Implement and compare the following search algorithm:

- Linear search
- Binary search in sorted array.
- Binary search tree
- Red-Black Tree

How analysis works in program?

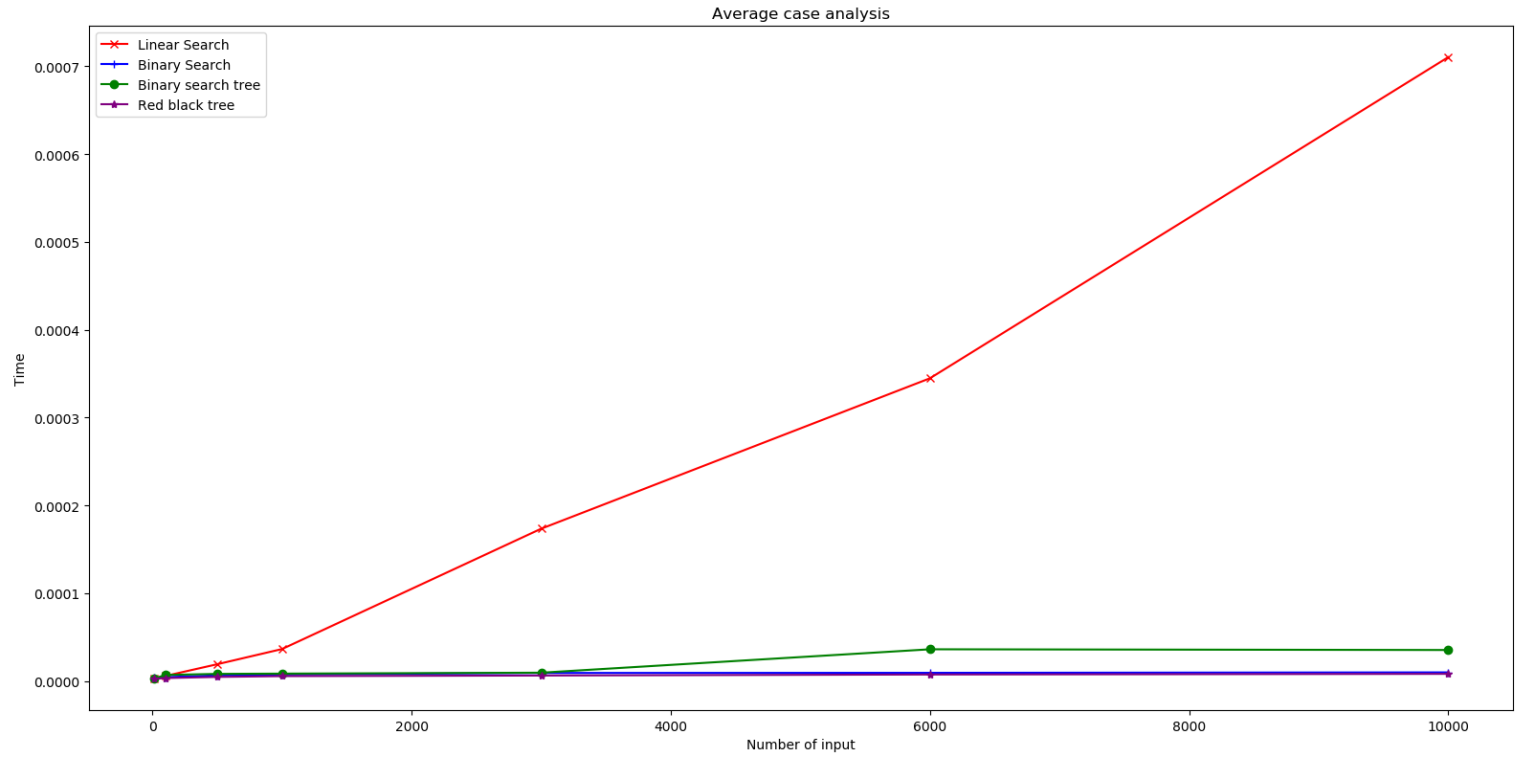
For analysis of different functions I have used timeit library of python which records running time of particular function. I have run all 4 search algorithms with respect to different data sizes and recorded time.

