COMP3210/6210 – BIG DATA – S1 2024

Assignment 2 – Task 3 – Project Report

Unit Convenor: Yan Wang & Guanfeng Liu

Group Presentation Video link:

https://drive.google.com/file/d/1oRX3665ecLDmxoKTBUWaIrIp0zHLGOyP/view?usp=sharing

TABLE OF CONTENTS

1. Group Member Information

Name	Student ID	Email:	Assigned Task
Cam Tu (Jenna) Pham - Coordinator	46864598	camtu.pham @students.mq. edu.au	• Group Coordinator - in charge of reviewing and submitting project documents • Task 2 (Source Code and Output txt file) • Presentation on Task 2
Thi Minh Thi Tran	45773157	thi-minh-thi.tra n @students.mq. edu.au	• Task 1 (Source Code and Output txt file) • Presentation on Group assignment Introduction and Task 1 • Combine the video records to a complete presentation
Govardhan (Gov) Bharadwaj	46980164	govardhan.bhar adwaj @students.mq. edu.au	• Task 3 (Project Report) • Presentation on Task 3 and Group assignment Conclusion

2. Program Execution Requirements

2.1 Program Environment (e.g., OS, database, CPU, etc.)

TASK 1 SPECIFICATION DESCRIPTIONS

OS ENVIRONMENT	Windows 11 Home
CPU	Intel Core 8th Gen
RAM	16GB
LANGUAGE	Python
VERSION	Python 3.11.7
SOFTWARE	Visual Studio Code

TASK 2 SPECIFICATION DESCRIPTIONS

OS ENVIRONMENT	MacOS

CPU	Apple M3
RAM	8GB
LANGUAGE	Python
VERSION	Python 3.11.7
SOFTWARE	Visual Studio Code

TASK 3 SPECIFICATION DESCRIPTIONS

OS ENVIRONMENT	Windows 10
CPU	Intel i5-8400
RAM	16GB
LANGUAGE	N/A
VERSION	N/A
SOFTWARE	N/A

2.2 Input Files and Parameters (directory settings and other parameters)

Task 1

Input Parameters: restaurant_dataset.txt, query_points.txt

Location of Input Files: Located in one folder on the user's local computer, with the result being shared in a shared folder on Google Drive

Task 2
Input Parameters: city2.txt

Location of Input Files: Located in one folder on the user's local computer, with the result being shared in a shared folder on Google Drive

Task 3

Input Parameters: sample_data_for_BaB_algorithm_based_NN_search.txt, sample_data_for_BBS_algorithm_based_skyline_search.txt

Location of Input Files: Located in one folder on the user's local computer, with the result being shared in a shared folder on Google Drive

2.3 Additional Requirements

RESULTS AND EXECUTION TIMES

Task 1:

Nearest Neighbor Search - Sequential Scan Output

```
192   Query ID 191: 7 points found
193   Query ID 192: 15 points found
194   Query ID 193: 5 points found
195   Query ID 194: 6 points found
196   Query ID 195: 6 points found
197   Query ID 196: 13 points found
198   Query ID 197: 9 points found
199   Query ID 198: 4 points found
200   Query ID 199: 11 points found
201   Query ID 200: 5 points found
202
203   Total processing time: 0.0162 seconds
204   Average time per query: 0.0001 seconds
```

Nearest Neighbor Search - Best First Output

```
195   Query ID 194: Nearest Point ID 118 at (38.22, 121.97)
196   Query ID 195: Nearest Point ID 34 at (67.57, 78.15)
197   Query ID 196: Nearest Point ID 161 at (26.26, 150.59)
198   Query ID 197: Nearest Point ID 119 at (32.22, 83.28)
199   Query ID 198: Nearest Point ID 52 at (7.55, 89.62)
200   Query ID 199: Nearest Point ID 139 at (14.88, 151.80)
201   Query ID 200: Nearest Point ID 131 at (46.72, 137.95)
202
203   Total processing time: 0.0832 seconds
204   Average time per query: 0.0004 seconds
```

Nearest Neighbor Search - Divide and Conquer Output

```
193   Query ID 192: Nearest Point ID 126 at (30.33, 122.62) Distance: 3.40
194   Query ID 193: Nearest Point ID 100 at (37.72, 105.43) Distance: 4.03
195   Query ID 194: Nearest Point ID 118 at (38.22, 121.97) Distance: 3.05
196   Query ID 195: Nearest Point ID 172 at (67.63, 79.81) Distance: 3.64
197   Query ID 196: Nearest Point ID 161 at (26.26, 150.59) Distance: 2.27
198   Query ID 197: Nearest Point ID 119 at (32.22, 83.28) Distance: 2.58
199   Query ID 198: Nearest Point ID 52 at (7.55, 89.62) Distance: 4.59
200   Query ID 199: Nearest Point ID 139 at (14.88, 151.80) Distance: 1.69
201   Query ID 200: Nearest Point ID 131 at (46.72, 137.95) Distance: 4.76
202
203   Total processing time: 0.1551 seconds
204   Average time per query: 0.0008 seconds
```

Task 2

Sequential Scan Output

```
Query processing time: 0.000014 seconds
30247 338.34 800.0
60245 300.27 794.59
68846 302.96 799.87
72081 312.74 799.99
74464 300.0 790.74
169722 300.66 797.96
241342 300.35 796.02
263791 301.12 799.7
280895 300.2 793.77
```

BBS Algorithm Output

```
Query processing time: 0.000001 seconds
74464 300.0 790.74
280895 300.2 793.77
60245 300.27 794.59
241342 300.35 796.02
169722 300.66 797.96
263791 301.12 799.7
68846 302.96 799.87
72081 312.74 799.99
30247 338.34 800.0
```

BBS with Divide-and-Conquer Output

```
Query processing time: 0.000000 seconds
74464 300.0 790.74
280895 300.2 793.77
60245 300.27 794.59
241342 300.35 796.02
169722 300.66 797.96
263791 301.12 799.7
68846 302.96 799.87
72081 312.74 799.99
30247 338.34 800.0
```

3. Program Documentation

3.1 Program Organization

If your assignment consists of multiple files and/or classes, please provide brief, high-level descriptions of each file/class within your program, as illustrated below.

File Name	Description (detailed information)
restaurant_dataset.txt	Includes the ID, x, and y-values of 150,000 restaurants, with the x and y-values representing the geographical coordinates of the restaurants (longitude, latitude)
query_points.txt	Includes the ID, x, and y-value of 200 users who are interested in finding the nearest facilities, with the x and y-values representing the geographical coordinates of the restaurants (longitude, latitude)
city2.txt	Includes the ID, x, and y-values of 300,000 homes in a city, with the x-value indicating the cost of the home (in thousands of dollars), and the y-value indicating the size of the home (in square metres)
sample_data_for_BaB_algorithm_based_NN_search.txt	Includes the facility locations of 10 places, with their ID, x, and y-coordinates in the dataset (representing longitude and latitude), and one query point, presumably of a user attempting to find the nearest facility location
sample_data_for_BBS_algorithm_based_skyline_search.txt	Includes 15 points with ID, x and y-coordinates (where its definition is unspecified), with the aim being to find the skyline points.

Class	Description (detailed information)
“R-Tree”	The R-Tree class encompasses functions crucial for constructing and managing the R-tree structure. The R-tree is a balanced tree data structure designed for efficient spatial querying and is particularly useful for indexing multi-dimensional data such as spatial data or geographical coordinates
“rtree.index.Index”	The Index class from the R-Tree library is integral to the implementation of the BBS algorithm, providing the necessary functionality to construct the R-tree and perform efficient spatial queries to identify skyline points. We leveraged the “rtree.index.Index” class from the R-Tree library, which internally manages nodes as part of the R-tree data structure. The Index class abstracts away the lower-level details of node management, allowing us to interact with the R-tree at a higher level.

3.2 Function Description

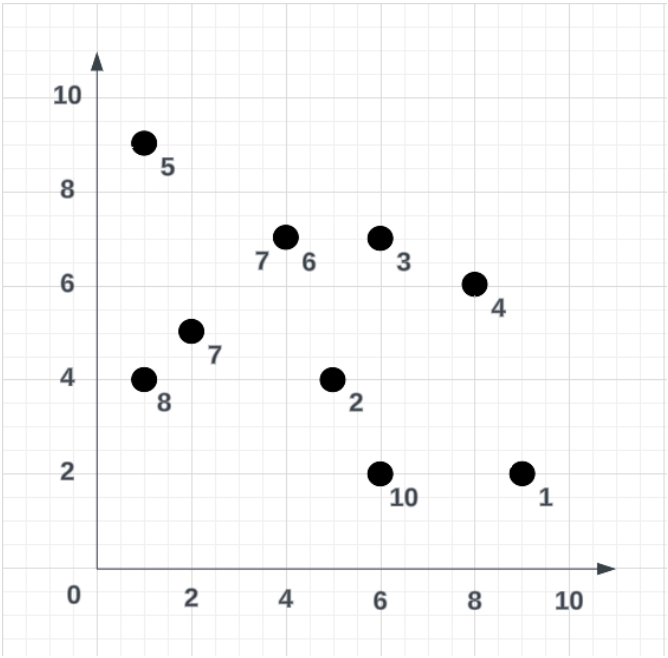
Function Name (including parameters)	Description
load_data(file_path)	Reads data from a file and returns a list of tuples that represents the points from the inputted file
euclidean_distance(x1, y1, x2, y2)	Calculates and returns the euclidean distance between two points, where the x and y coordinates of both points are the inputs
process_queries(points, query_points, radius=10.0)	Processes query points with the goal of finding how many points are within a radius of 10 units for each query point. The function returns a tuple with a list of results, the total processing time, and the average processing time per query
create_index(points)	Creates an R-tree from a list of inputted points, and returns the R-tree index, called “idx”
query_index(idx, query)	Searches for the nearest point to an inputted query point using an R-tree index and returns a tuple with the ID of the query point, the ID of the nearest point (in the inputted index), and the x and y coordinate of that point
best_first_search(idx, query_points)	Similar to query_index, best_first_search uses an R-tree to find the nearest points, but instead utilises multithreading to perform this operation using multiple query points, increasing efficiency and reducing the time taken to do so
split_data_by_median(points, coordinate=’x’)	Using a list of tuples containing the point ID, its x-coordinate and y-coordinate, it splits a list of points and returns two sublists based on whether the list of points have a lower or higher x-value than the median
best_first_search_divide_conquer(trees, query_points)	Finds the nearest points for a given set of query points and R-tree indices, and utilises the divide-and-conquer idea to search the indices provided to match the query point with the nearest point
is_dominated(p1, p2)	Checks if point p1 is dominated by point p2 based on the criteria of dominance, which is whether p2’s x-coordinate is lower or equal to p1, and if p2 has a larger or equal y-coordinate than p1
skyline_sequential(points)	Finds and returns skylines points from a list of points inputted using the Sequential Scan method
write_output(filename, skyline, query_time)	Writes the skyline points and the query processing time into a text file, with the output being a list of dictionaries where each

	dictionary represents a skyline point and contains the ID
query_points(points, query)	Finds and returns the points that fall within a specified rectangular query range by taking in a list of points, identified by its ID, x, and y-coordinates, and compares it with the query range coordinates to check whether it falls within its rectangular boundaries
main()	Organises the structure of operations that the script needs to perform to ensure that everything works properly
read_dataset()	Reads data from city2.txt and converts it into a list of tuples, where each tuple includes a home with an ID, x-coordinate and y-coordinate
construct_rtree(homes)	Constructs an R-tree index from an inputted dataset of homes, which is a list of tuples containing the home ID, the x, and the y-coordinate of the home, and returns an R-tree with the coordinates of the homes tuples
bbs_algorithm(rtree_idx)	This function takes the created R_tree and finds its skyline points using the BBS algorithm
is_skyline_point(home, skyline_bbs)	Checks to see if the home is a skyline point by comparing its coordinates to the skyline points of a given R-tree
dominates(home1, home2)	Compares the coordinates of two homes to determine whether one home dominates another
def write_output(filename, skyline_bbs, query_processing_time)	Writes the results of the entire skyline query, including the processing time and skyline points into a specified output file
divide_dataset(homes)	Splits a dataset of homes into two sublists based on the median value of their x-coordinate (home cost)
combine_skylines(skyline1, skyline2)	Combines the skyline points from two subspaces into one combined dictionary by checking the points from both input tuples that aren’t dominated

4. Analyzing BF Algorithm based NN Search

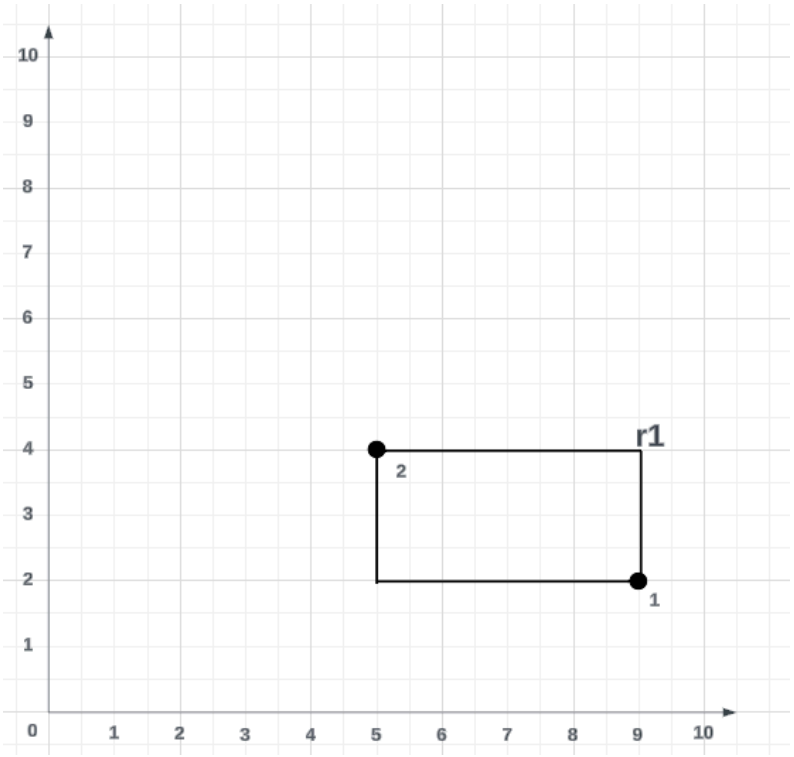
4.1 The Process of R-Tree Construction

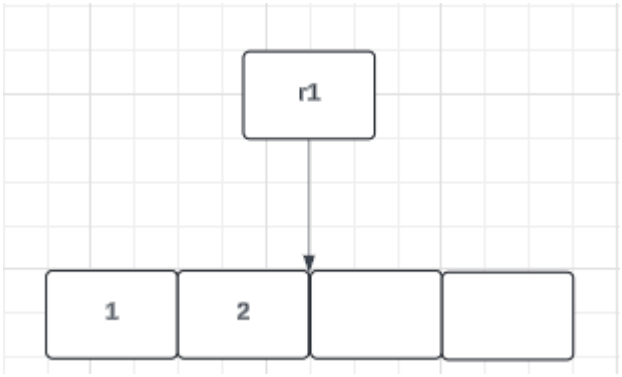
R-TREE CONSTRUCTION



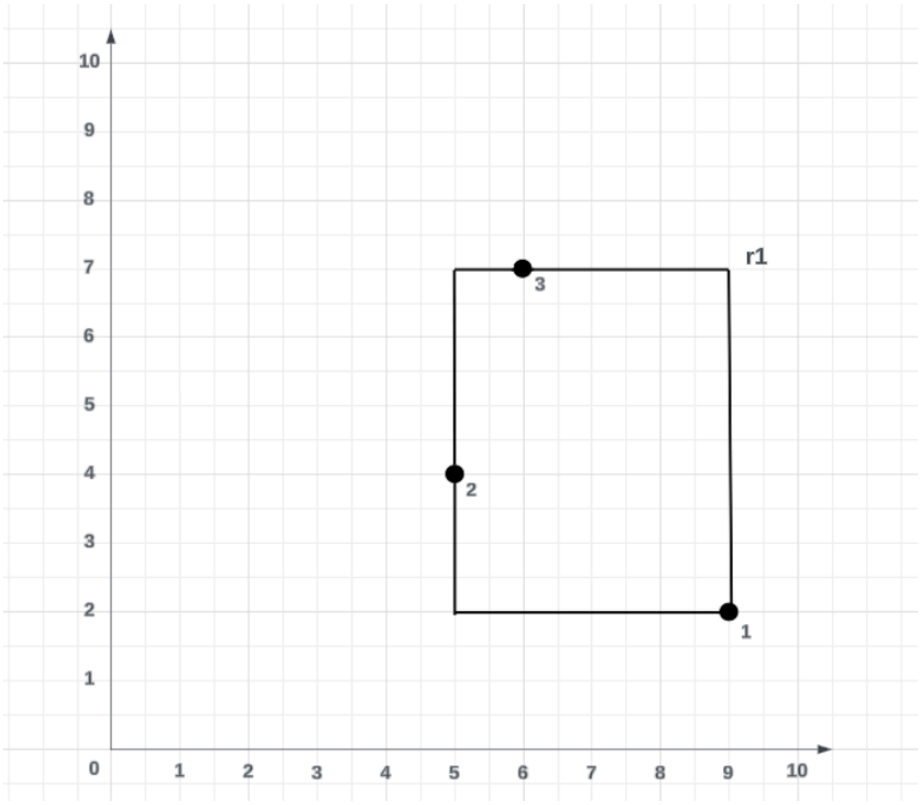
Firstly, the assumption will be made that the node capacity *B* is 4. Now the ten points will be gathered one-by-one and R-Trees will be created as each point is added.

