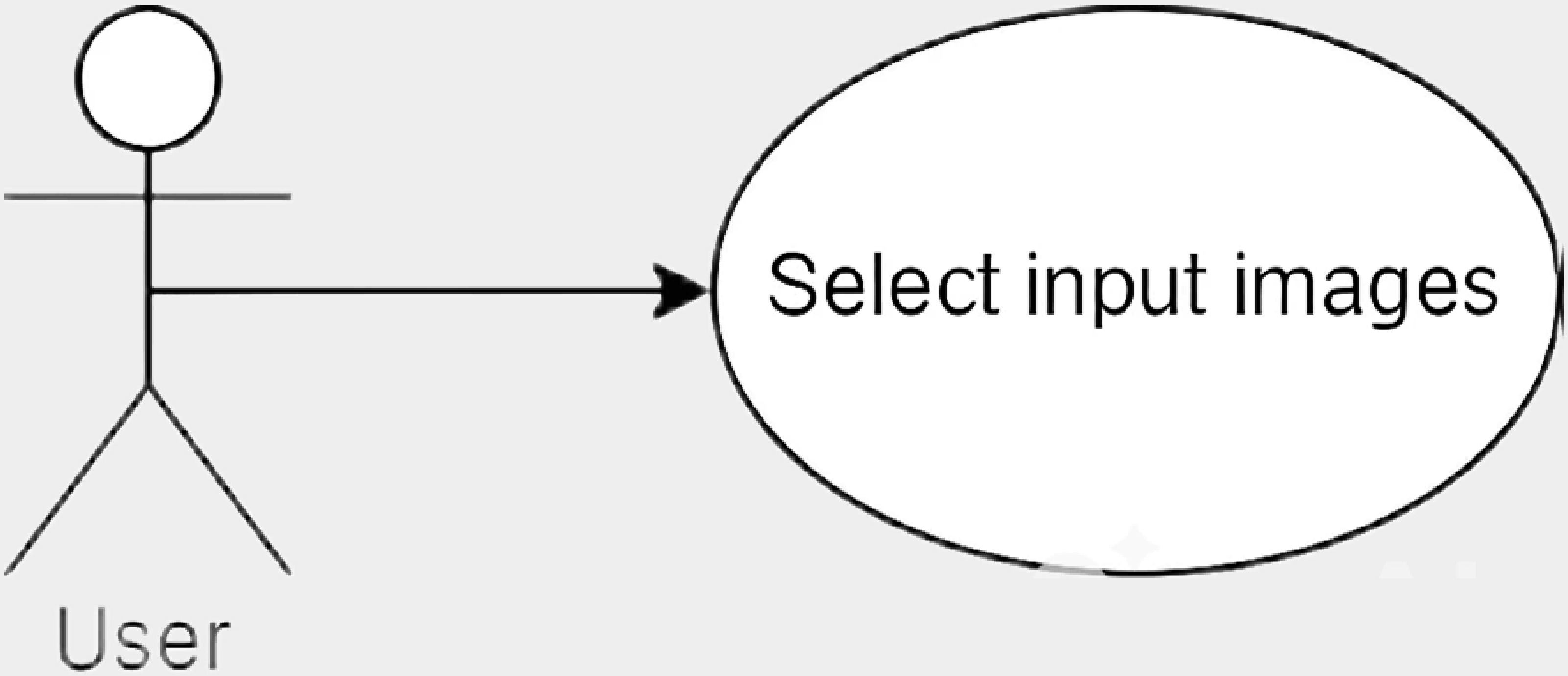
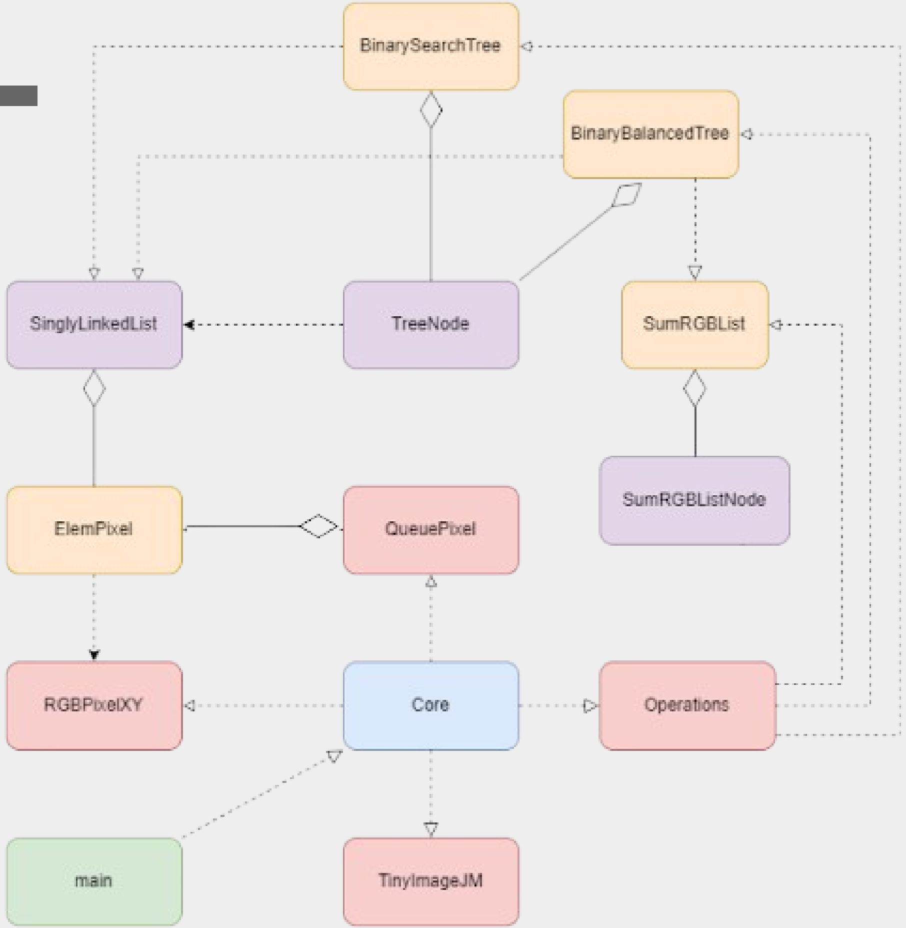
Treeant