Two type of analysis is done in the program

1) Average case
2) Worst case

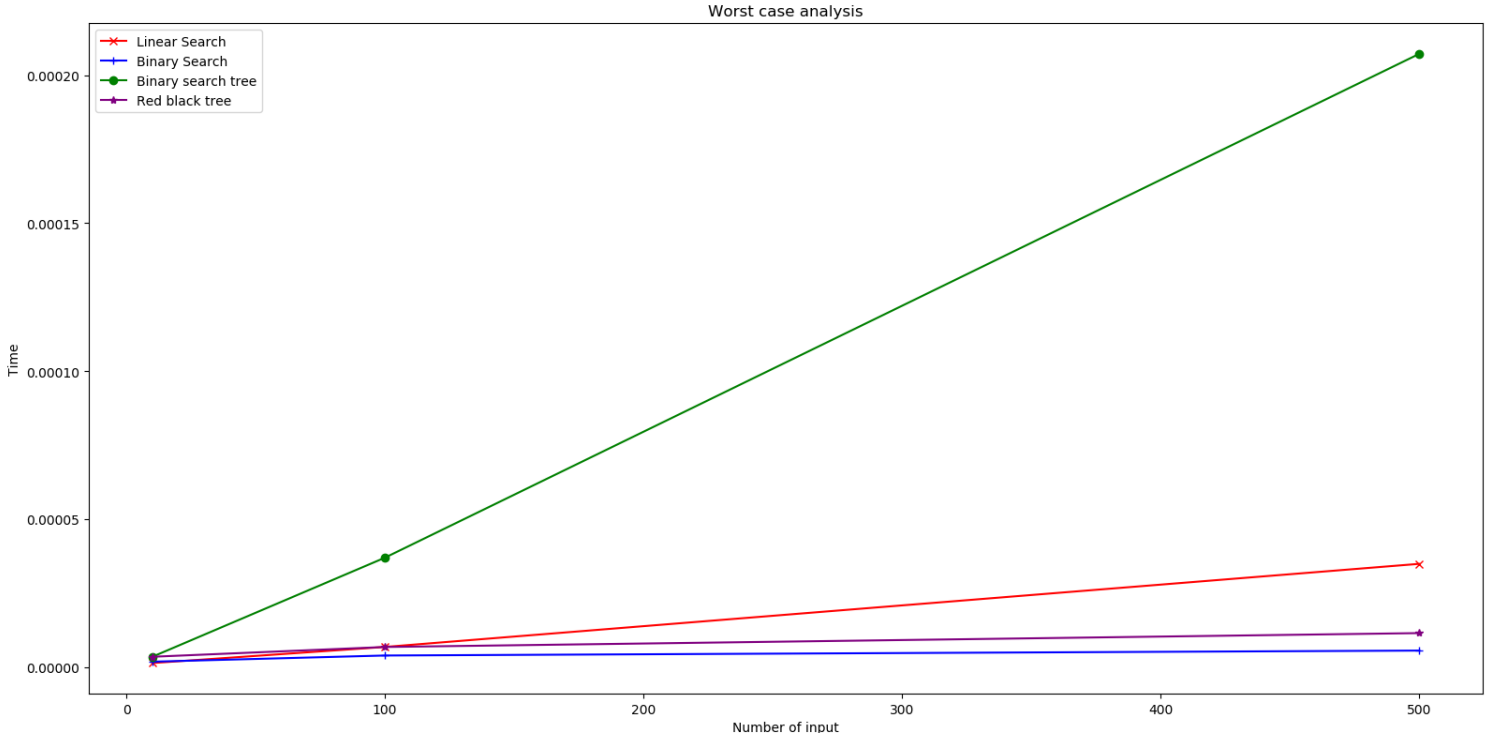
Average case analysis

For average case analysis of algorithm, I gave a random data as input and random key to search from array. Data size used for average case analysis is 10,100,500,1000,3000,6000,10000



Worst case analysis

For a worst case analysis as input I gave array that is sorted and I tried to search for the last element in the array. Reason to choose sorted array as input is to create the worst case scenario for binary search tree because for binary search tree for sorted array will completely right skewed and finding the last element will require the at maximum time.