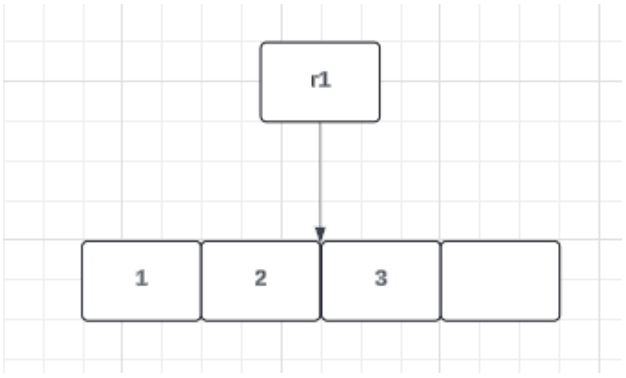
Insert 1(9, 2) and 2(5, 4)



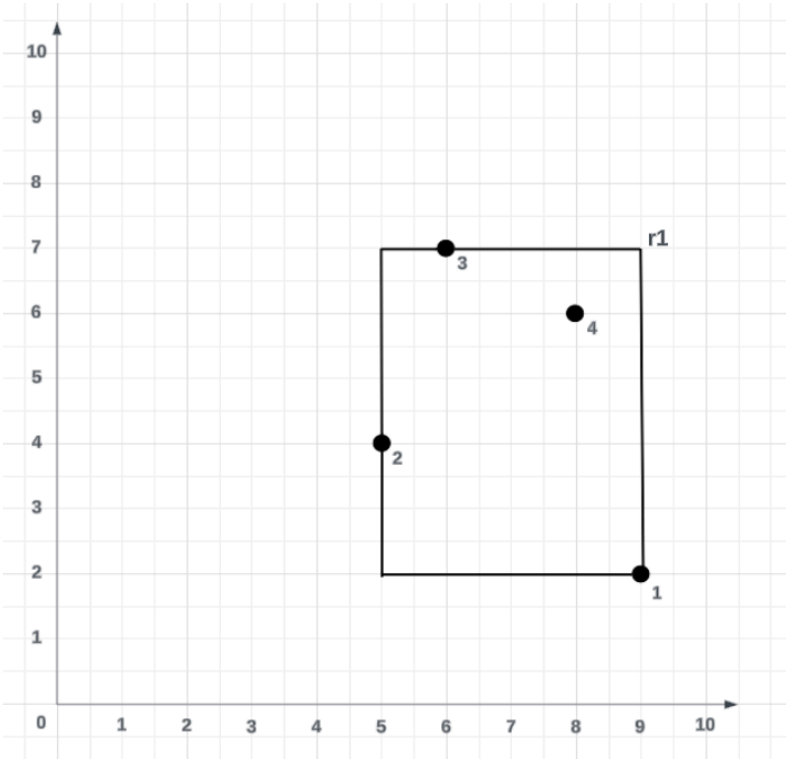


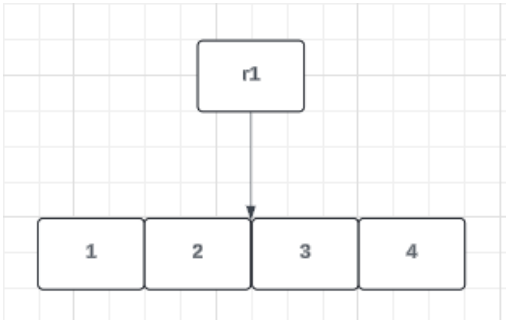
Insert 3(6, 7)



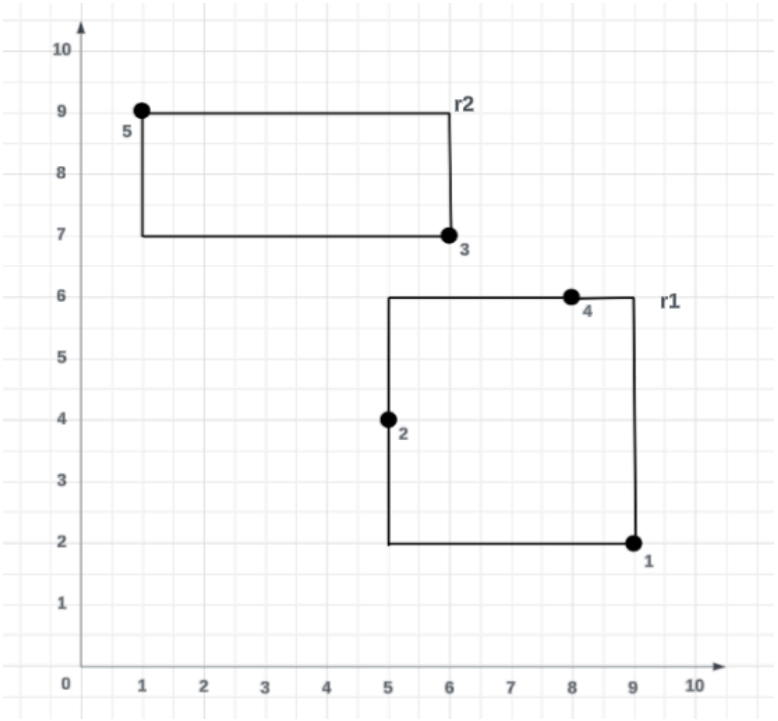


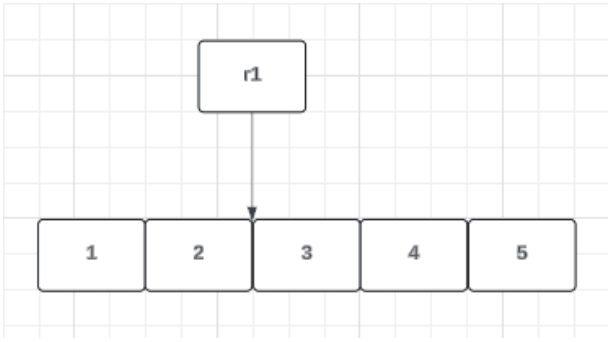
Insert 4 (8, 6)



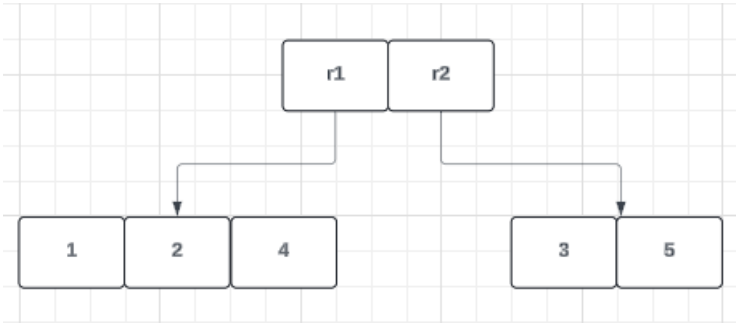


Insert 5 (1, 9) and First Split



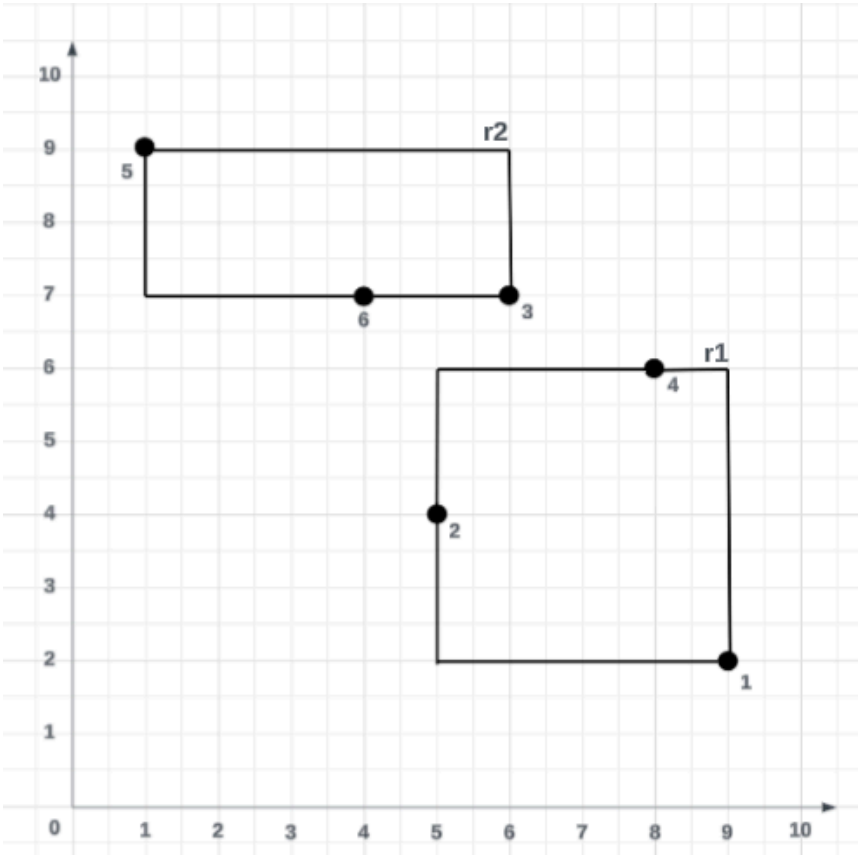


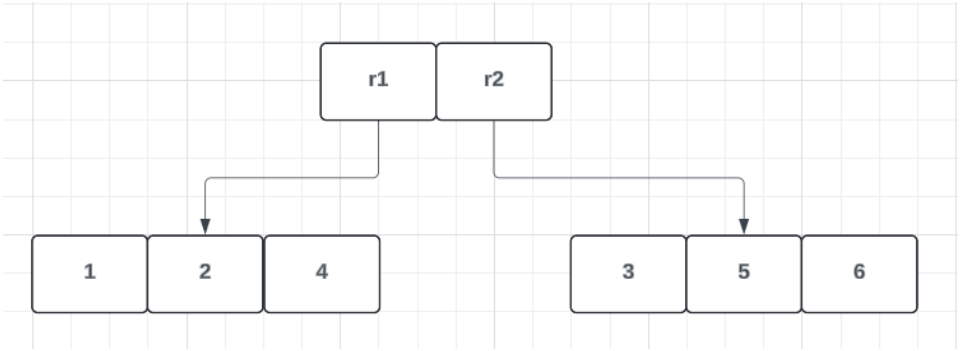
Since the maximum number of points in a node is 4, the original node must split into two to stop node overflow and to accommodate the 5th point. There also can't be 1 point in a node since the minimum amount of points in a node in this case is 2 (0.4B → 0.4(4) → 1.6 → 2)



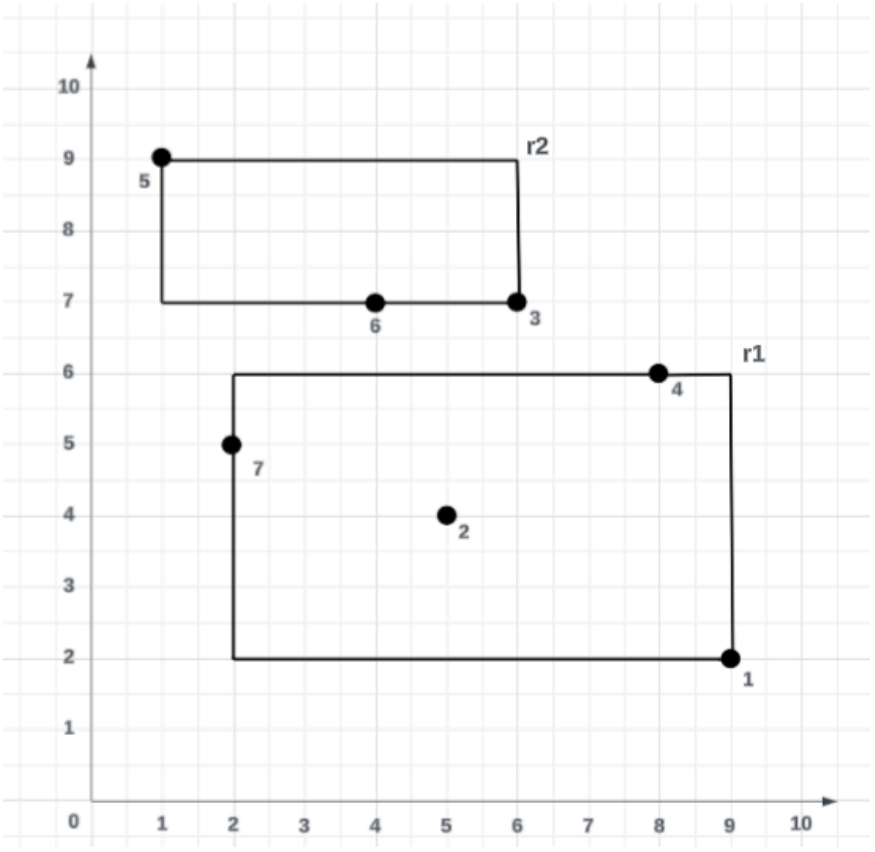
The nodes are split with the aim of minimising the area of the Minimum Bounding Rectangle (MBR).

