == 0){
        if(founded){
            addFirst((T) trash.getTrashNode(i).getData());
            trash.removeFromTrash(i);
        }
        else{
            addFirst(t);
        }
        return true;
    }
    if(founded){
        addLast((T) trash.getTrashNode(i).getData());
        trash.removeFromTrash(i);
    }
    else{
        addLast(t);
    }
    return true;
```

- addFirst(T t) → θ(1)

```
private void addFirst(T t){
    root = new Node<>(t);
    tail = root;
    size++;
}
```

- addLast(T t) → θ(1)

```
private void addLast(T t){
    tail.next = new Node<>(t);
    tail = tail.next;
    size++;
}
```

- getData(int index) → O(n)

```
public T getData(int index) {
    if(index < 0 || index >= size){
        throw new IndexOutOfBoundsException();
    }
    Node<T> temp = root;
    for(int i = 0 ; i < index ; i++){
        temp = temp.next;
    }
    return temp.getData();
}
```

- getTrashNode(int index) →O(n)

```
public Node<T> getTrashNode(int index){
    Node<T> temp = root;
    for(int i  = 0 ; i < index && temp != null; i++){
        temp = temp.next;
    }
    return temp;
}
```

- getNode(int index) →O(n)

```
public Node<T> getNode(int index){
    if(index < 0 || index >= size){
        throw new IndexOutOfBoundsException();
    }
    Node<T> temp = root;
    for(int i  = 0 ; i < index ; i++){
        temp = temp.next;
    }
    return temp;
}
```

- indexOf(Object o) →O(n)

```
public int indexOf(Object o){
    Node<T> temp = root;
    for(int i = 0 ; i < size ; i++){
        if(temp.data.equals((T) o)){
            return i;
        }
        temp = temp.next;
    }
    return -1;
}
```

- removeFromTrash(int index) →O(n)

```
public T removeFromTrash(int index){
    Node<T> temp,removed;
    if(index < 0 || index >= size){
        throw new IndexOutOfBoundsException();
    }
    if(getTrashNode( index: 0).getData().equals(get(index))){
        return removeFirst();
    }
    return removeAfter(getTrashNode( index: index-1));
}
```

- removeFromTrashFirst() → θ(1)

```
public T removeFromTrashFirst(){
    Node<T> temp = root;
    if(root != null){
        if(root.next == null){
            root = null;
        }
        else{
            root = root.next;
            temp.next = null;
        }
        size--;
        return temp.data;
    }
    return null;
}
```

- removeFromTrashAfter(Node<T> temp) →O(n)

```
public T removeFromTrashAfter(Node<T> temp){
    Node<T> removedNode = temp.next;
    if(temp != null){
        if(removedNode.next == null){
            temp.next = null;
        }
        else{
            temp.next = removedNode.next;
            removedNode.next = null;
        }
        size--;
        return removedNode.data;
    }
    return null;
}
```

- remove(int index) →O(n)

```
@Override
public T remove(int index){
    Node<T> temp,removed;
    if(index < 0 || index >= size){
        throw new IndexOutOfBoundsException();
    }
    if(getNode( index: 0).getData().equals(get(index))){
        return removeFirst();
    }
    return removeAfter(getNode( index: index-1));
}
```

- removeFirst() → O(n)

```
public T removeFirst(){
    Node<T> temp = root;
    if(root != null){
        if(root.next == null){
            root = null;
        }
        else{
            root = root.next;
            temp.next = null;
        }
        trash.addTrash(temp);
        size--;
        trash.size++;
        return temp.data;
    }
    return null;
}
```

- removeAfter(Node<T> temp) → O(n)

```
public T removeAfter(Node<T> temp){
    Node<T> removedNode = temp.next;
    if(temp != null){
        if(removedNode.next == null){
            temp.next = null;
        }
        else{
            temp.next = removedNode.next;
            removedNode.next = null;
        }
        trash.addTrash(removedNode);
        trash.size++;
        size--;
        return removedNode.data;
    }
    return null;
}
```