Main components of program

Node class – A class to represent node of tree

Attributes :- key, right, left

rbtree class – A class to represent red black tree

Attributes :- root

rbnode class – A class to represent node of red black tree

Attributes :- key, left, right, parent, root, colour

Add function – This function is used to add nodes in binary search tree

This function checks for the value to be added is less that the root or greater than root and according to that it will call itself recursively. If value is greater then it will recurse in right part of tree otherwise it will recurse to left part of tree.

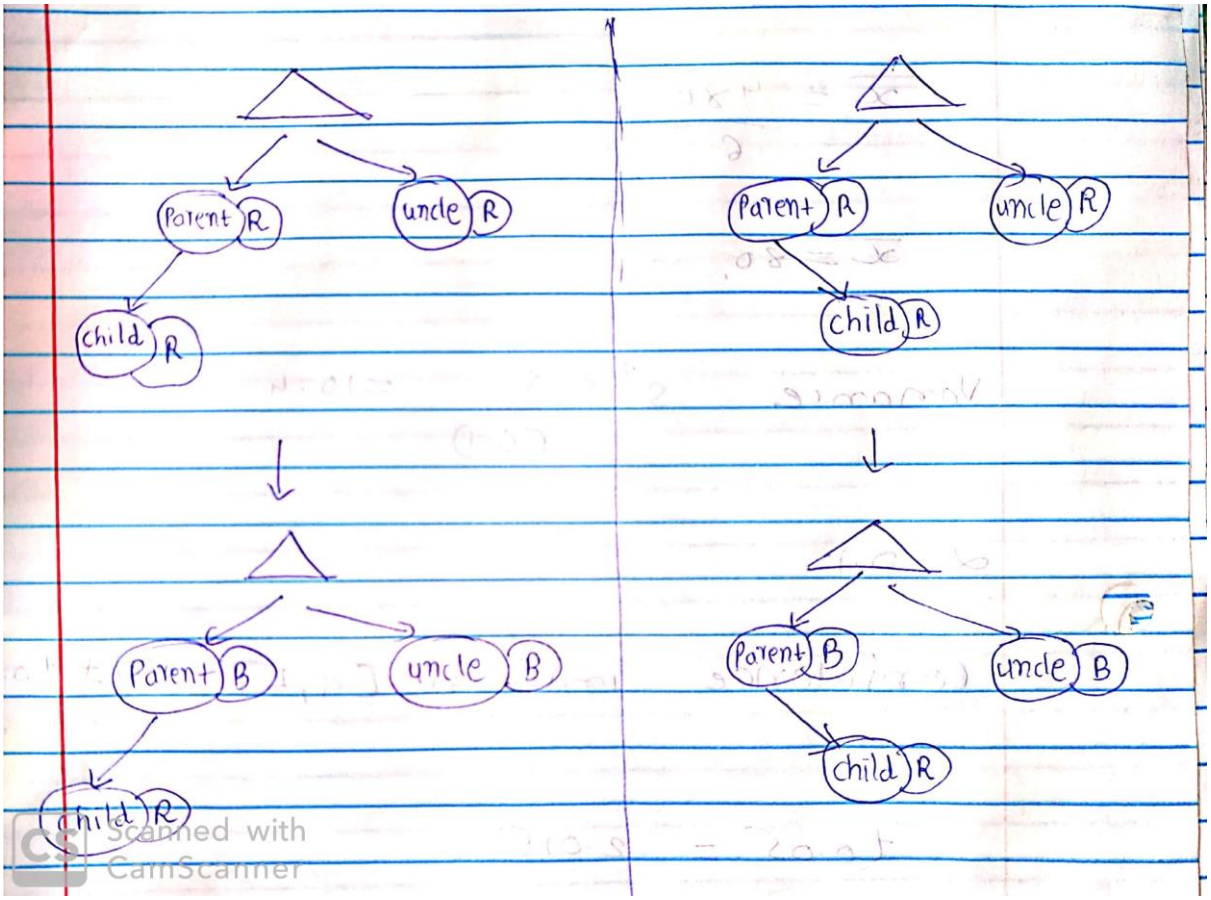
bst function – This function search for particular key in binary search tree

addrbt function – This function adds a node in red black tree. It follows the same methodology as binary search tree to add data in tree but after adding data it will check for conflict and solve it accordingly