Content

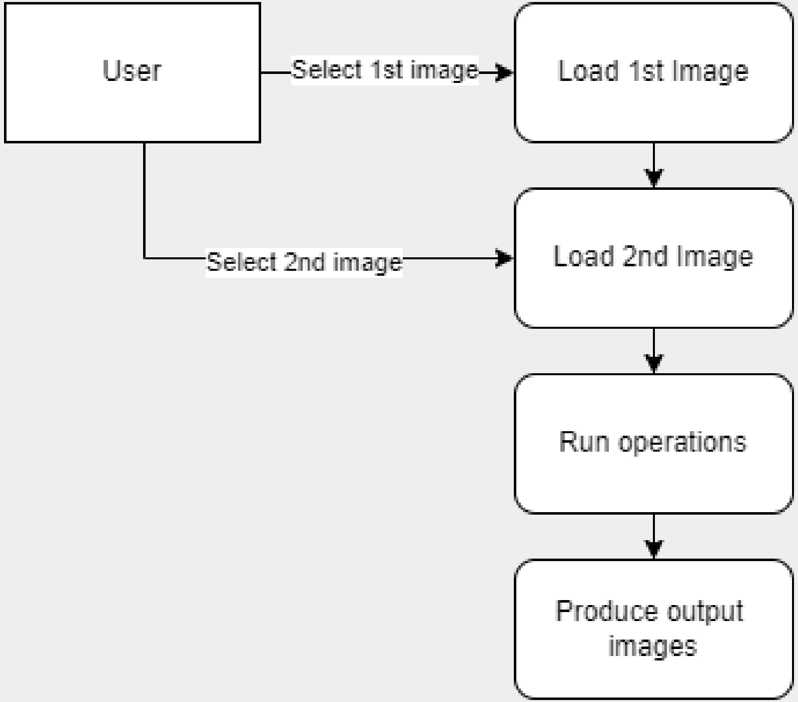
Use case diagram



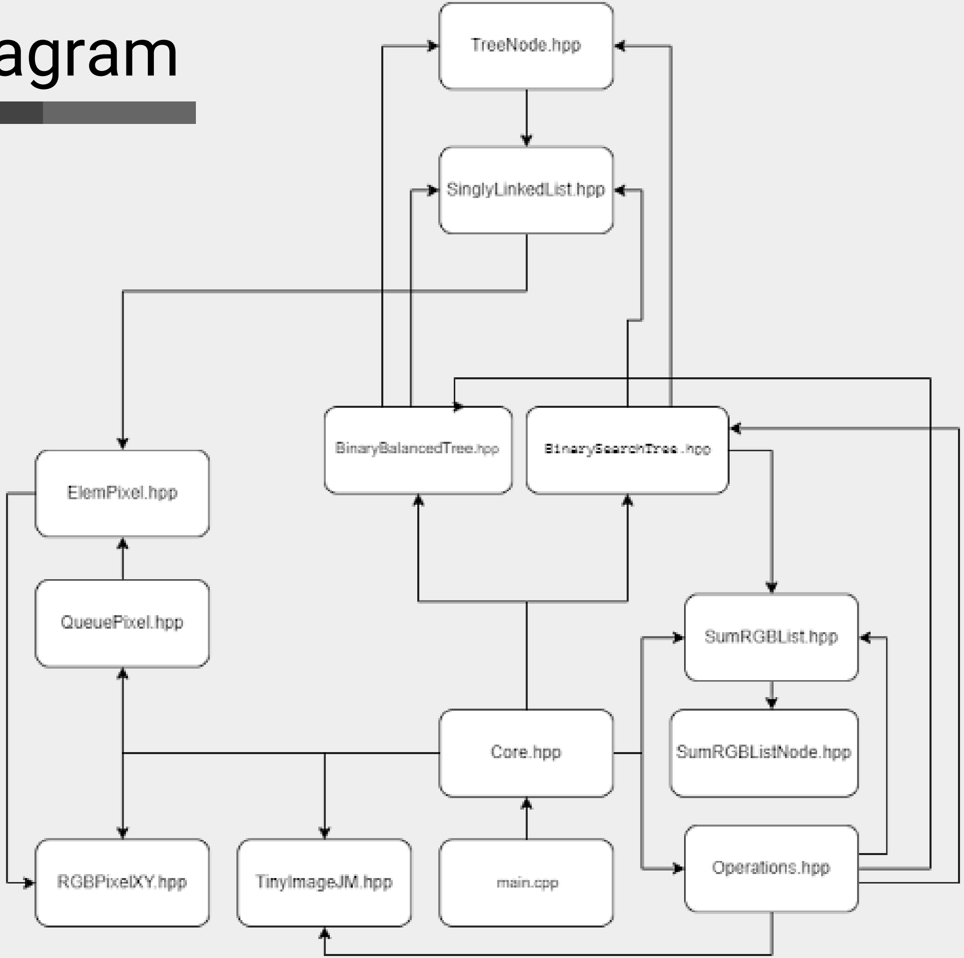
Class diagram



Data flow diagram



Source file diagram



How does Treeant works?