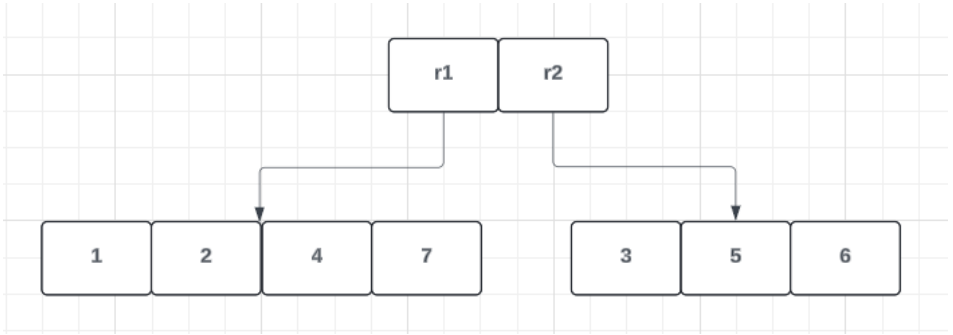
Insert 6 (4, 7)



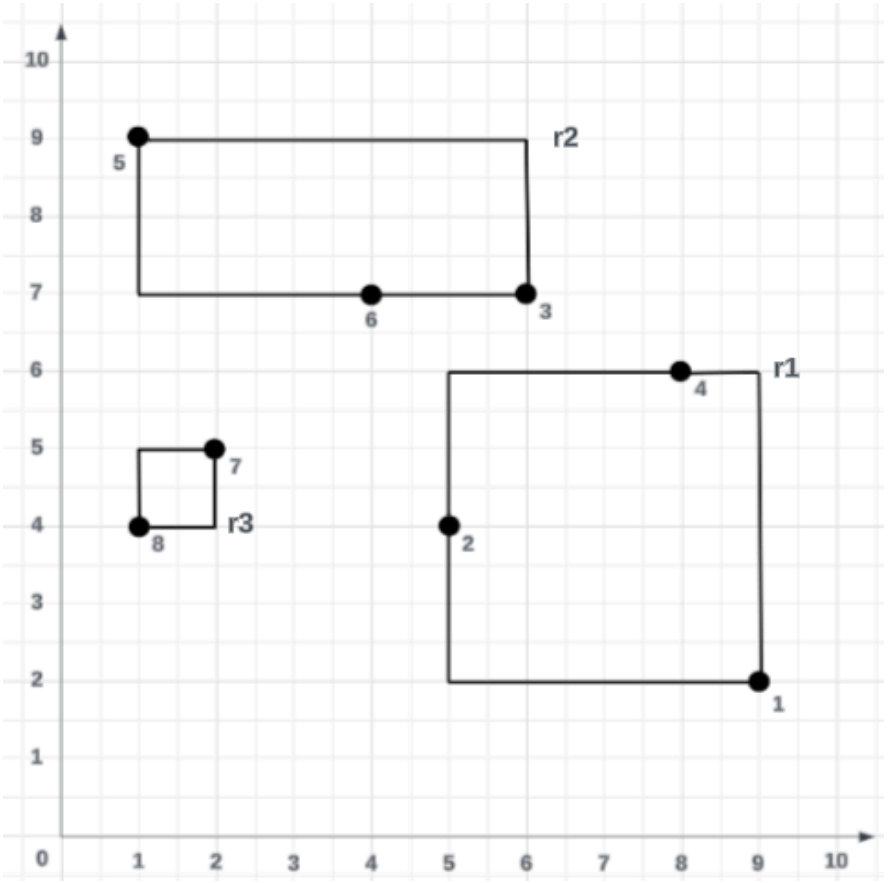


Insert 7 (2, 5)

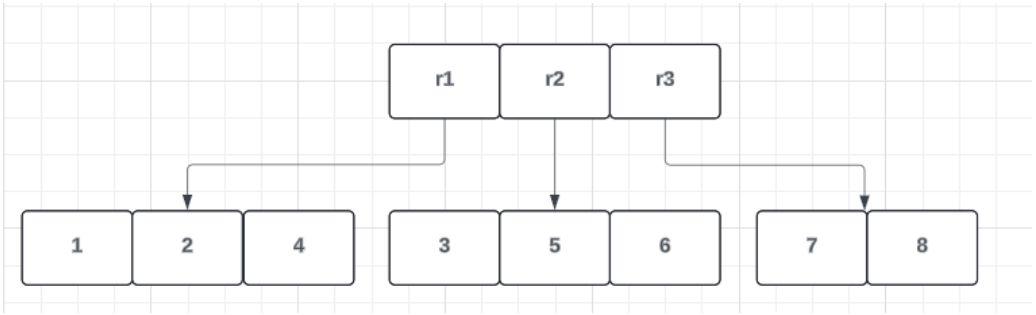




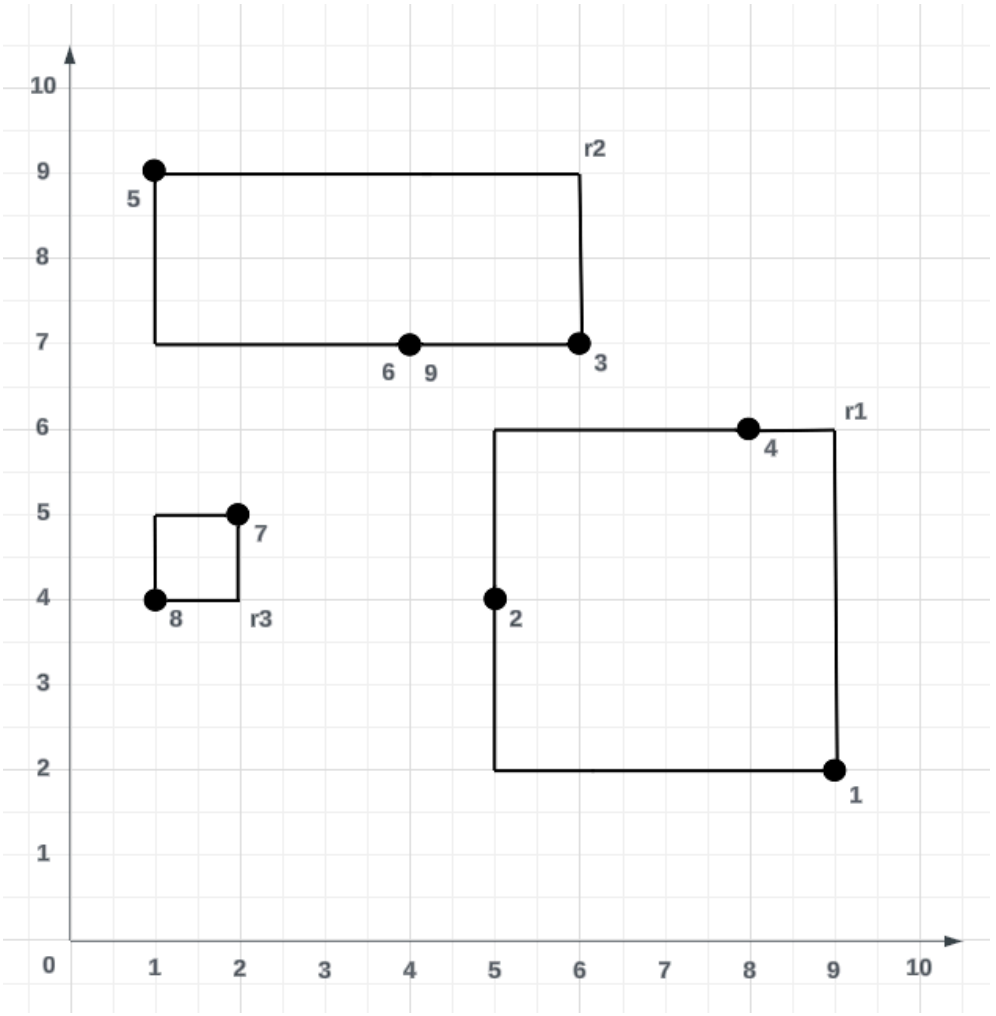
Insert 8 (1,4) and Second Split

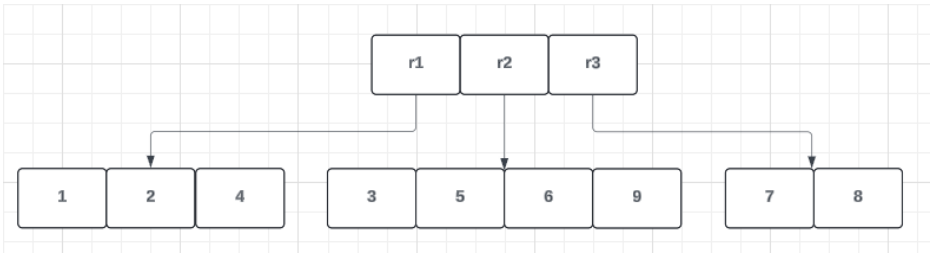


Here, the second split occurs. A new node is created around Points 7 and 8 to minimise the total area of MBRs.

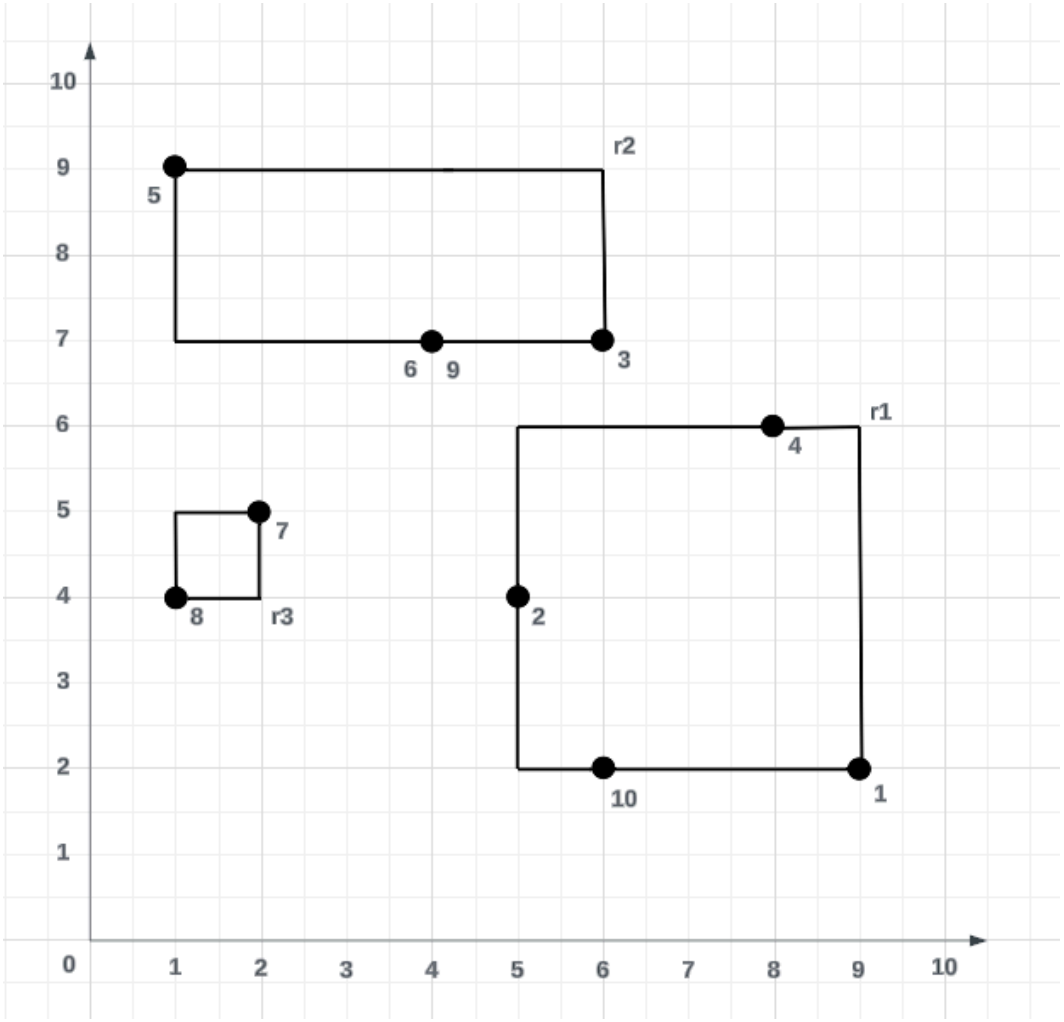


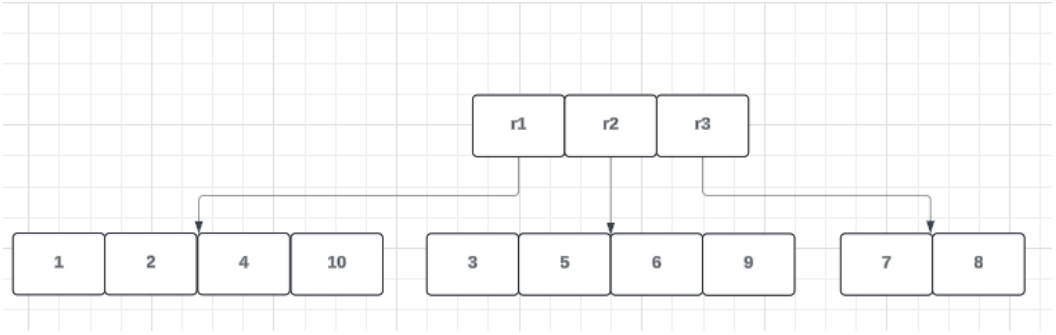
Insert 9 (4, 7)


10
9
8
7
6
5
4
3
2
1
0
1
2
3
4
5
6
7
8
9
10
r2
5
6
9
3
r1
4
7
8
r3
2
1


r1
r2
r3
1
2
4
3
5
6
9
7
8

Insert 10 (6, 2)

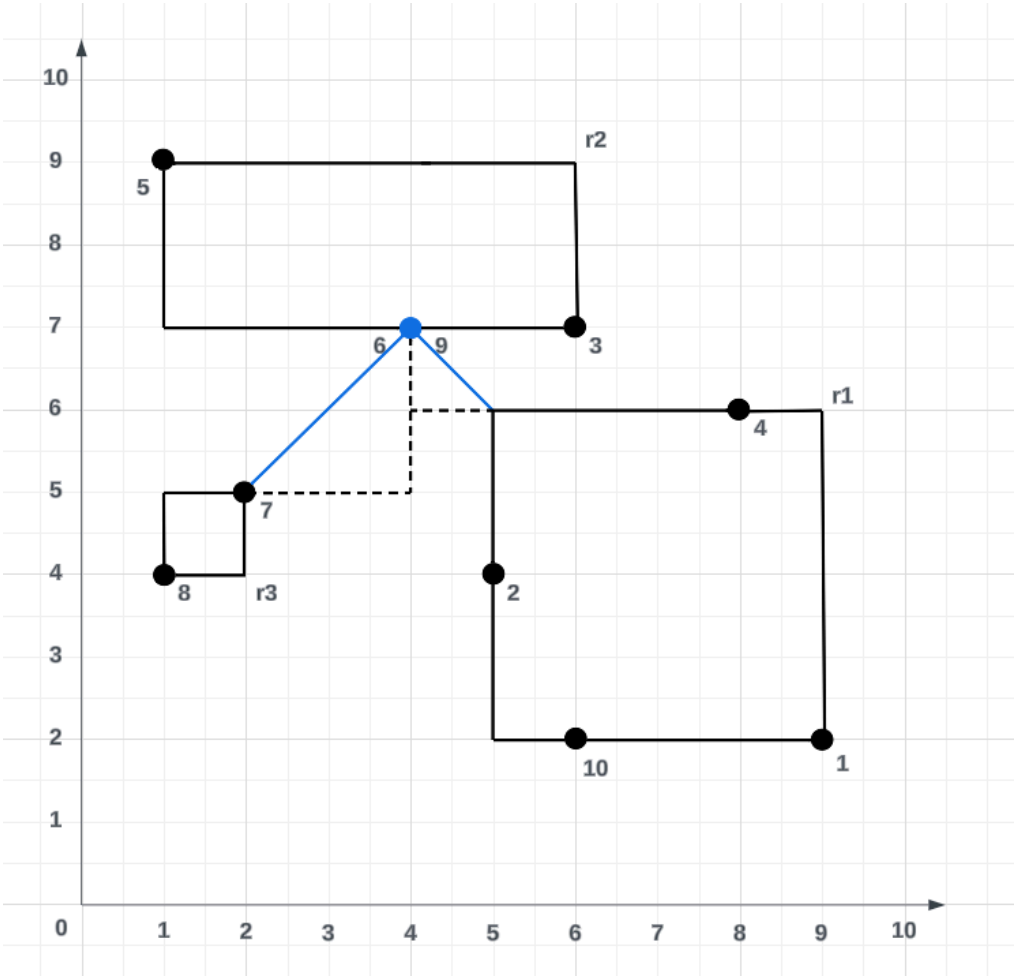




This is the final structure of the R-Tree. Setting B = 4 ensures that the area of all nodes is efficient, minimizing overlap and waste of space when creating MBRs.

4.2 The Process of BF Algorithm

The BF algorithm involves finding the minimum distance of the query point to the nearest point in the dataset, or “nearest neighbor”. This algorithm is more efficient as it keeps a sorting list to store and update the distances between the query point and the nearest point.