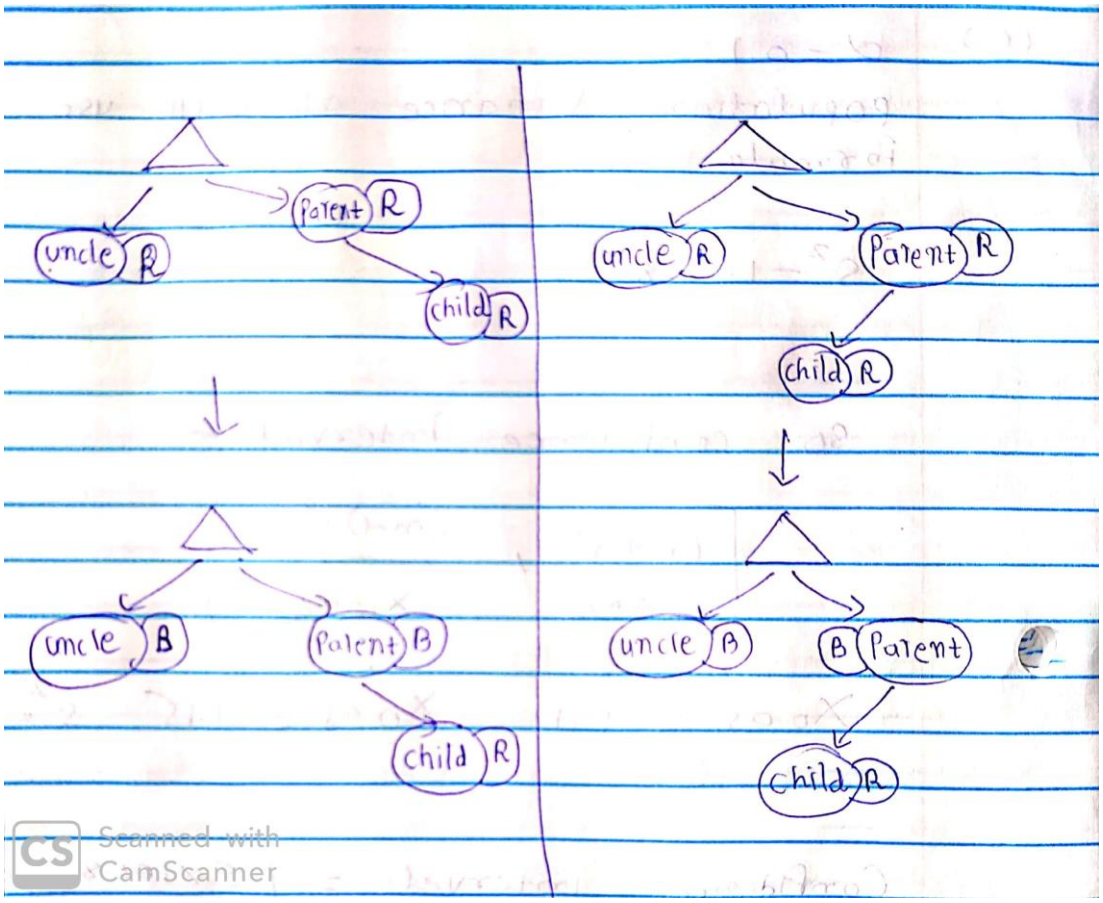
checkrr function – This function checks for red-red conflict in red black tree

solver function – This function solve the red-red conflict by applying particular case