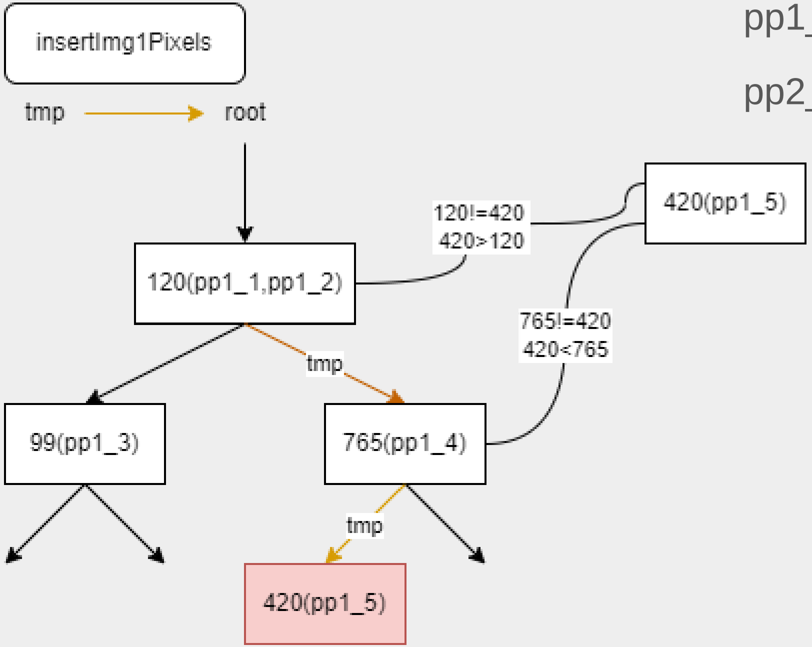
PHASE 1

How does Treeant works?

PHASE 2 - GROUP 1

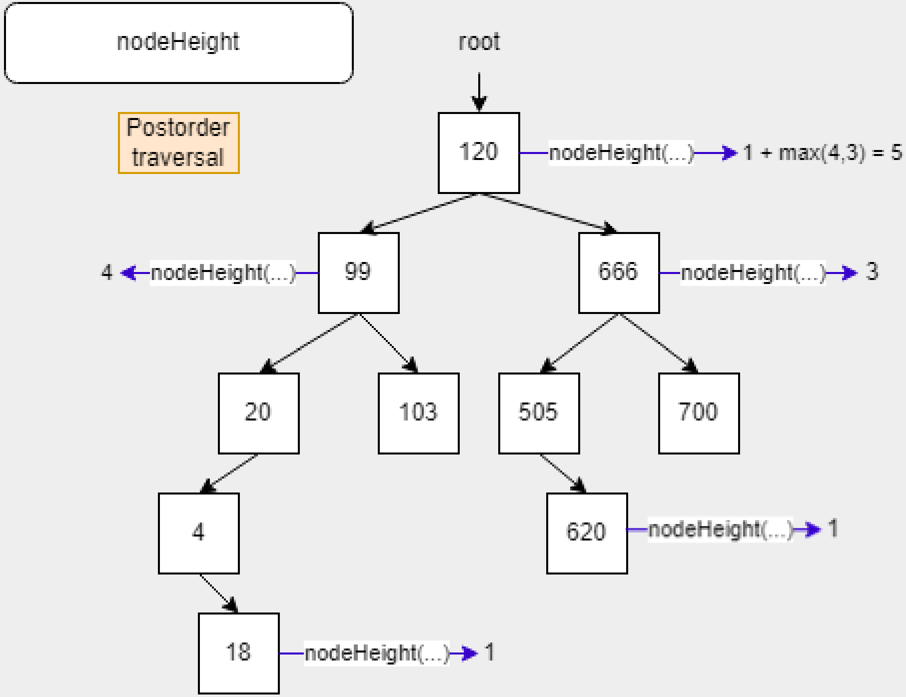
pp1_x pixel from first image

pp2_x pixel form second image



t1::insertImg1Pixels(RGBPixelXY* pixel)

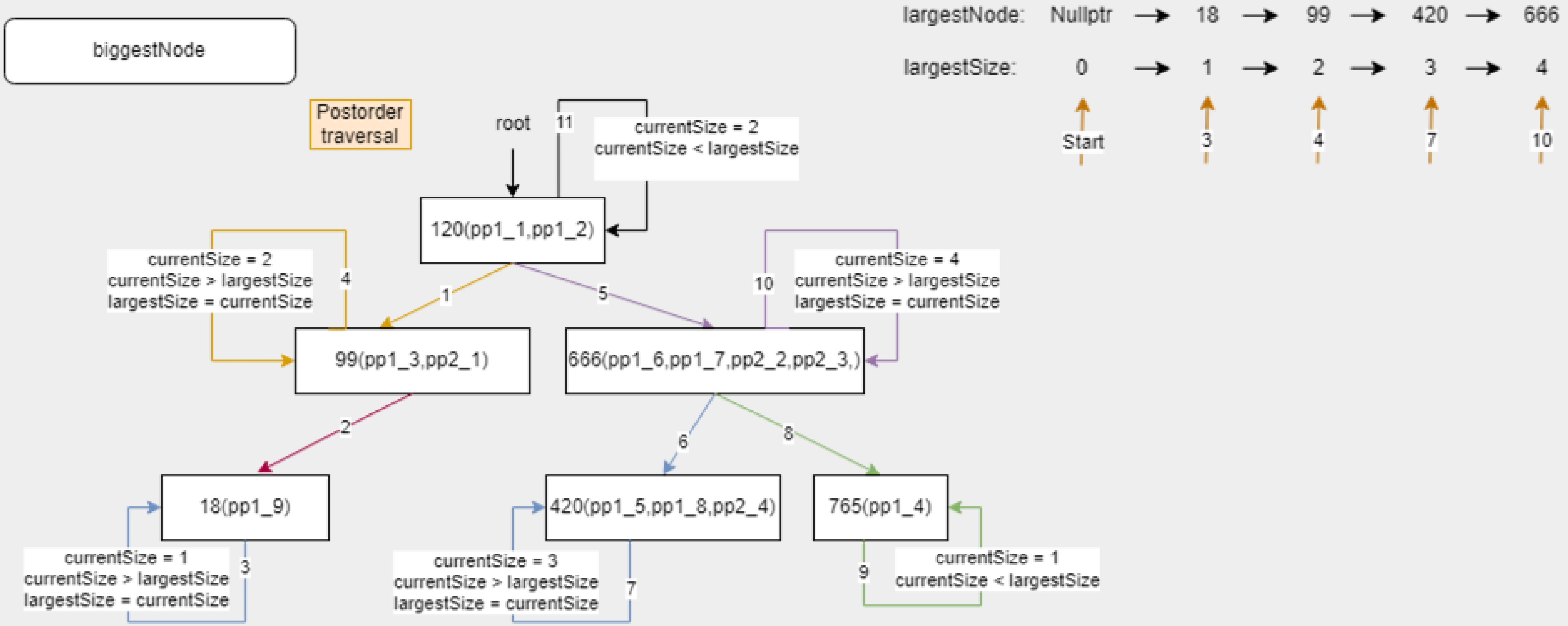
How does Treeant works? STATISTICAL MEASURES



Maximum tree depth

How does Treeant works?