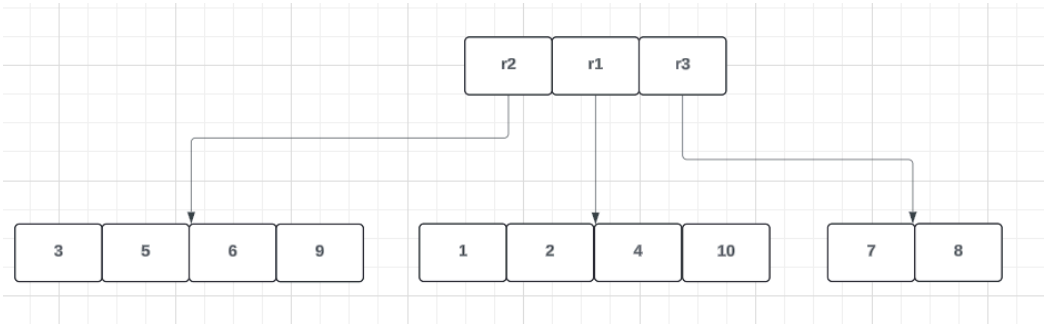
As shown above, the query point is highlighted in blue.

The first element in the sorting list is r1, which encompasses Points 3, 5, 6, 9. The distance between the query point and r1 is zero, as it conveniently sits on Point 6 and Point 9.

The second element in the sorting list is r2, which encompasses points 7 and 8. The distance between the query point and r2 is 2.828, as found using the pythagorean theorem to find the distance.

The third element in the sorting list is r3, which encompasses points 1, 2, 4 and 10. The distance between the query point and r3 is 1.414, which was also found using the pythagorean theorem.

The sorting list will look as follows:

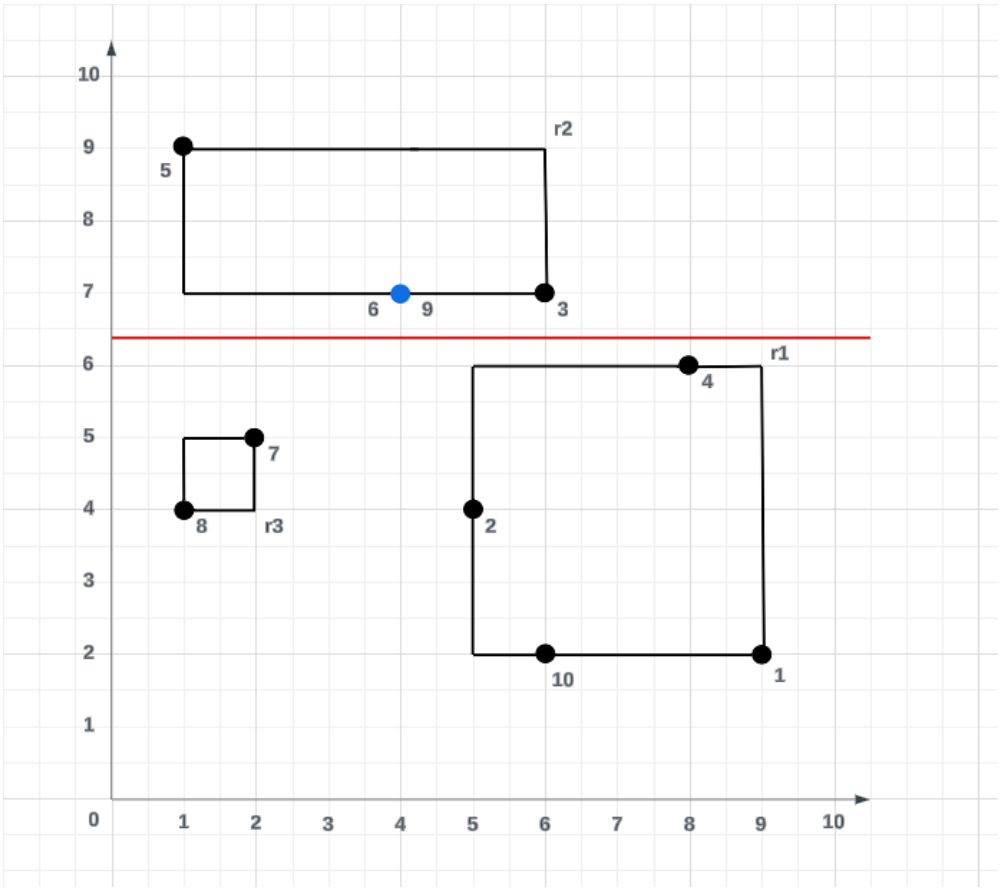


Firstly, the algorithm will go through r2, which is the closest node to the query point, and calculate the distance between the query point and all points within r2. Once the algorithm comes across Point 6, it will calculate the distance to be 0. Since 0 is the smallest distance possible between the query point and an individual point within a node, the algorithm will terminate early to save computational power.

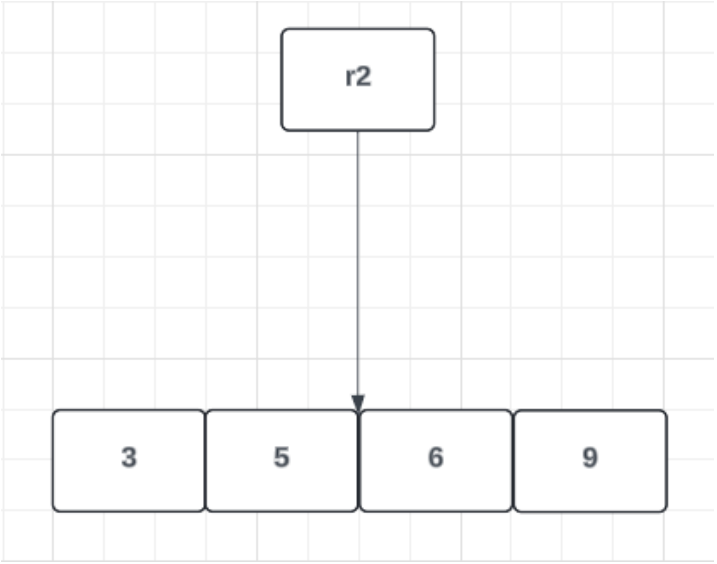
4.3 The Process of Divide-and-Conquer

The divide-and-conquer idea will be applied to the BF algorithm by partitioning the dataset and analysing each segment to find the query result. To save computational power, the BF algorithm will terminate early if the query result has been found in the first segment it searches.

Searching the Top Segment



The Divide-and-Conquer implementation will begin in the top segment. The only element in this segment is r2, which the BF algorithm will add to its sorting list.

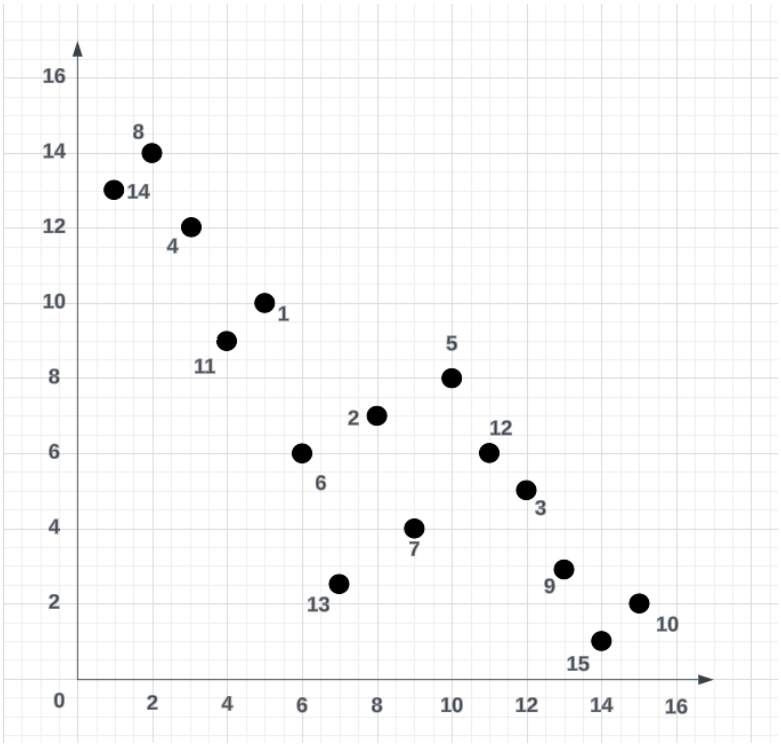


With r1 being the only node in the top segment, the BF algorithm will visit the node and find that the first nearest point to the query point is point 6, with a distance of zero. Having found the query result, the BF algorithm will terminate to save computational resources.

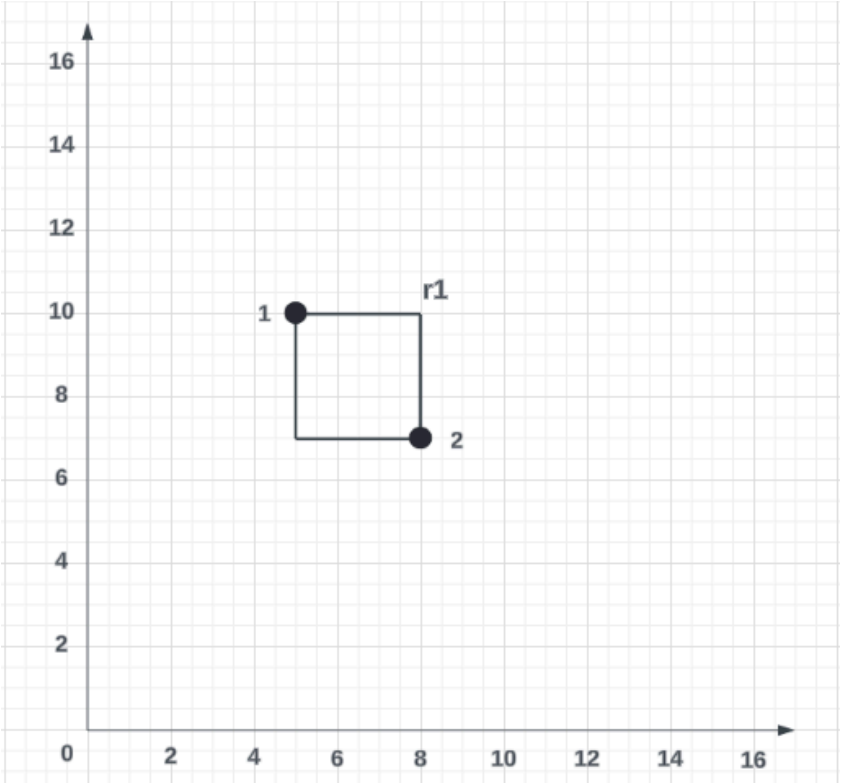
5. Analyzing the BBS Algorithm based Skyline Search

5.1 The Process of R-Tree Construction

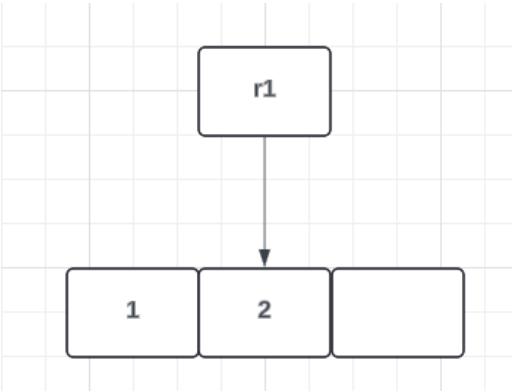
Firstly, gather all of the 15 points and insert each point into the R-Tree one by one. In this scenario, when one node exceeds its maximum capacity of points (B = 3), it will split into two nodes.



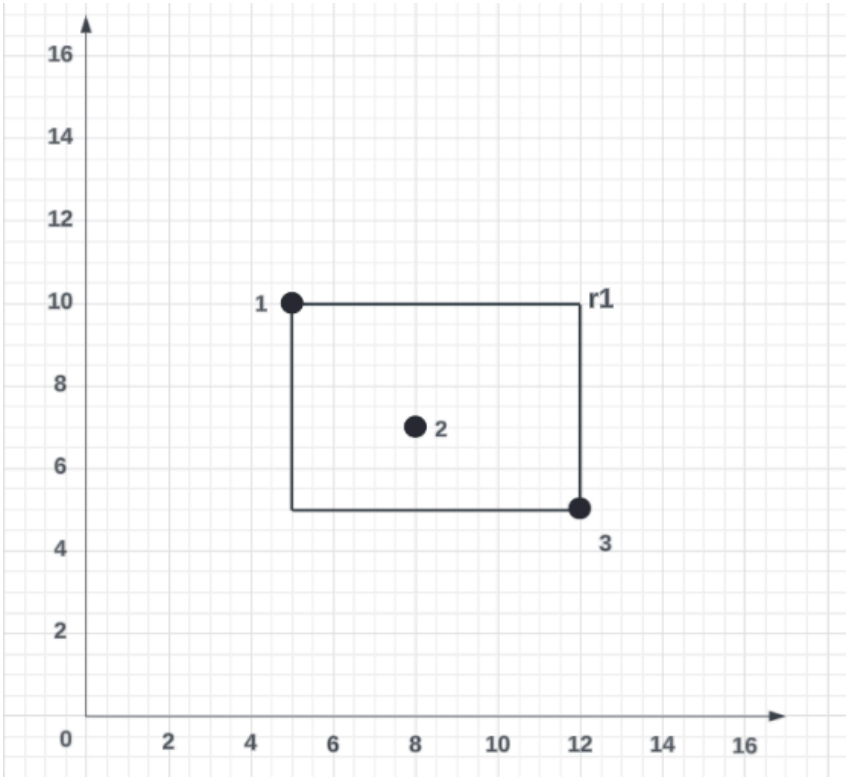
Insert 1(5,10) and 2(8, 7)

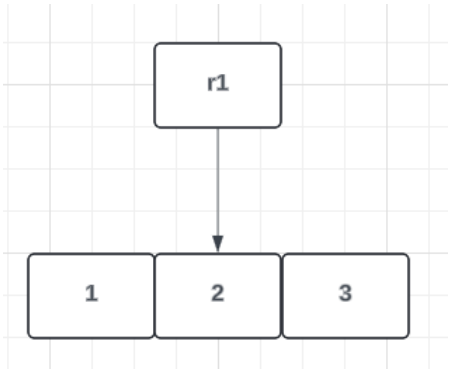


The first two points are added, making up the first Minimum Bounding Rectangle, r1.