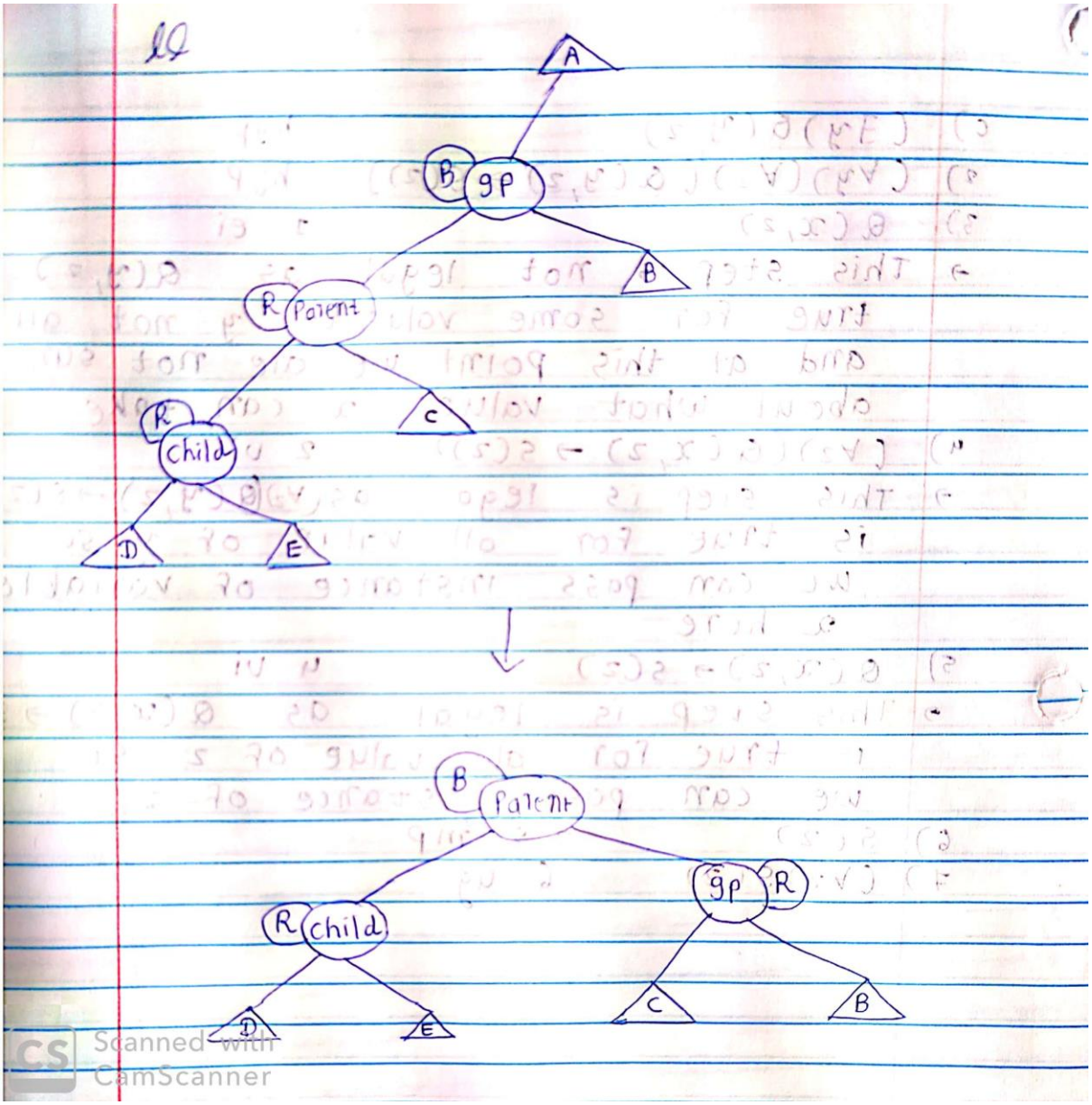
case1 function - This function solve the conflict if the tree is as follow. This case applies when parent and uncle has a same colour