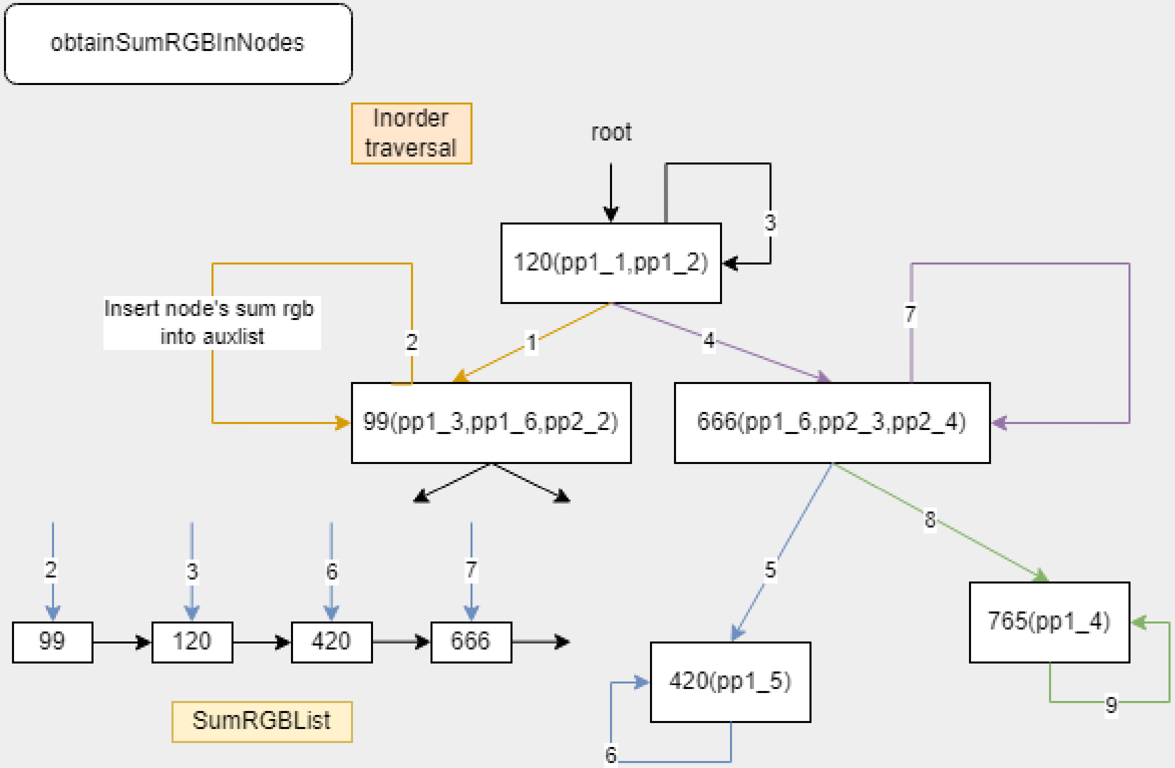
STATISTICAL MEASURES



biggestNode(TreeNode* node, TreeNode* &largestNode, int &largestSize)

How does Treeant works?

