ZJU ID: 3210115439
Intl ID: yuyi.21
UIUC NetID: yuyiao2

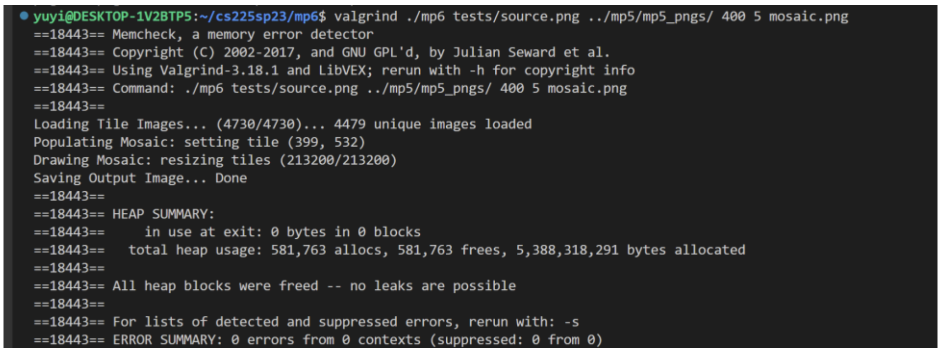
1. analysis of memory allocation and running time in your original MP5 implementation

I think my origin mp5 implementation is good enough and it meets all the requirements. So, I think for questions, there is no need to make extra explanation in my report. I will analyze the memory allocation and running time.

Let me analyze memory use it from 3 part.
Loading Tile Images: *n∗PNG* memory
Populating Mosaic: *w∗h∗C*

*I think my code does not use any extra location. In the first stage, it load n tile images so the total memory use is n*PNG. In the second stage, the memory use is c*w*h. w*h is the number of tiles we need. Since we pass pointers in the maptile function, each pointer needs c memory space so the total memory it uses is c*w*h.*

I use valgrind ./mp6 tests/source.png ../mp5/mp5_pngs/ 401 5 mosaic.png to see the heap memory use.

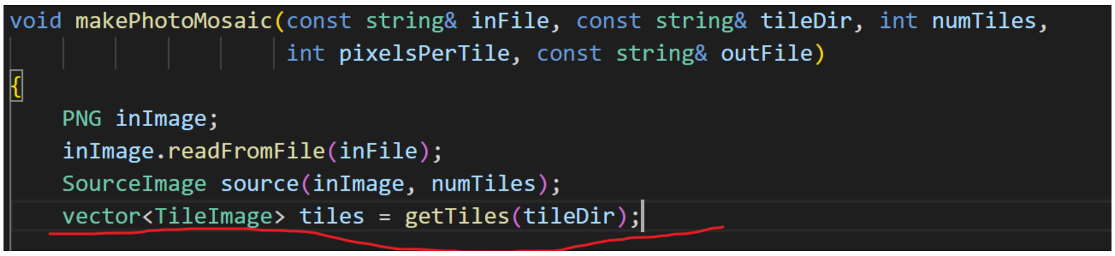


```
● yuyi@DESKTOP-1V2BTP5:~/cs225sp23/mp6$ valgrind ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png
  ==18443== Memcheck, a memory error detector
  ==18443== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
  ==18443== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
  ==18443== Command: ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png
  ==18443==
  Loading Tile Images... (4730/4730)... 4479 unique images loaded
  Populating Mosaic: setting tile (399, 532)
  Drawing Mosaic: resizing tiles (213200/213200)
  Saving Output Image... Done
  ==18443==
  ==18443== HEAP SUMMARY:
  ==18443==     in use at exit: 0 bytes in 0 blocks
  ==18443==   total heap usage: 581,763 allocs, 581,763 frees, 5,388,318,291 bytes allocated
  ==18443==
  ==18443== All heap blocks were freed -- no leaks are possible
  ==18443==
  ==18443== For lists of detected and suppressed errors, rerun with: -s
  ==18443== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Running time:

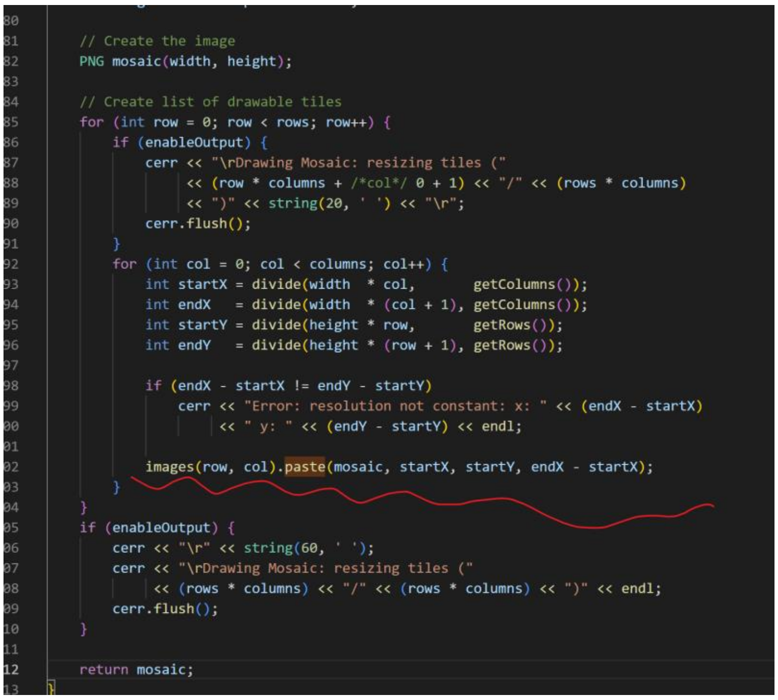
Let me analyze memory use it from 3 part.
Loading Tile Images: O(n) time
Populating Mosaic: O(w∗h∗lg(n))
Drawing Mosaic: O(w∗h∗w’∗h’)

1) We just need to analyze the getTiles function. We load n tile images and each loading operation takes O(1) time only because it just add the tileImage into the vector. So in total O(n) time.



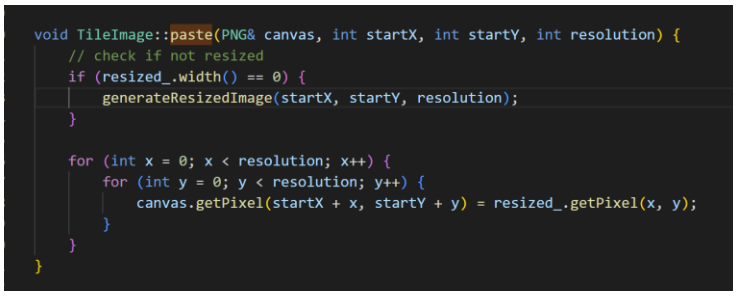
```
void makePhotoMosaic(const string& inFile, const string& tileDir, int numTiles,
                     int pixelsPerTile, const string& outFile)
{
    PNG inImage;
    inImage.readFromFile(inFile);
    SourceImage source(inImage, numTiles);
    vector<TileImage> tiles = getTiles(tileDir);
```

2) When we populate the mosaic, we have to fill w*h "pixel", which means we need to find w*h nearest neighbor in the KD-tree and each search takes O(logn) time. So in total O(w*h*lg(n)).
3) In this process, we need to analyze two functions.