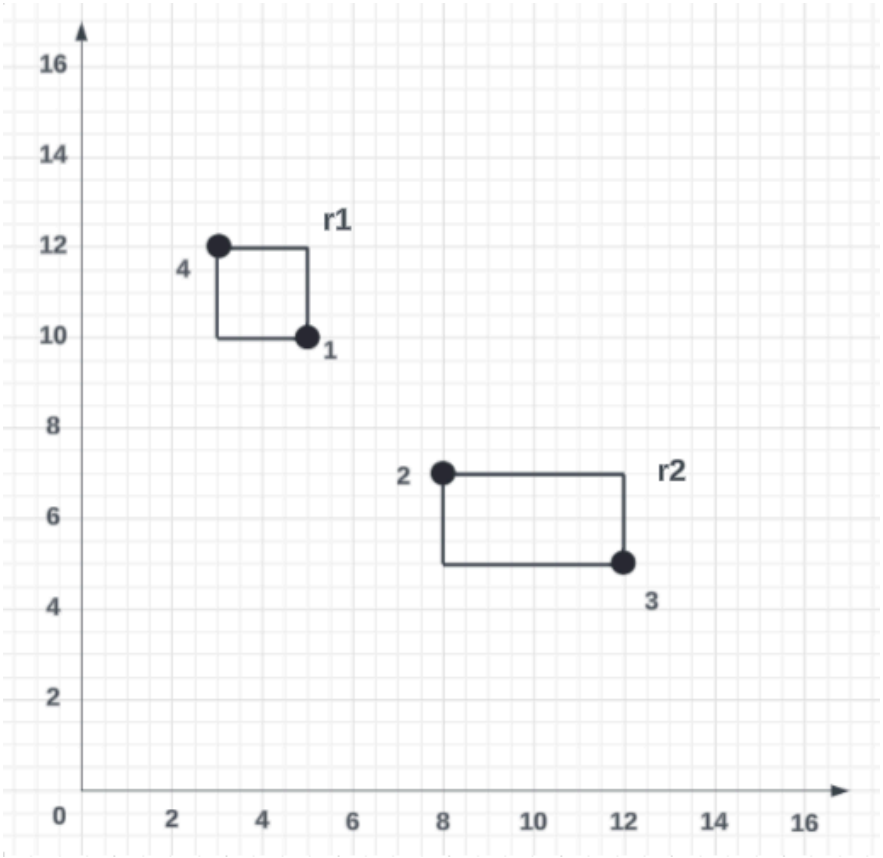


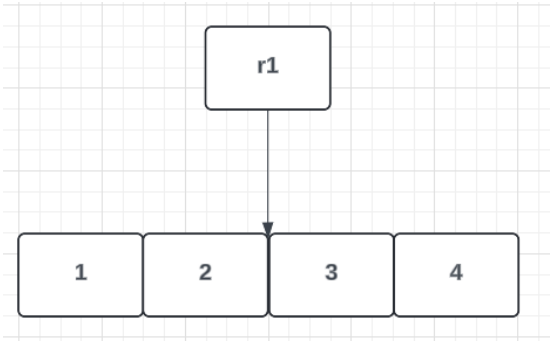
Insert 3 (12, 5)





Insert 4 (3, 12) + Splitting

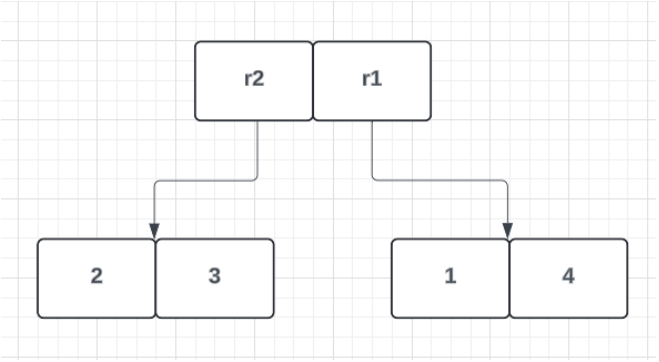




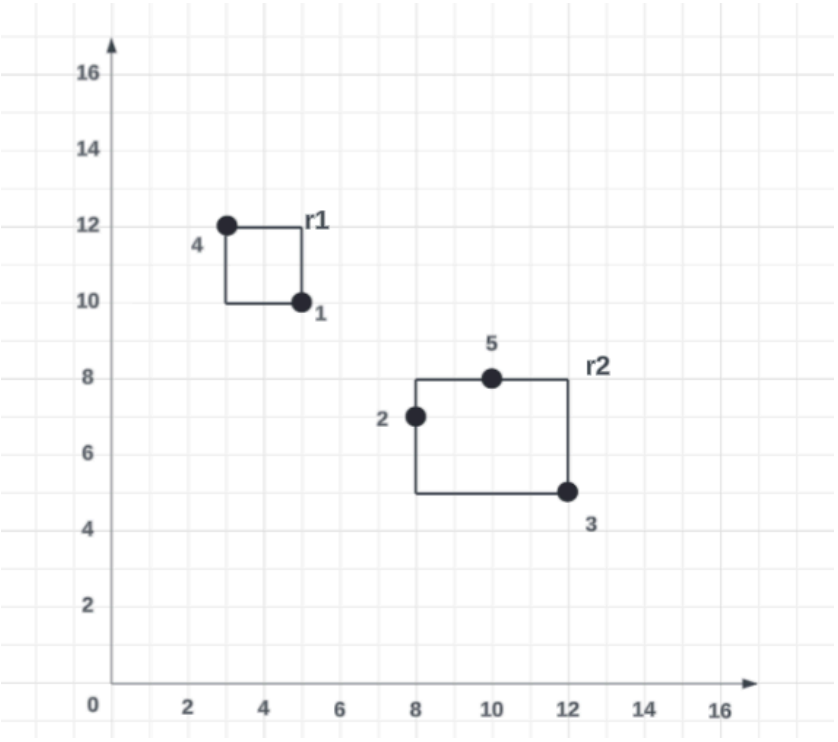
Since we have set B = 3, having more than 3 points leads to an overflow, thus needing a split.

the original node must split into two to accommodate the 5th point. There also can't be 1 point in a node since the minimum amount of points in a node in this case is 2 (0.4B → 0.4(3) → 1.2 → 2)

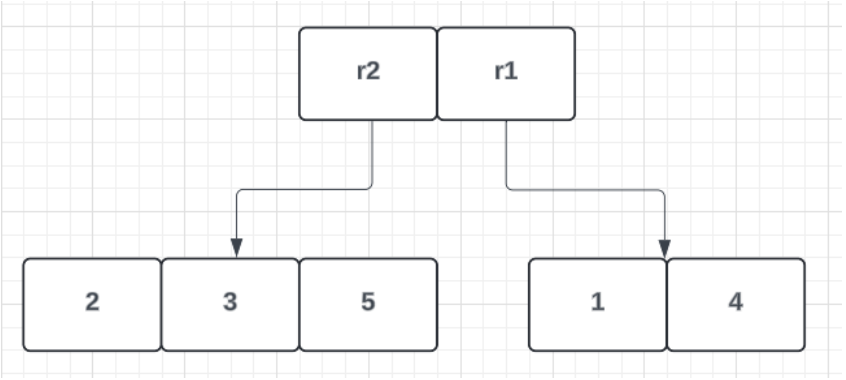
The node is split into two smaller nodes with the aim of minimising the areas of the MBRs.



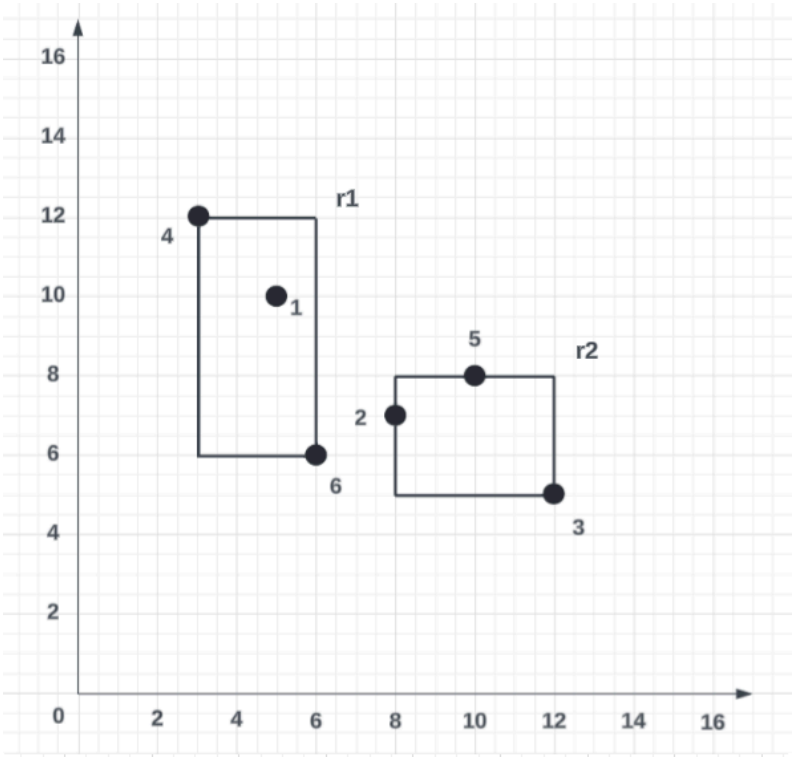
Insert 5 (10, 8)

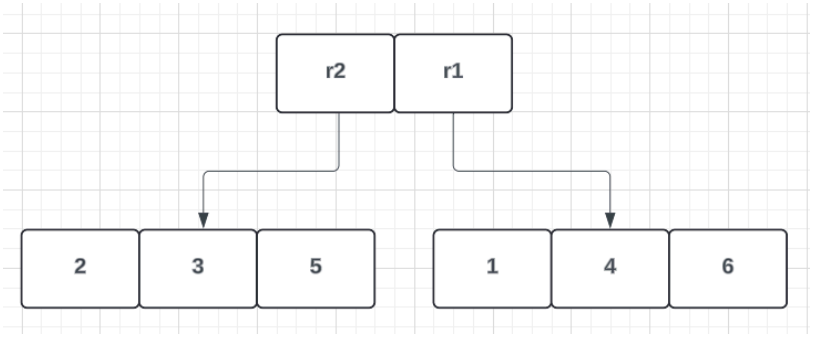


Adding point 5 to the lower MBR will reduce the total sum of areas for both MBRs

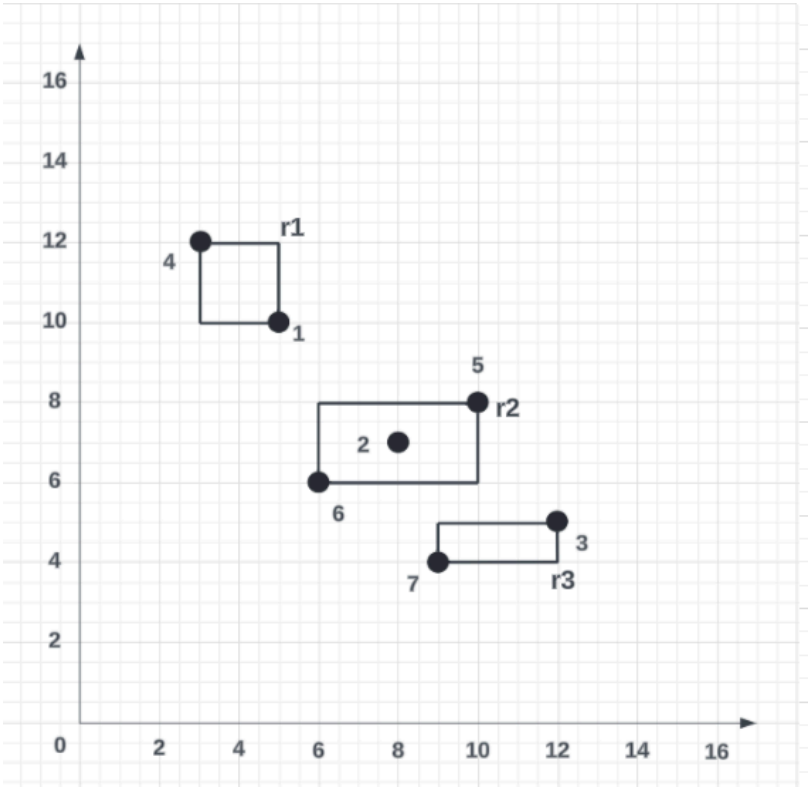


Insert 6 (6,6)

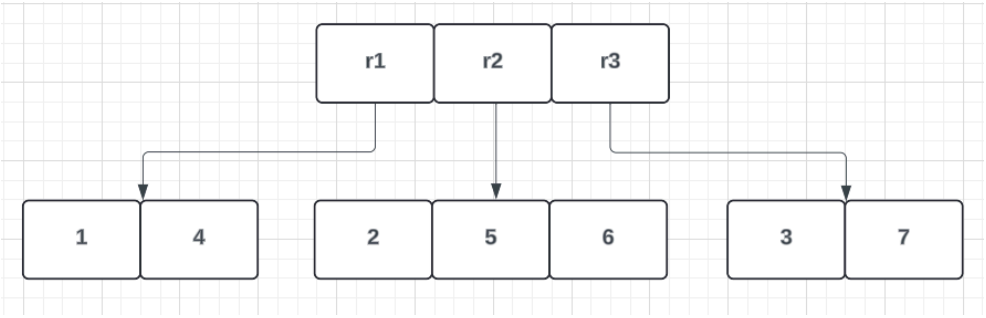




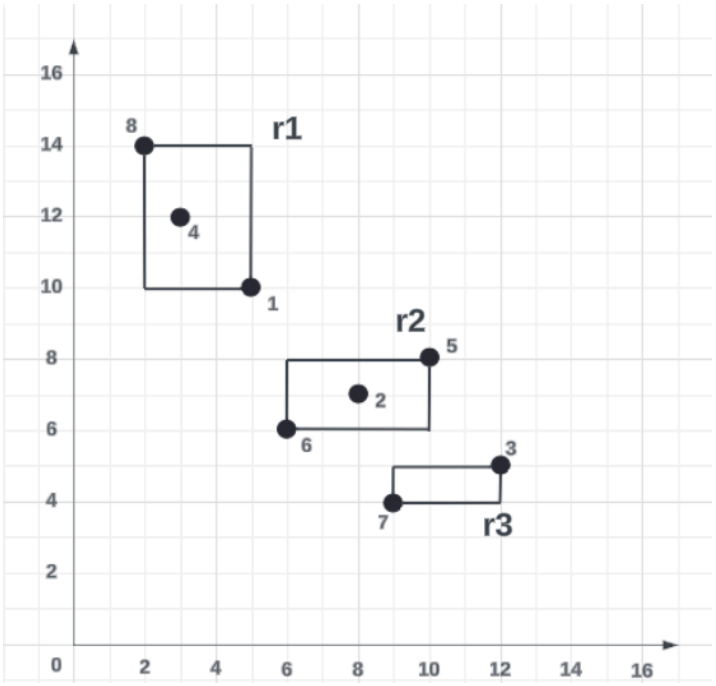
Insert 7 (9,4) + Splitting

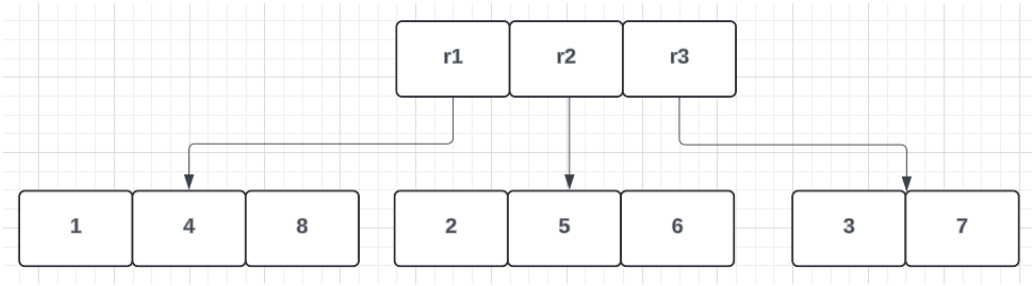


Inserting point 7 causes another split, with new MBRs being drawn up, and allowing smaller areas for each rectangle and overall area.