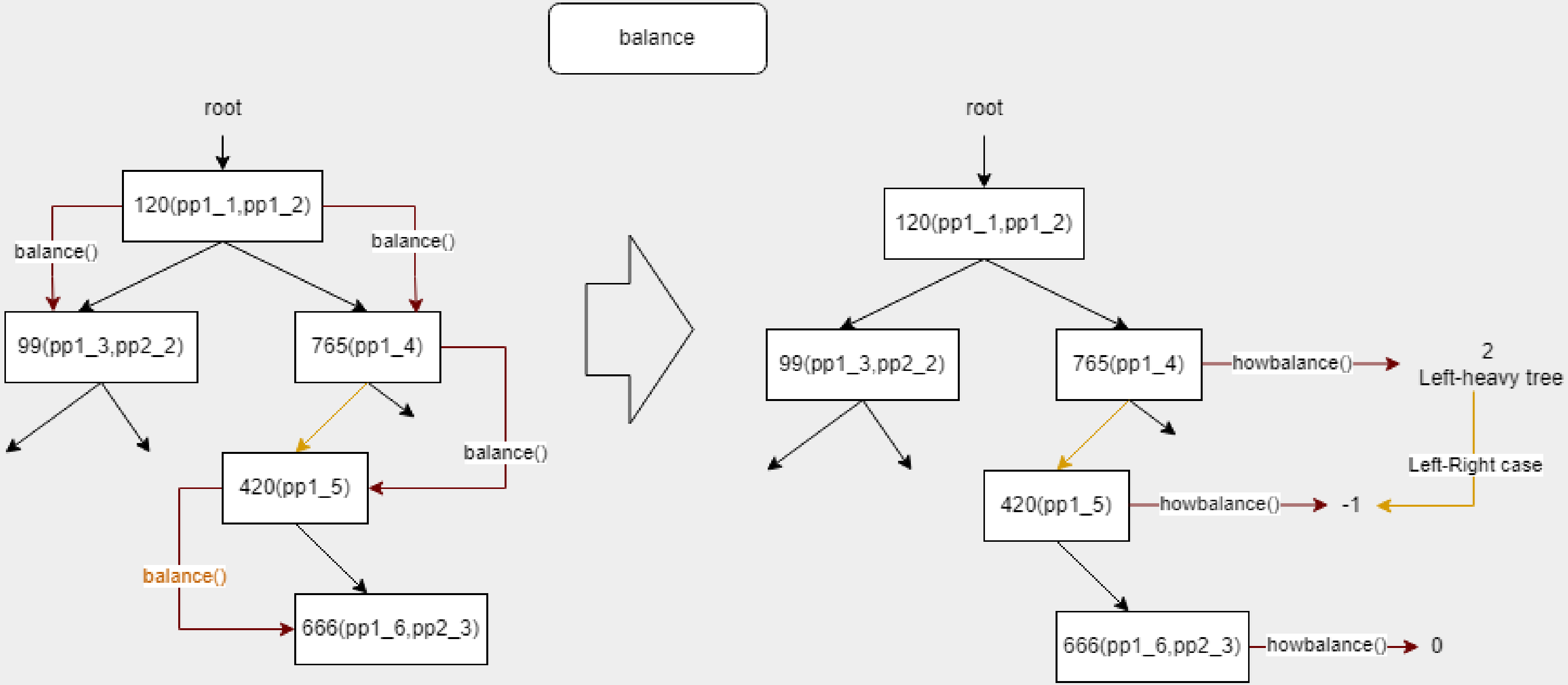
PHASE 2 - GROUP 1



Operations::t1Lists(BinarySearchTree* t1)

How does Treeant works?

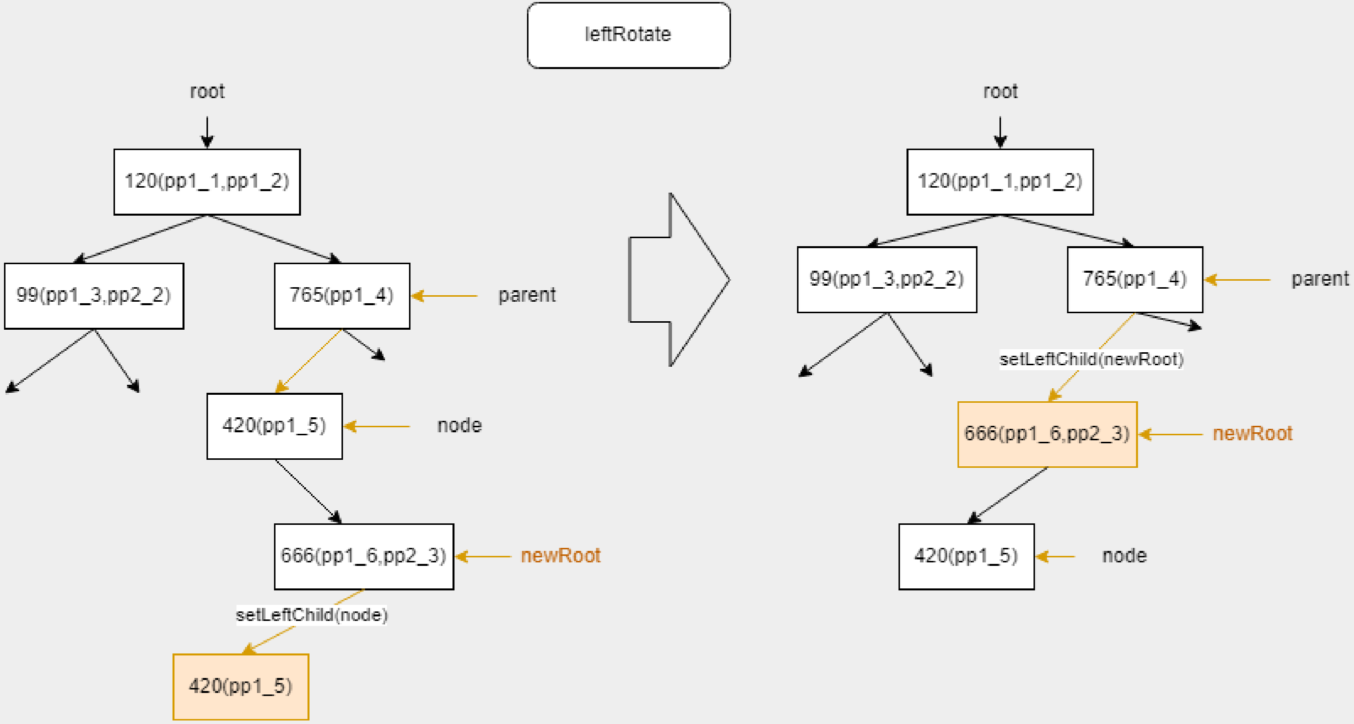
PHASE 2 - GROUP 1



t2::create_balance(TreeNode* node)

How does Treeant works?