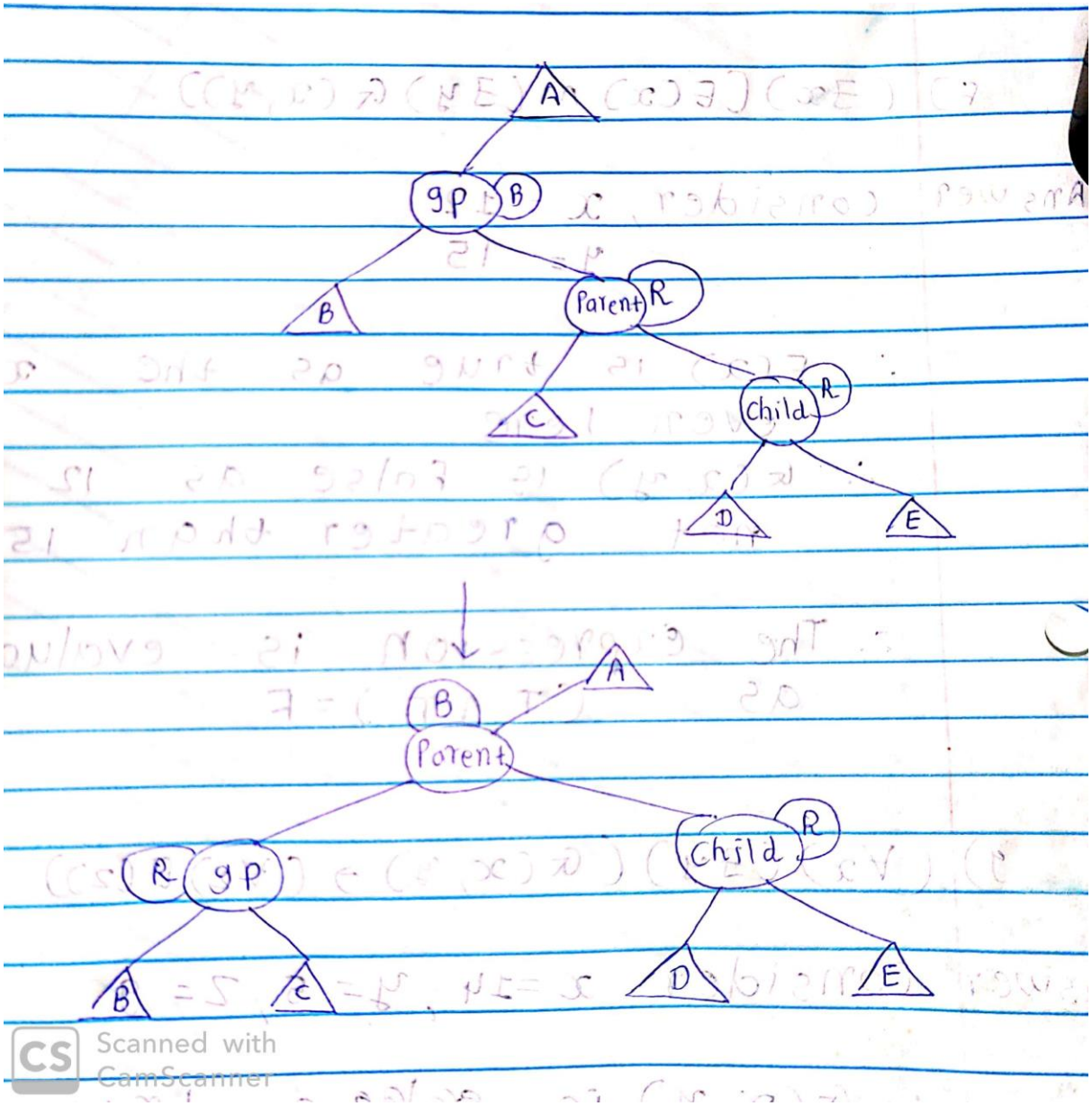
case2 function - This function solve the conflict if the tree is as follow. This case applies when parent and uncle has a same colour