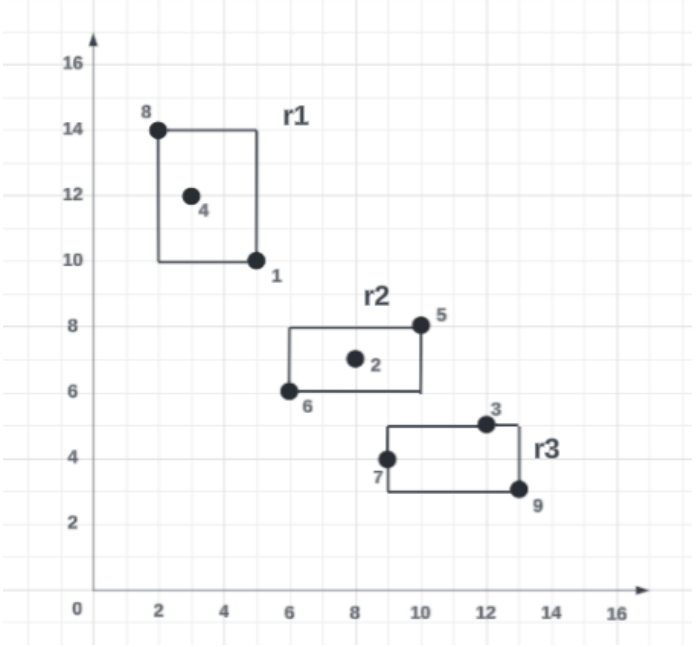


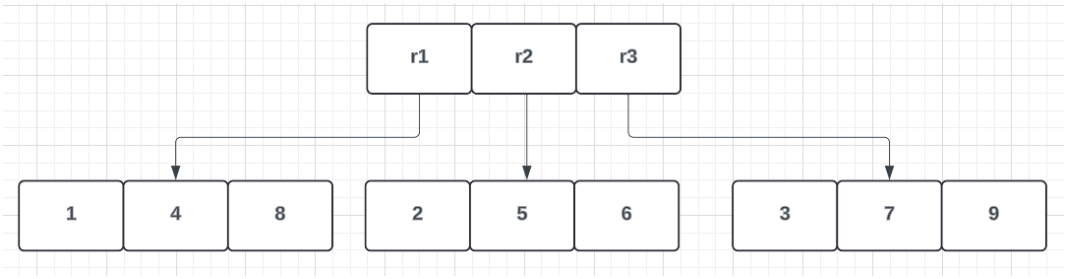
Insert 8 (2,14)



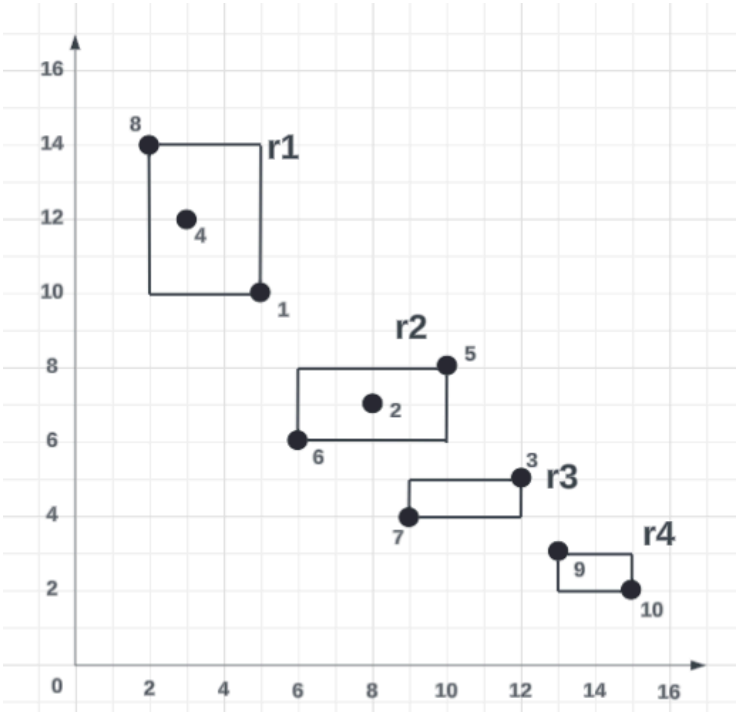


Insert 9 (13,3)

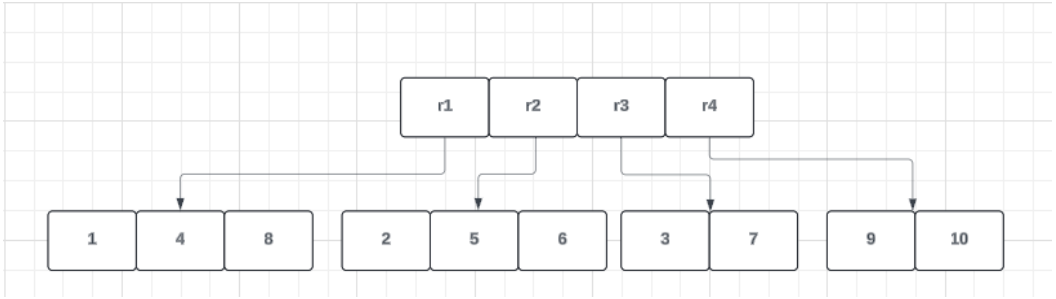




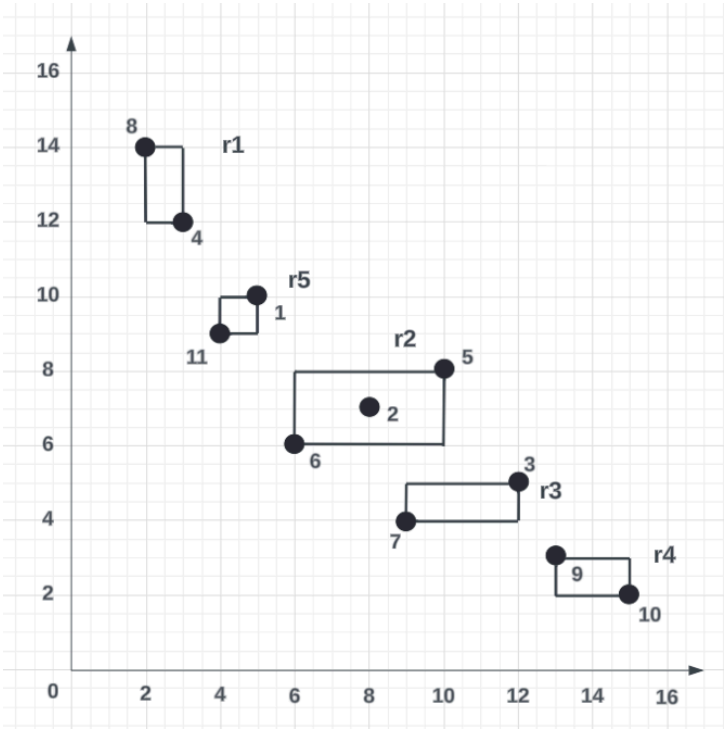
Insert 10 (15,2) + Splitting



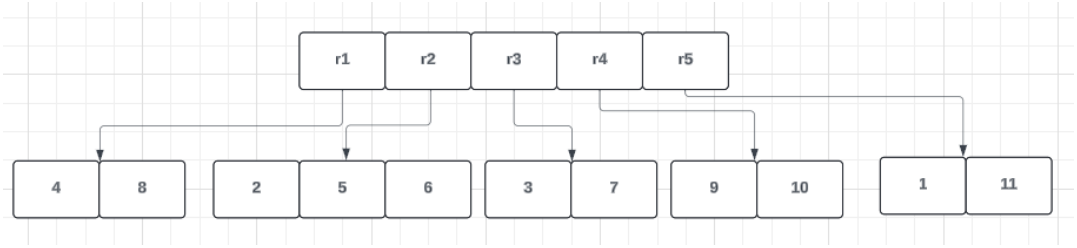
The insertion of point 10 causes another split of nodes due to overflow, where splitting r3 into two nodes will help minimize the perimeter area of the two new MBRs



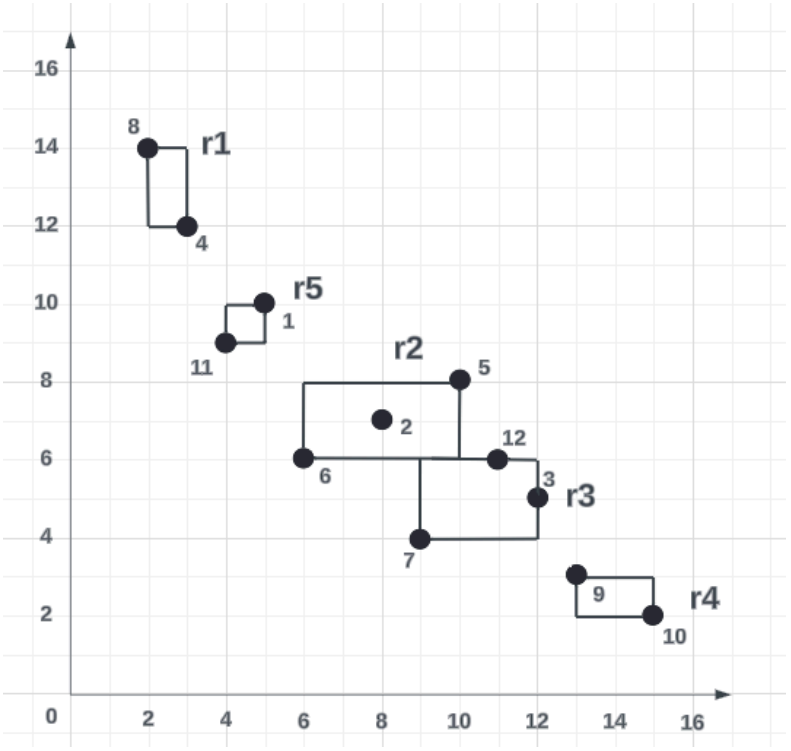
Insert 11 (4, 9) + Splitting

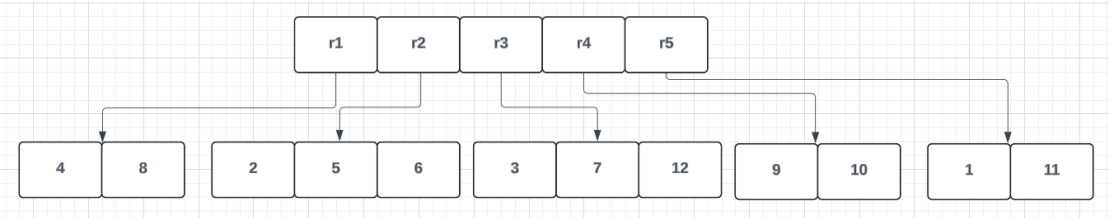


Since the insertion of point 11 into the top node will cause an overflow, splitting must occur, with r1 being split into two nodes.

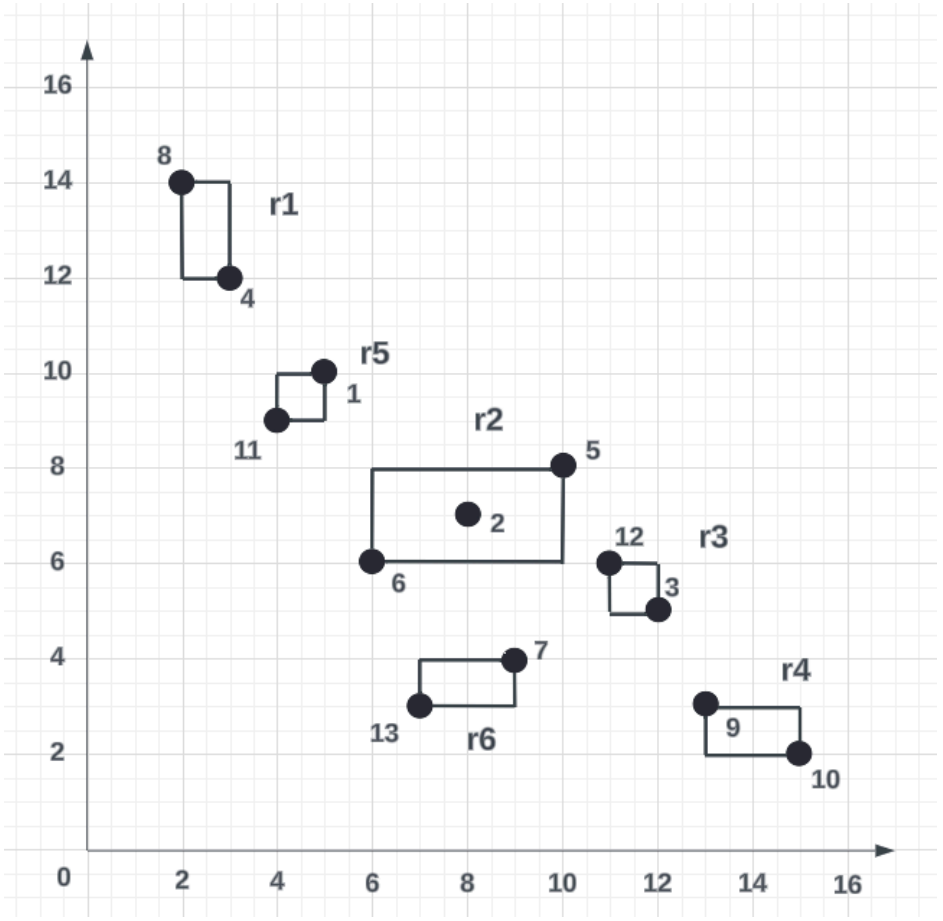


Insert 12 (11, 6)

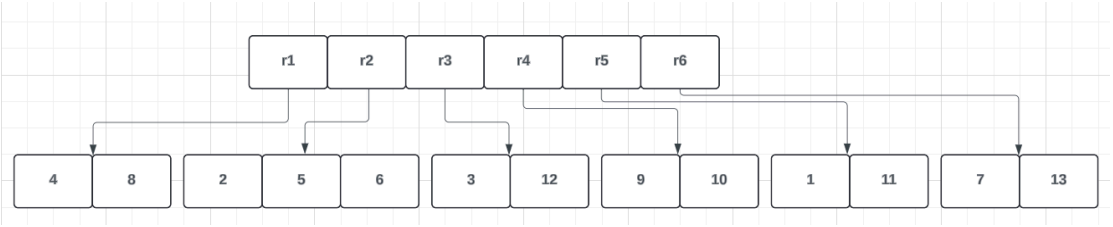




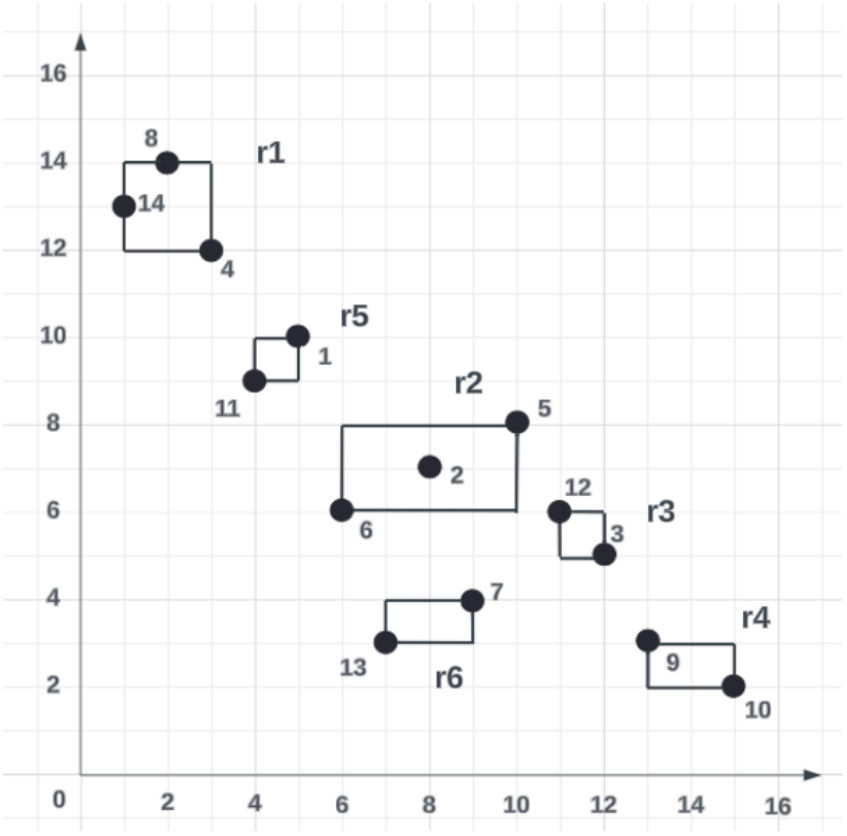
Insert 13 (7, 3) + Splitting

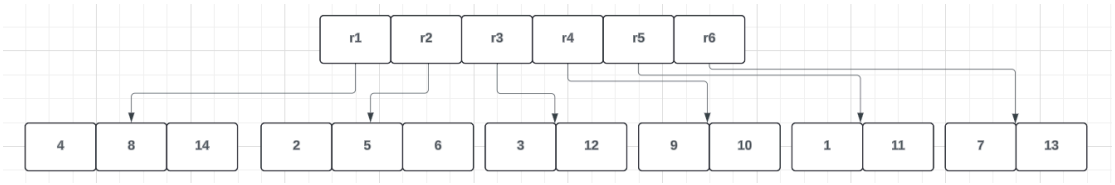


The insertion of point 13 and the split of node r3 allows the MBRs of the new points to be smaller due to its smaller distances