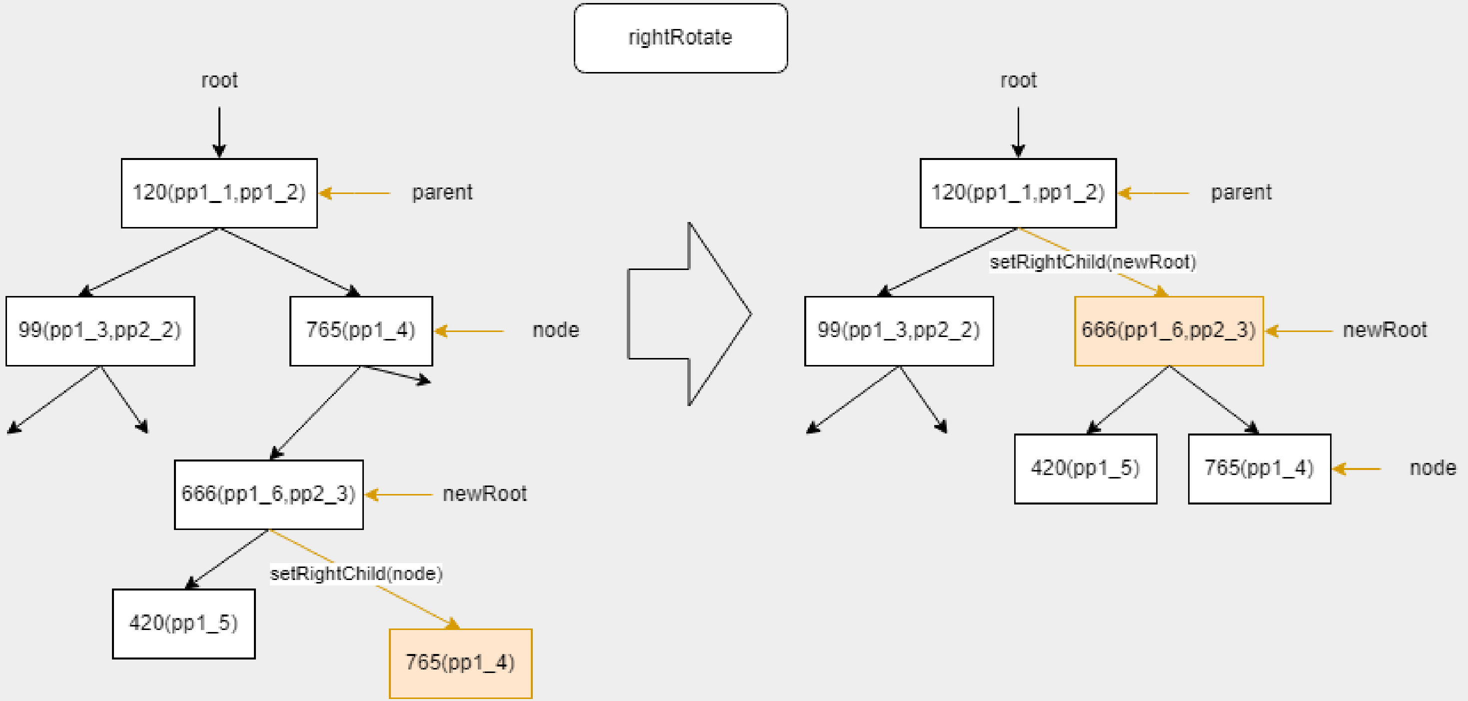
PHASE 2 - GROUP 1



t2::leftRotate(TreeNode* node, TreeNode* parent)

How does Treeant works?

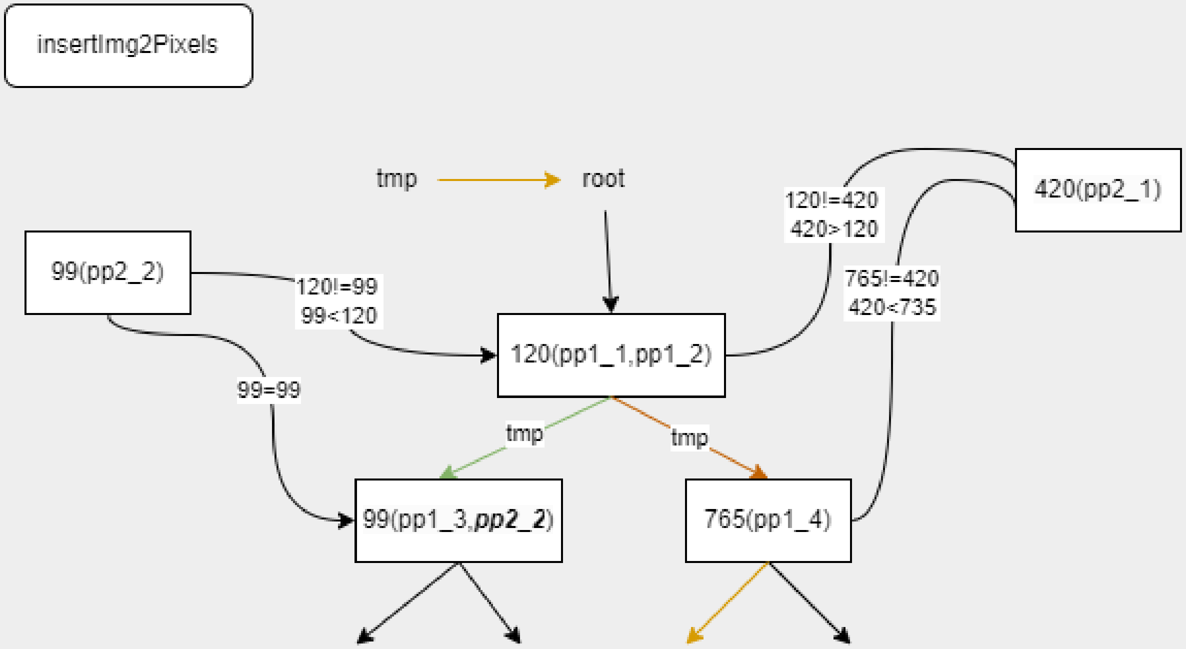
PHASE 2 - GROUP 1



t2::rightRotate(TreeNode* node, TreeNode* parent)

How does Treeant works?