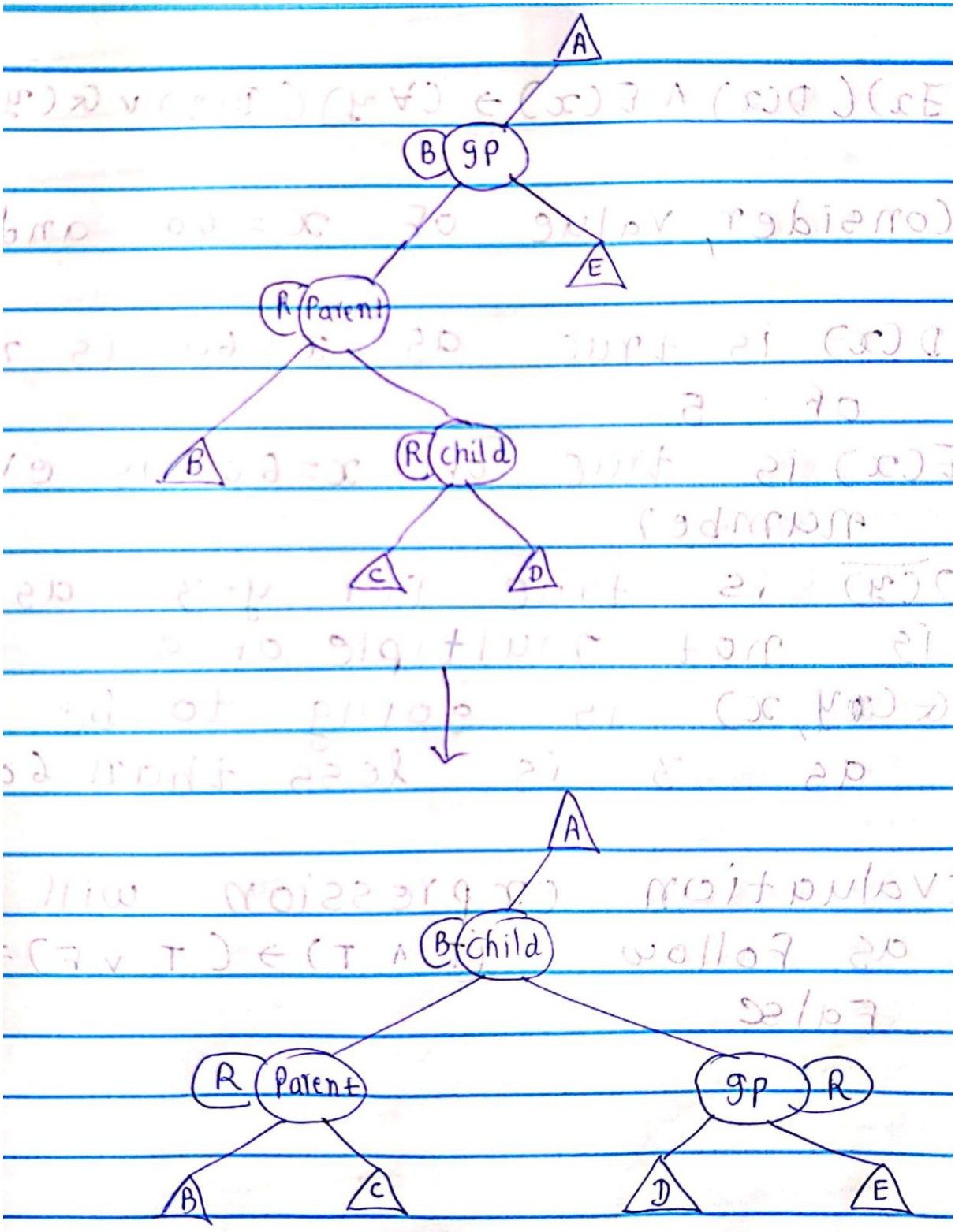
ll function – This function solve red-red conflict by applying left-left rotation. This function applies when uncle and parent has different colour.