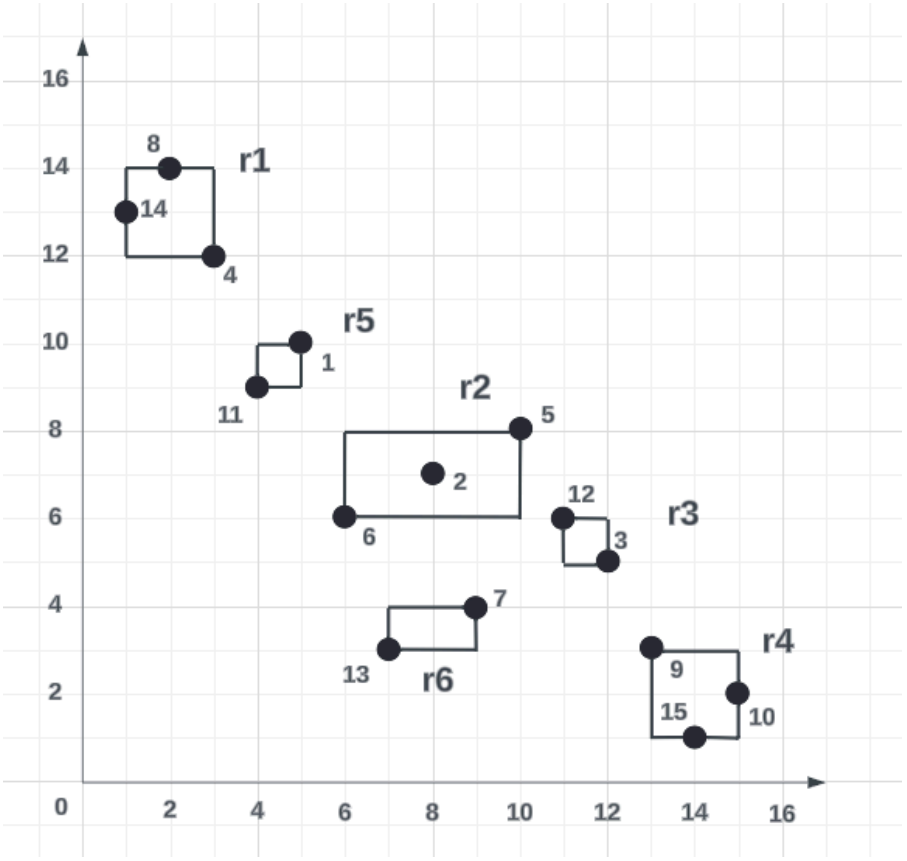


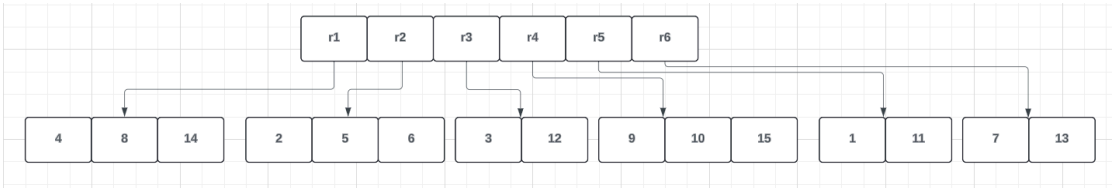
Insert 14 (1, 13)





Insert 15 (14, 1)



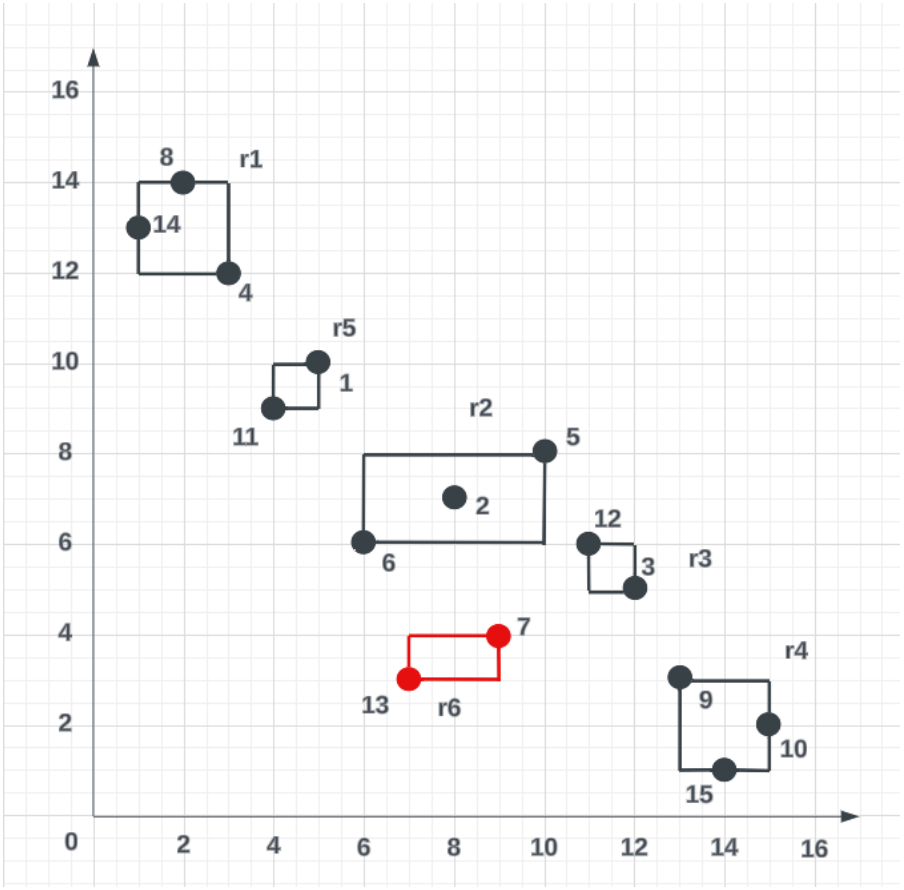


5.2 The Process of BBS Algorithm

Executing the BBS Algorithm using the R-tree structure above to identify the skyline points is achieved by checking all of the nodes for points that aren't dominated by other points in the dataset. If all points in a node are dominated, they are not considered skyline points, and the node will be pruned. One point is considered to dominate another point if it is at least as good in all dimensions and at least better in one dimension.

The first node that will be searched is the node that is closest to the origin.

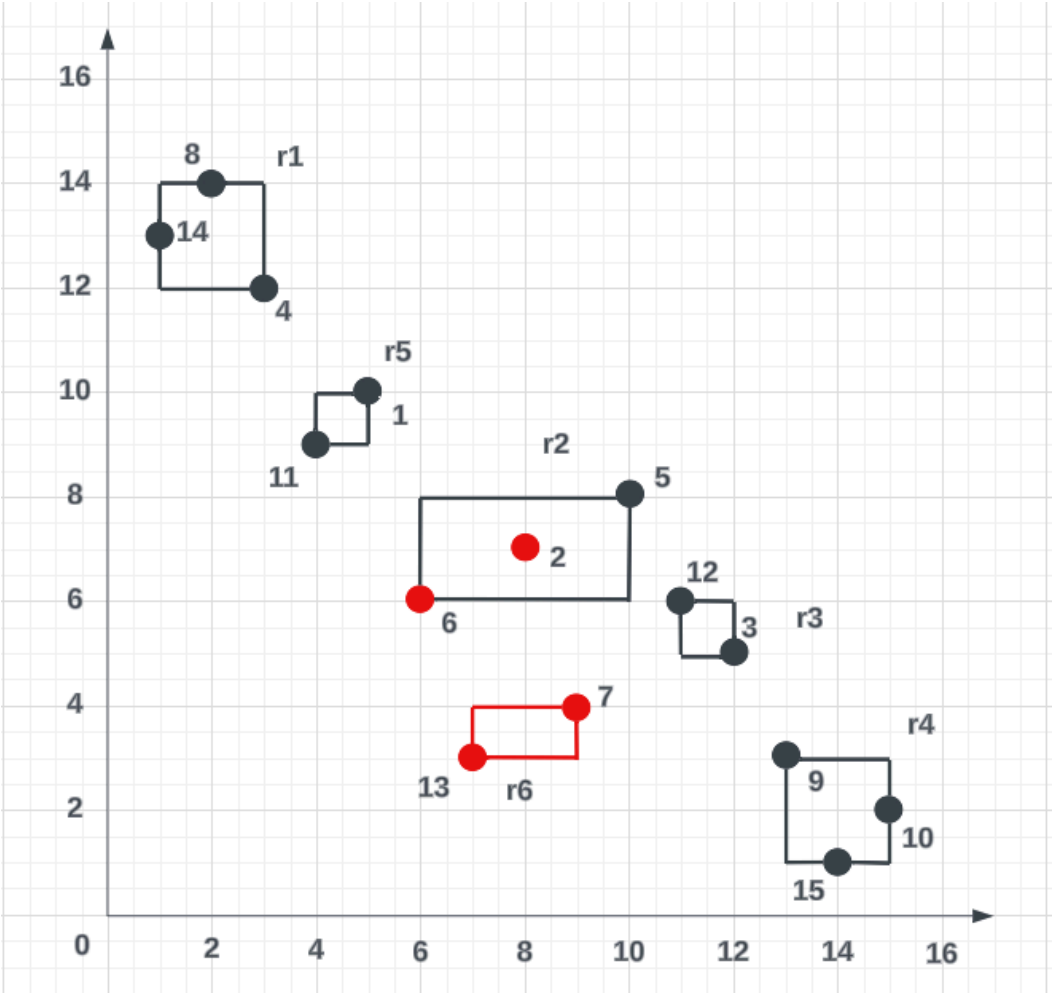
Check Node r6



Since Point 7 and 13 are dominated by various other points in the dataset, Node r6 is pruned.

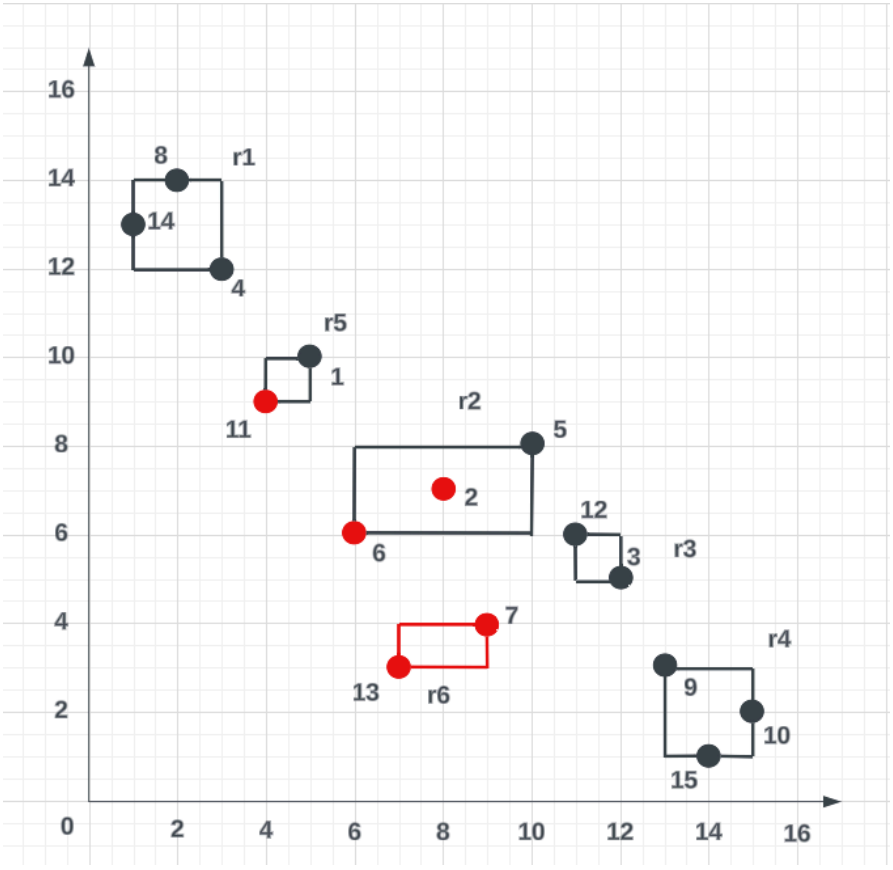
Check Node r2

This node will be checked as it is the next closest to the origin.



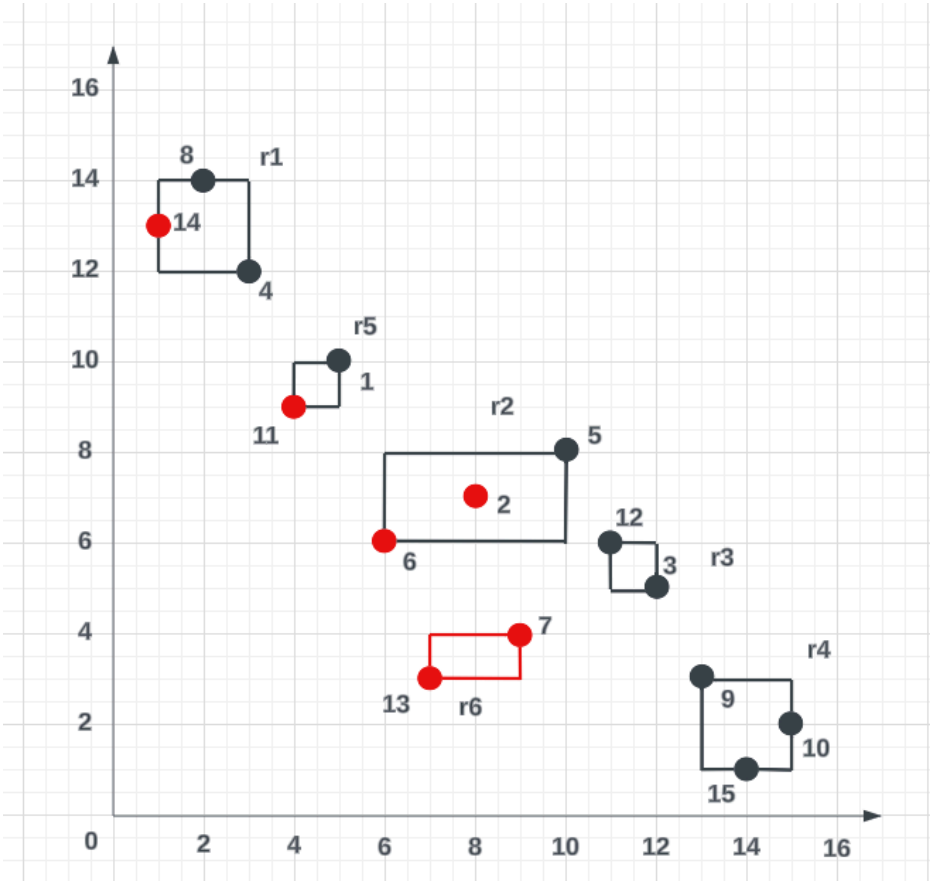
Here, Points 2 and 6 are considered dominated due to their inferior qualities compared to Point 5. Point 5 can't be ruled as dominated as there isn't a single point in the dataset that isn't the same in all dimensions and at least better in one dimension, therefore meaning that Node r2 remains.