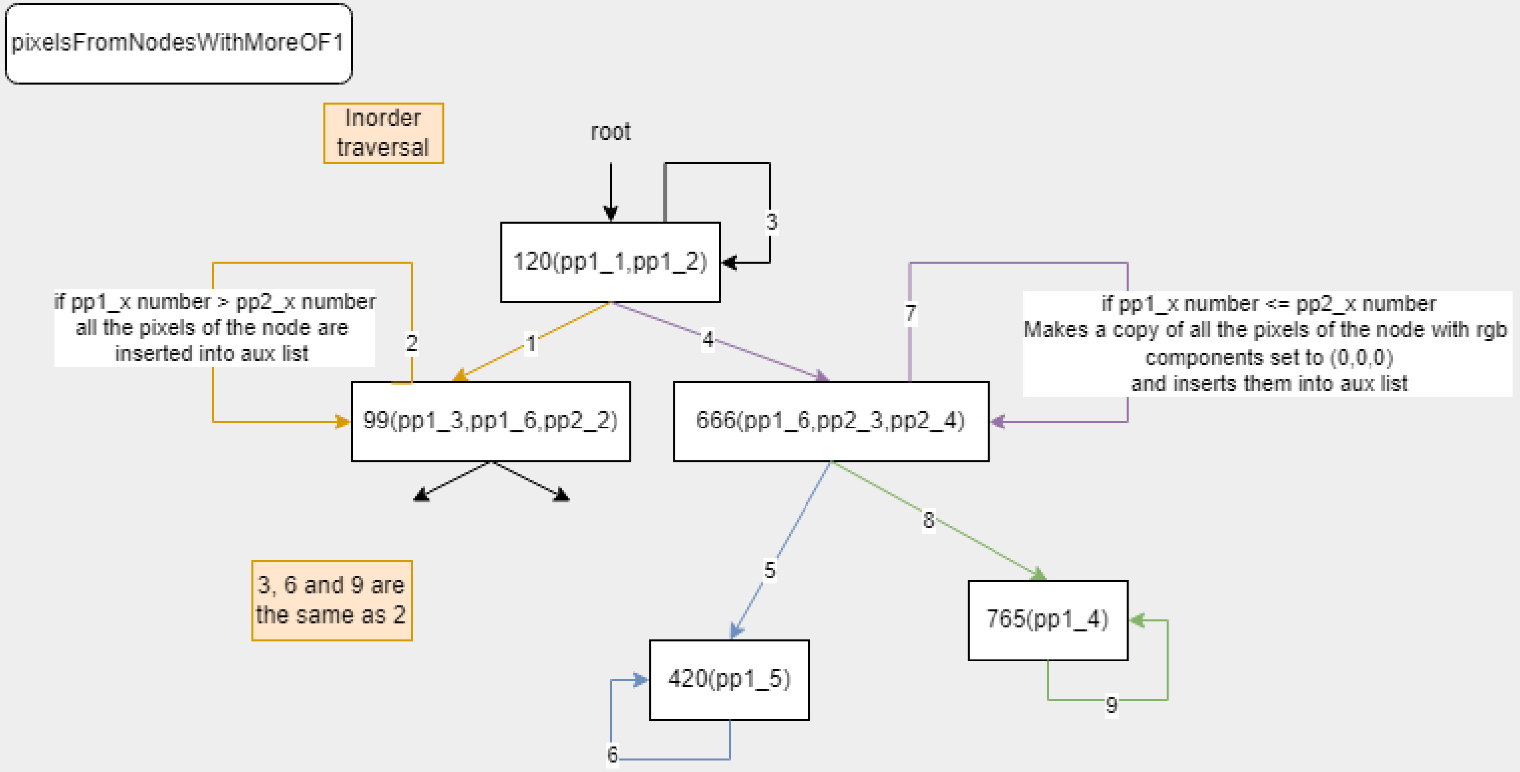
PHASE 2 - GROUP 2



insertImg2Pixels(RGBPixelXY* pixel)

How does Treeant works?

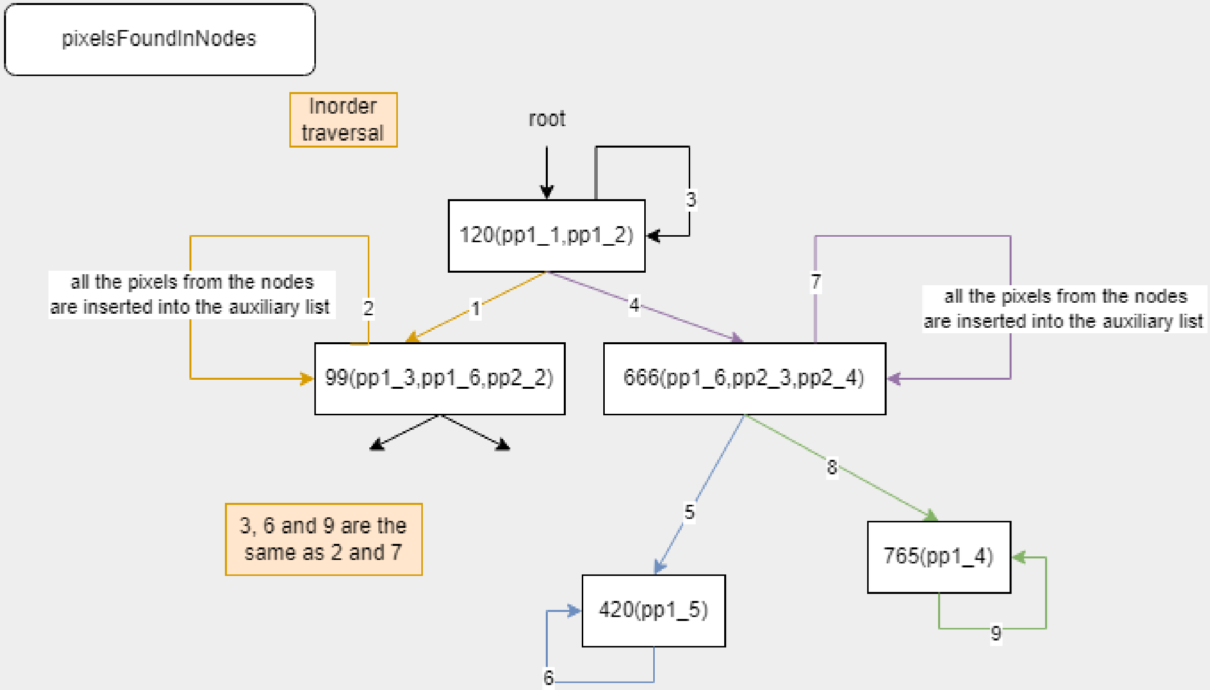
PHASE 2 - GROUP 3



Operations::Output1(BinarySearchTree* t1, TinyImageJM* img1)

How does Treeant works?