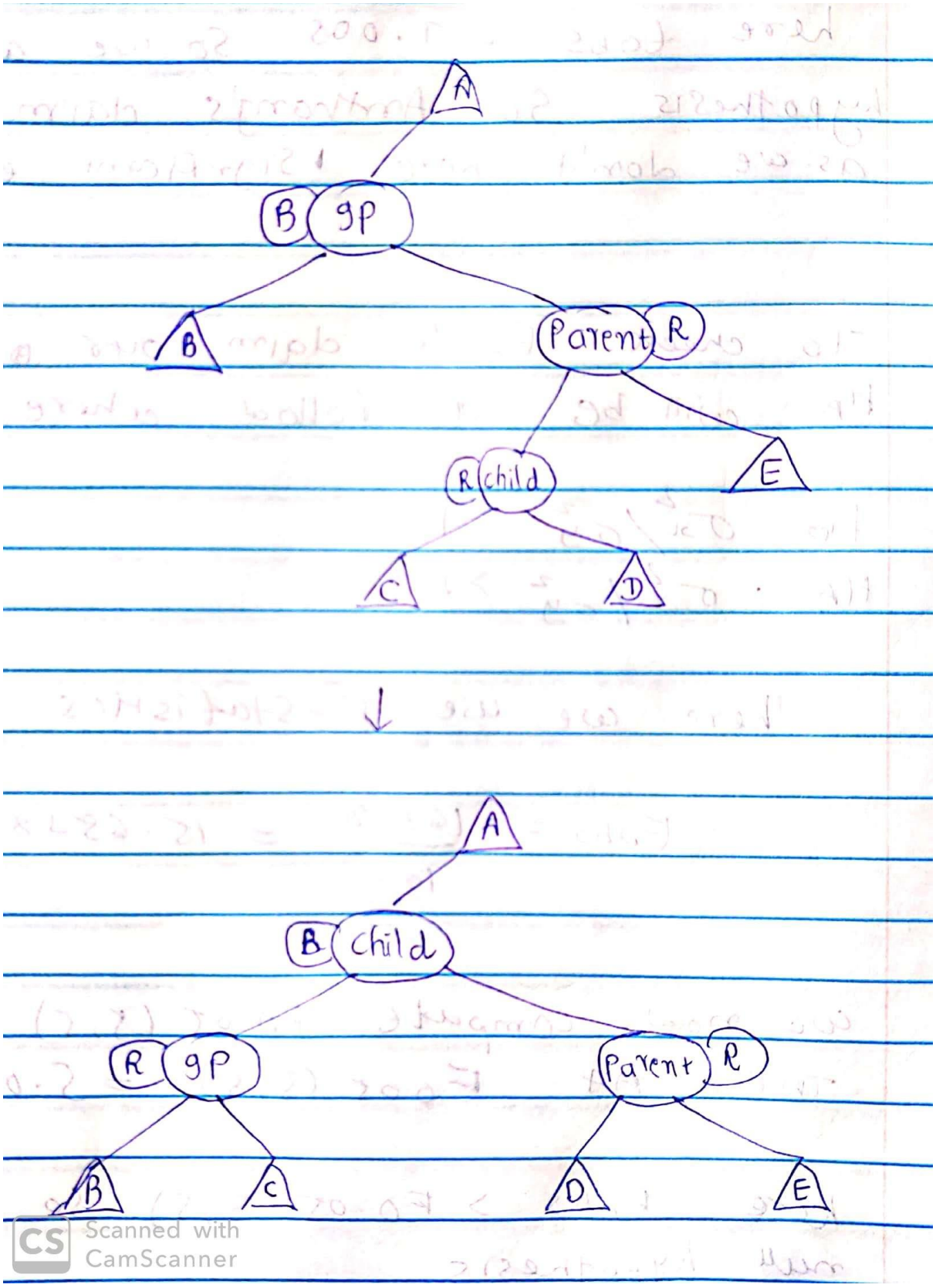
rr function - This function solve red-red conflict by applying right-right rotation. This function applies when uncle and parent has different colour.



lr function - This function solve red-red conflict by applying left-right rotation. This function applies when uncle and parent has different colour.



rl function - This function solve red-red conflict by applying right-right rotation. This function applies when uncle and parent has different colour.



Changecolor - This function checks if after changing colour at upper level red-red conflict occurs or not

searchtree function – This function search for particular key in binary search tree/Red black tree

bsearch function – This function performs binary search

lsearch function – this function performs linear search

Conclusion

After applying worst case and average case analysis on theses 4 algorithms I have reached to the conclusion that binary search and red black tree works efficiently with worst and best cases where as linear and binary search tree does not give a good performance with worst case. In the worst case binary search tree is the slowest algorithm. To improve the efficiency of binary search tree in worst cases we should apply some balancing technique on tree such as AVL or Red black tree.