Check Node r5



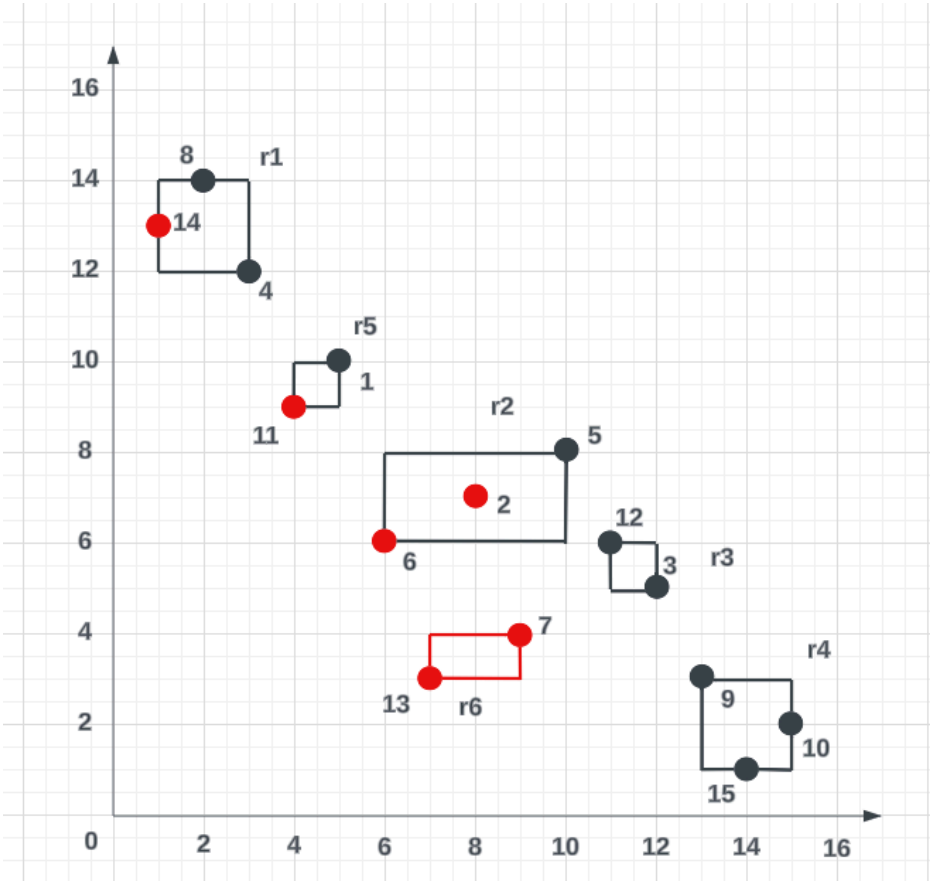
Here, Point 11 is dominated by Point 1 as it is greater in both dimensions than Point 11. Since all points aren't dominated, Node r5 will not be pruned.

Check Node r1



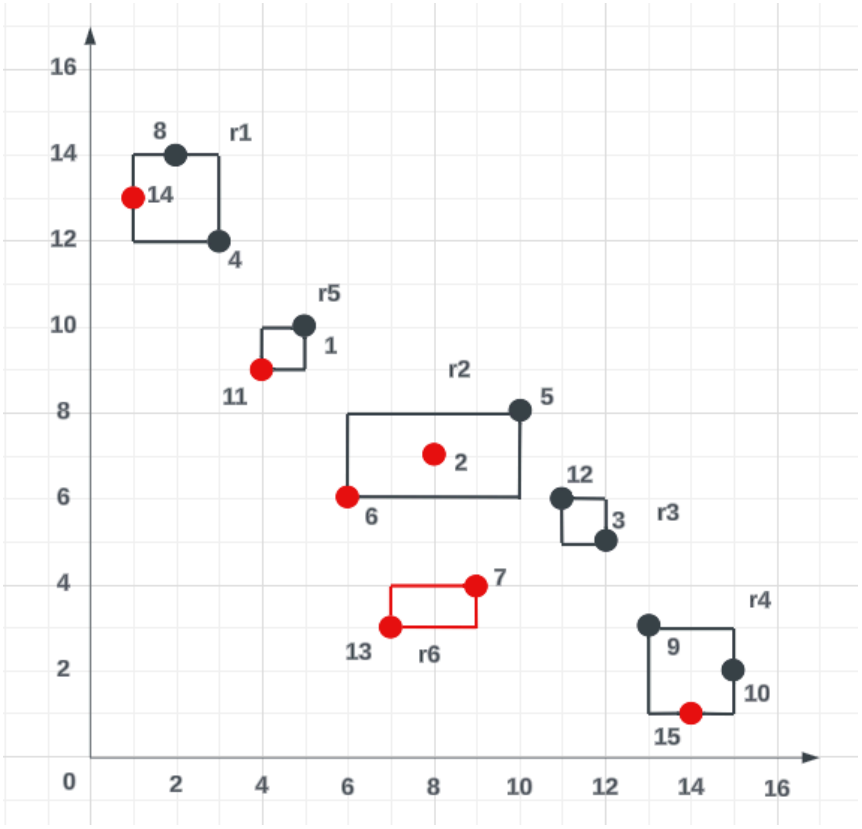
In this node, only Point 14 is considered dominated as both its x and y dimensions are smaller than Point 8. Since all of the points aren't dominated, this node won't be pruned.

Check Node r3



Here, both Point 3 and Point 12 aren't dominated as no point meets the dominating criteria, and as a result, Node r3 will not be pruned.

Check Node r4



Only Point 15 in this node is dominated (by Point 10), but since all of the points aren't dominated, the node won't be pruned.

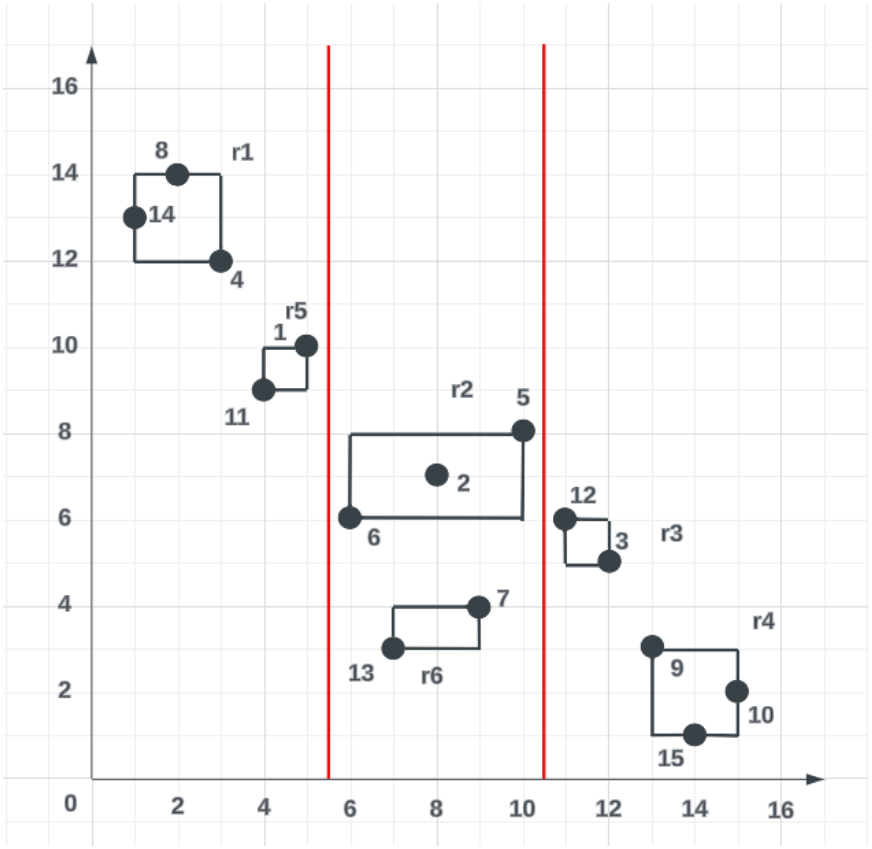
Overall, Points 1, 3, 4, 5, 8, 9, 10 and 12 are chosen as the skyline points as they aren't dominated, nor do they dominate each other.

5.3 The Process of Divide-and-Conquer

The divide-and-conquer strategy involves breaking down a problem into smaller components, solving them independently, and combining the solutions to form the final result. This idea can be applied to the BBS algorithm by dividing the dataset into segments, and then utilising nodes to find local skyline points, with the results being merged to compare them and ultimately find the global skyline points.

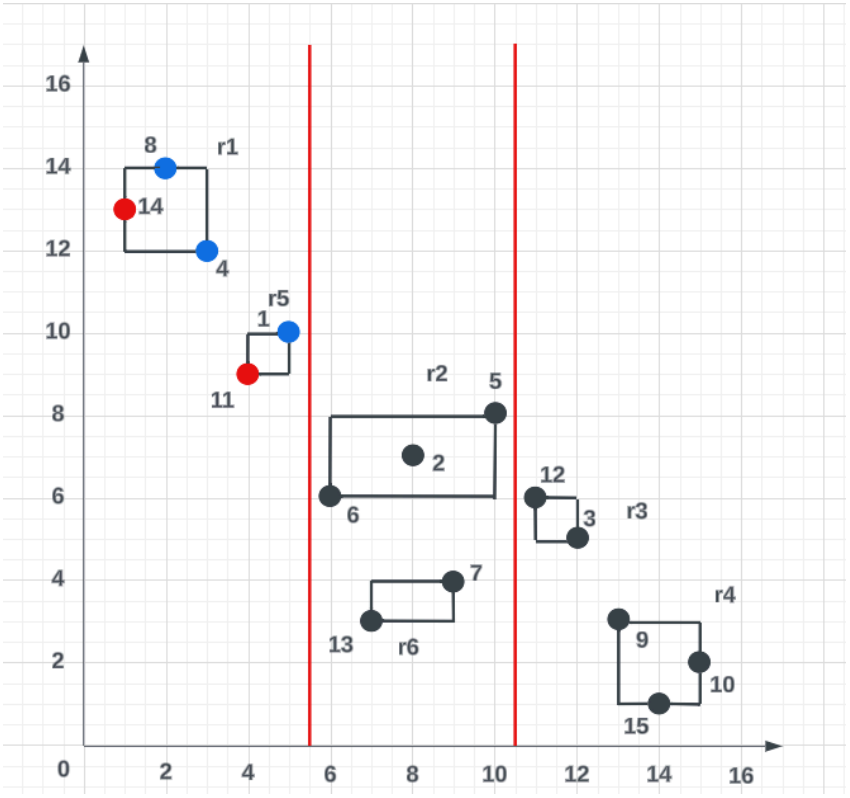
Divide Dataset into Three Sections

The dataset will first be divided into three segments to find the local skyline points within the established nodes:



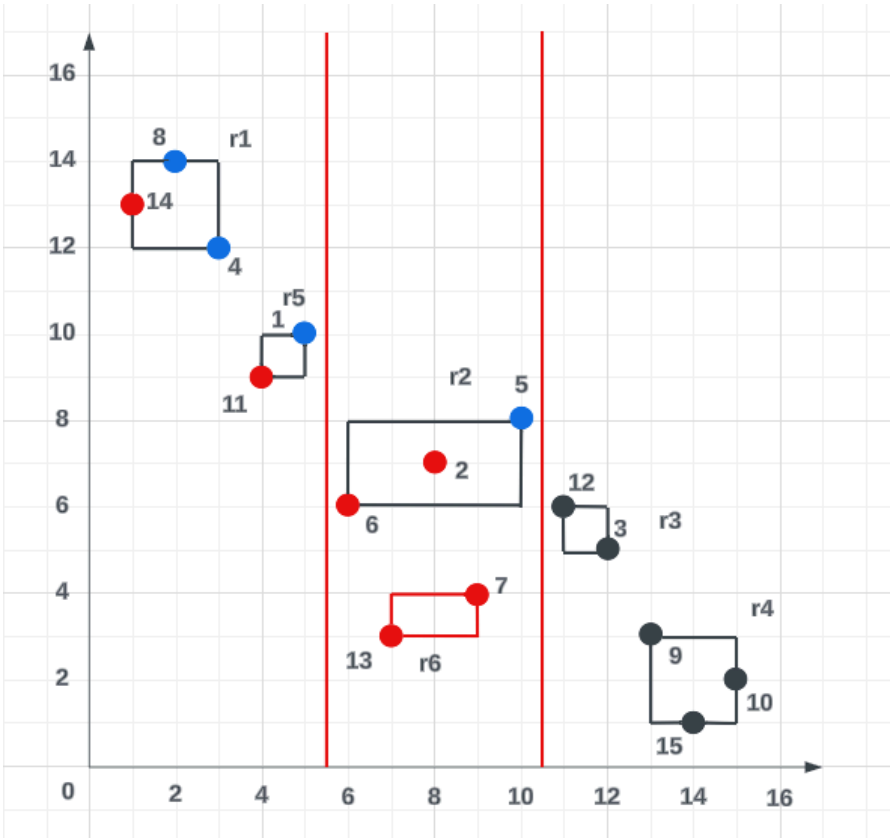
Afterwards, each segment will be individually analysed to find local skyline points.

Left Side



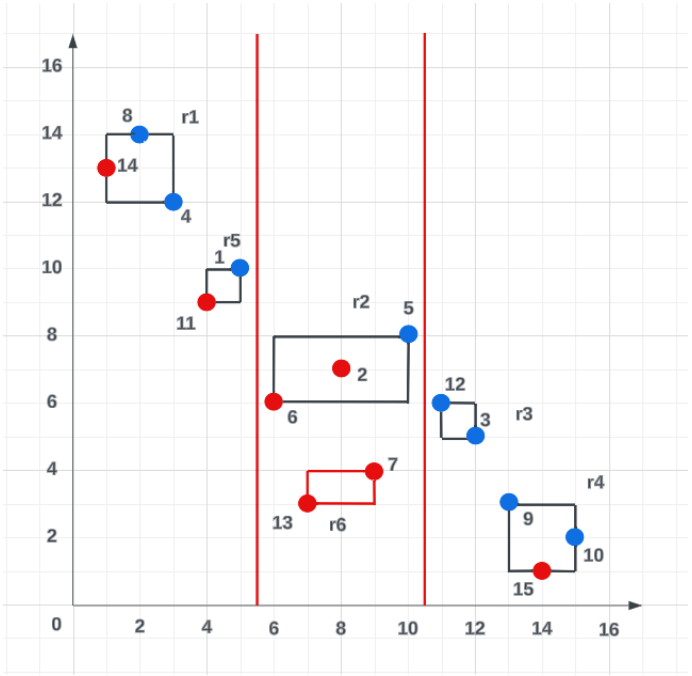
In the left segment, the BBS algorithm will use the established R-trees to find local skyline points. The points in blue indicate the local skyline points, and the red points show the points that have been dominated. In this case, no node will be pruned as there isn't a node in which all of its points have been dominated.

Middle Side



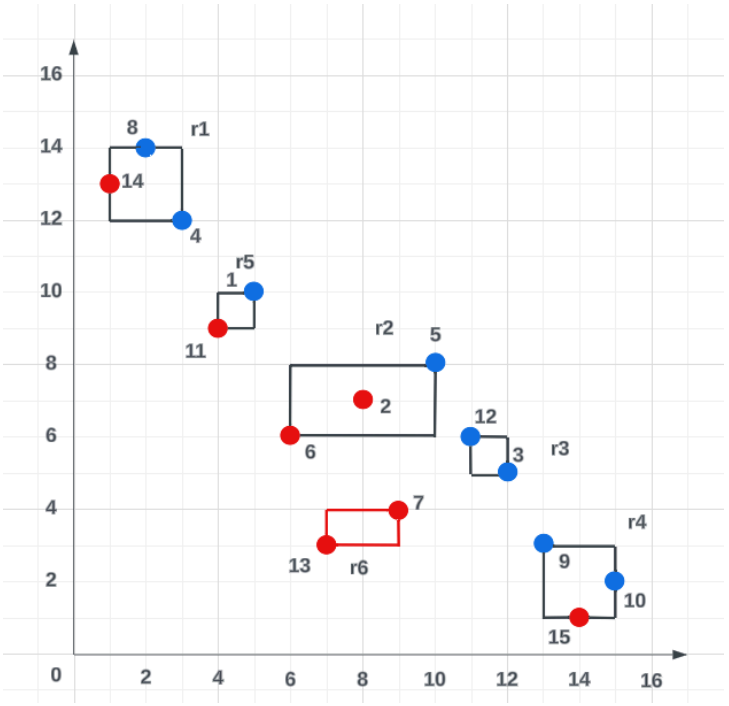
In the middle segment, Node r6 pruned because of both points being dominated by Point 5. Similarly, Points 2 and 6 are also dominated by Point 5, but the node containing the three points isn't going to be pruned as Point 5 emerges as the only local skyline point.

Right Side



In the right segment, only Point 15 of the bottom node is completely dominated, but all other points are identified as local skyline points as they aren't completely dominated by one another.

Combining the Results



When combining the results, all of the local skyline points highlighted in blue emerge as global skyline points due to none of them completely dominating each other. This is accurate with the BBS algorithm and highlights the dimension of accuracy that the divide-and-conquer idea provides when coupled with the BBS algorithm.