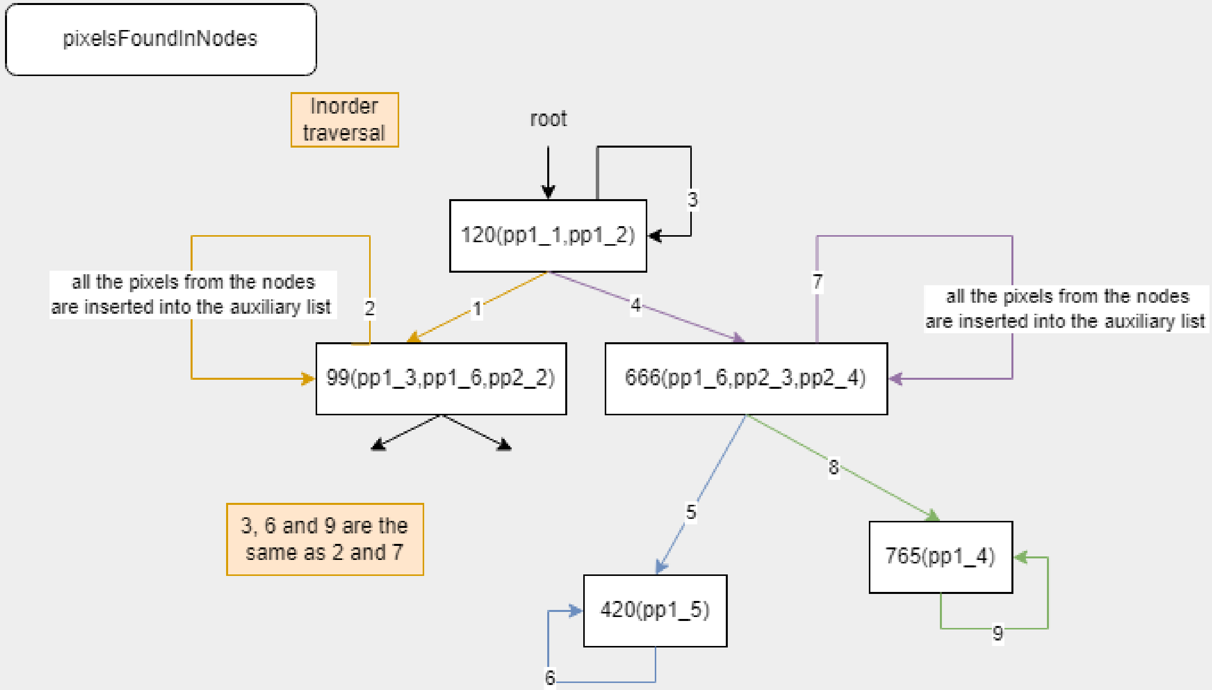
PHASE 2 - GROUP 3



Operations::Output2(BinarySearchTree* t1, TinyImageJM* img2)

How does Treeant works?

PHASE 2 - GROUP 3



Operations::Output3(BinarySearchTree* t2, TinyImageJM* img2)

RUNNING TIME EXPLANATION

Scope: Method Level

3 Phases:

-Pixel loading

> Loading pixels into Queue.
> Loading pixels into T1, first image and second image.
> Loading pixels into T2, first image and second image.

-Analysis

> T1 analysis after first image and second image.
> T2 analysis after first image and second image.

-Output generation

> Output 1 generation.
> Output 2 generation.
> Output 3 generation.

Pixel Loading

Loading images into *queuePixel*. → $O(N_1) + O(N_2)$

Loading pixels into *T1*. → $O(M*N_1) + O(M*N_2)$

Generic image loading: $O(D*N) \rightarrow O(M*N)$

Loading pixels into *T2*. → $O(N_1 + M^2) + O(N_2 \log(M))$

-Cloning: $O(M+N_1) \rightarrow O(N_1)$

-Balancing: $O(M*D) \rightarrow O(N_1 + M^2)$

-Balanced tree insert → $O(N_2 \log(M))$

Where N is the total number of pixels in the image, M the number of nodes in the tree, and D the depth of it.

Analysis

T1 analysis. $\rightarrow O(N_{1+2})$

- Maximum depth. $\rightarrow O(M) + O(M)$

- Biggest node: $O(M+N_1) + O(M+N_{1+2}) \rightarrow O(N_1) + O(N_{1+2})$

- List.: $O(M+M+M) \rightarrow O(M)$

T2 analysis. $\rightarrow O(N_{1+2})$

- Maximum depth. $\rightarrow O(M) + O(M)$

- Biggest node: $O(M+N_1) + O(M+N_{1+2}) \rightarrow O(N_1) + O(N_{1+2})$

Where N is the total number of pixels in the image, M the number of nodes in the tree.

Output generation

Output 1. $\rightarrow O(N_{1+2})$

Pixel collection: $O(M + N_{1+2}) \rightarrow O(N_{1+2})$

Pixel export: $O(M + N_{1+2}) \rightarrow O(N_{1+2})$

Output 2. $\rightarrow O(N_{1+2})$

Pixel collection: $O(M + N_{1+2}) \rightarrow O(N_{1+2})$

Pixel export: $O(M + N_{1+2}) \rightarrow O(N_{1+2})$

Output 3. $\rightarrow O(N_{1+2})$

Pixel collection: $O(M + N_{1+2}) \rightarrow O(N_{1+2})$

Pixel export: $(M + N_{1+2}) \rightarrow O(N_{1+2})$

Where N is the total number of pixels in the image, M the number of nodes in the tree.

The whole program efficiency

$\rightarrow O((N_1 * M) + (N_2 * M) + M^2 + (N_2 * \log(M))\ \Rightarrow N_2 * log(M) \leq N_2 * M$

$\rightarrow O((N_1 * M) + (N_2 * M) + M^2)\ \Rightarrow (N_1 * M) + (N_2 * M) = (N_L * M)$

$\rightarrow O((N_L * M) + M^2)\ \Rightarrow M^2 \leq (NL * M)$

$\rightarrow O(N_L * M)$

******N_L *equals the number of pixels from the largest of both images and M equals the number of nodes in T1/T2 (both have the same number of nodes).*

CONCLUSIONS