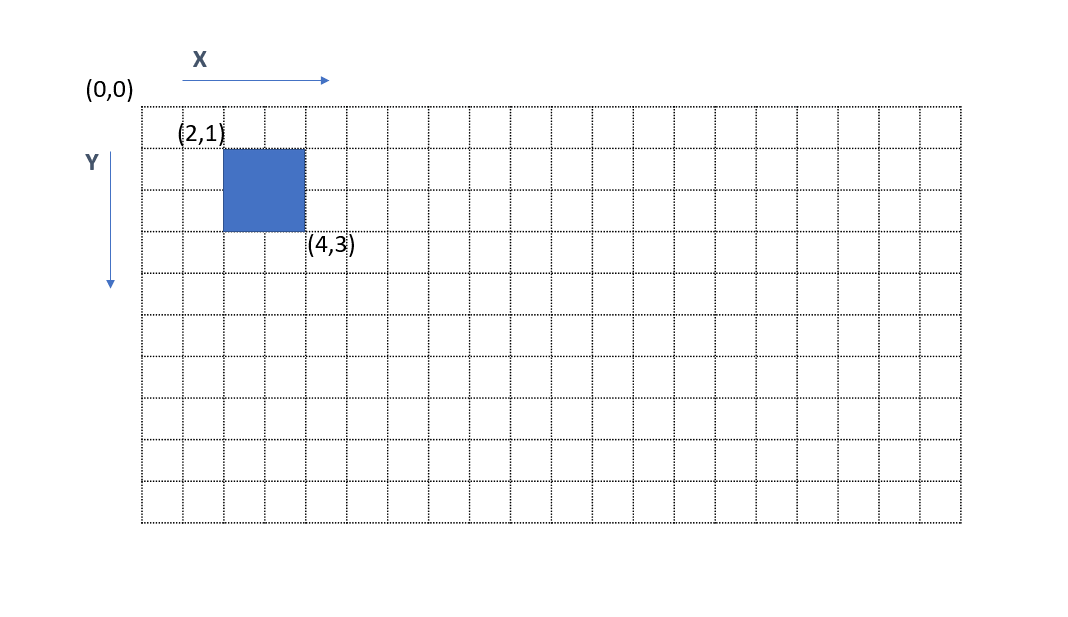
# Homework: Interval Trees, Quad Trees, K-d Trees

This document defines the **homework assignments** for the ["Data Structures" course @ Software University](https://softuni.bg/trainings/1147/Data-Structures-June-2015). Please submit a single zip / rar / 7z archive holding the solutions (source code) of all below described problems.

## Rectangle Coordinates

All of the problems below assume that a square object with coordinates **X1**, **Y1**, **X2**, **Y2** represents an object with upper left corner **X1**, **Y1** and lower right corner **X2**, **Y2** and for each object that **X1** < **X2** and **Y1** < **Y2**



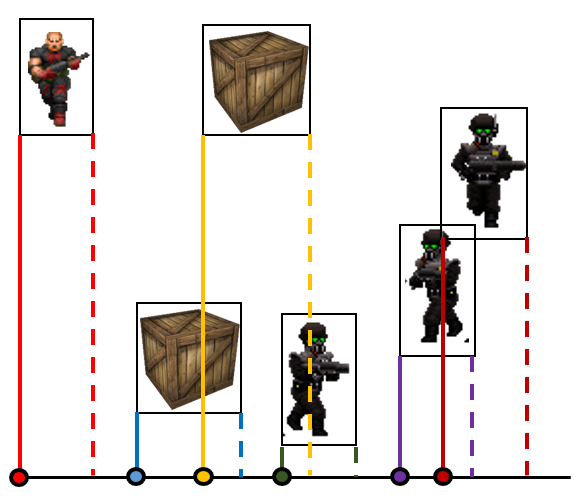
## Rectangle Intersection

You can find a visualization and Boolean conditions for rectangle intersection [here](https://silentmatt.com/rectangle-intersection/).

## Sweep and Prune

Sweep and prune is a simple algorithm for limiting collision detection in 2D space.

### Algorithm

* Store all objects in a **list** and **sort them by their X1 bound** (**see image**). This way every next object has a **greater or equal X1**.
  + For each object in the list, perform collision detection with all next objects whose **X1 is less or equal to the current object's X2**.
* Each game frame sort the list again and recheck for collisions in the same way. Use a sorting algorithm which performs fast on **nearly sorted collections**, such as [**insertion sort**](https://en.wikipedia.org/wiki/Insertion_sort)(best case **O(n)** complexity, worst case - **O(n2)**).
  + Here we assume **objects will not move too much** between game frames, that's why our **list will remain nearly sorted** at almost all times.

### Input

You must read and execute commands from the input.

* **add {name} {x1} {y1}** - adds a game object with the given name at coordinates (x1, x2). Assume all objects have **width = 10** and **height = 10**.
* **start** - starts the game (infinite while loop). After this point we may enter 2 commands, after which :
  + **tick** - does nothing
  + **move {name} {new-X1} {new-Y1}** - moves the game object to the new position

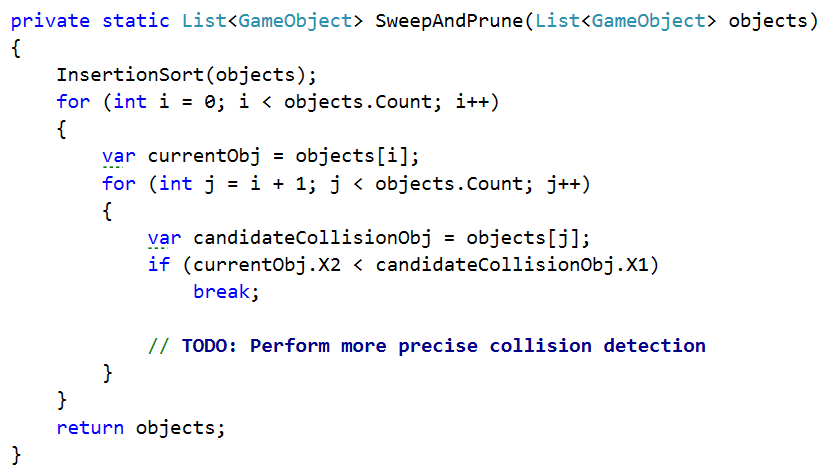
### Output

After each **tick** or **move** command you must **print** **which objects have collided** (two object collide if their **rectangle bounds overlap**) in the format "**{tick-count} {obj1} collides {obj2}**".

|  |  |
| --- | --- |
| **Input** | **Output** |
| **add pesho 0 0**  **add gosho 20 0**  **start**  **tick**  **tick**  **move pesho 20 0**  **tick**  **tick**  **end** | **(3) pesho collides with gosho**  **(4) pesho collides with gosho**  **(5) pesho collides with gosho** |
| **add pesho 0 0**  **add gosho 30 70**  **add batkata 60 60**  **add crate1 15 60**  **add crate2 23 40**  **add sir\_stanley\_royce 66 55**  **start**  **tick**  **move gosho 30 38**  **move sir\_stanley\_royce 38 42**  **end** | **(1) batkata collides with sir\_stanley\_royce**  **(2) crate2 collides with gosho**  **(2) batkata collides with sir\_stanley\_royce**  **(3) crate2 collides with gosho**  **(3) gosho collides with sir\_stanley\_royce** |

##### Hints (Click on the arrow to show)

* Go through all objects in the list, from **0** to **Count - 1**.
  + For **object A**, perform collision check with consequent objects in the list (objects B, C, D, etc.), **only if the 2 objects' X bounds** intersect.   
    If they do not, then **object A** will not intersect with any of the remaining objects too (since they are sorted by X1) -> so we break.



## Quad Tree

Implement a **Quad Tree** by following the guidelines from the [lab document](http://softuni.bg/downloads/svn/data-structures-and-algorithms/Data-Structures-May-2017/09.%20Data-Structures-Quad-Trees-KD-Trees-Interval-Trees/09.%20Quad-Tree-Exercise.docx). The tree should support only **Insert** and **Report** operations. Make sure all unit tests pass.

## \* Mass Effect Galaxy Map

Commander Shepard likes to browse his **Galaxy Map**. The map holds several **star clusters**, represented as **2D** **points**.

Your task is to implement the galaxy map as a **2-dimensional** [**k-d tree**](https://en.wikipedia.org/wiki/K-d_tree). The map should support **fast retrieval** of star clusters in a **given range** (see the **red circle** in the visualization below).

### Input

The input will consist of several lines:

* On the first line you will be given **n** - **the** **number of star clusters**.
* On the next **n** lines you will be given names and coordinates of the start clusters in the format   
  "**{name} {x} {y}**".
* On the last line you will be given Commander Shepard's cursor, represented as a circle in the format   
  "**report** **{x} {y} {radius}**".

### Output

Print all start clusters within the given cursor range (order does not matter).

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Visualization** |
| **7**  **Crescent\_Nebula 5.5 5**  **Krogan\_DMZ 6 15**  **Local\_Cluster 8 16**  **Kepler\_Verge 15 8**  **Hades\_Gamma 19 13**  **Exodus\_Cluster 12 13.5**  **Artemis\_Tau 15.4 17**  **report 6.3 16.5 2** | **Local\_Cluster**  **Krogan\_DMZ** |  |

##### Hints (Click on the arrow to show)

* Here is a good article on k-d trees - <http://algoviz.org/OpenDSA/Books/Everything/html/KDtree.html>.
* Randomly inserting elements can be dangerous as it can leadtoa **linked-list-**likestructure.
  + To distribute them more uniformly and **achieve a balanced structure**, **sort the points before** inserting them (by **X** for example).
  + Follow [this recursive algorithm](https://en.wikipedia.org/wiki/K-d_tree#Construction) to insert the points in a balanced way.