```

    // Create the image
    PNG mosaic(width, height);

    // Create list of drawable tiles
    for (int row = 0; row < rows; row++) {
        if (enableOutput) {
            cerr << "\rDrawing Mosaic: resizing tiles ("
                 << (row * columns + /*col*/ 0 + 1) << "/" << (rows * columns)
                 << ")" << string(20, ' ') << "\r";
            cerr.flush();
        }
        for (int col = 0; col < columns; col++) {
            int startX = divide(width  * col,       getColumns());
            int endX   = divide(width  * (col + 1), getColumns());
            int startY = divide(height * row,       getRows());
            int endY   = divide(height * (row + 1), getRows());

            if (endX - startX != endY - startY)
                cerr << "Error: resolution not constant: x: " << (endX - startX)
                     << " y: " << (endY - startY) << endl;

            images(row, col).paste(mosaic, startX, startY, endX - startX);
        }
    }
    if (enableOutput) {
        cerr << "\r" << string(60, ' ');
        cerr << "\rDrawing Mosaic: resizing tiles ("
             << (rows * columns) << "/" << (rows * columns) << ")" << endl;
        cerr.flush();
    }

    return mosaic;
}
```

Each mosaic needs to draw w*h tiles. The code call paste for every tiles.

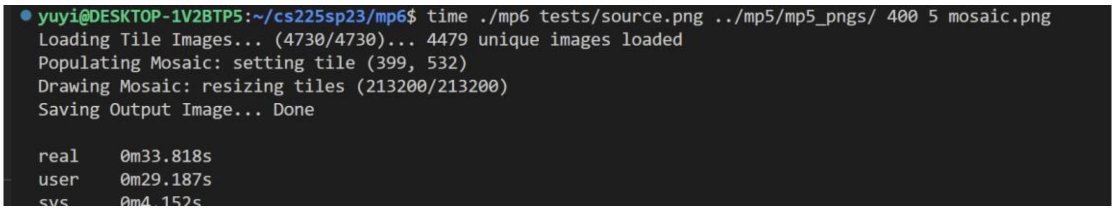


```
void TileImage::paste(PNG& canvas, int startX, int startY, int resolution) {
    // check if not resized
    if (resized_.width() == 0) {
        generateResizedImage(startX, startY, resolution);
    }

    for (int x = 0; x < resolution; x++) {
        for (int y = 0; y < resolution; y++) {
            canvas.getPixel(startX + x, startY + y) = resized_.getPixel(x, y);
        }
    }
}
```

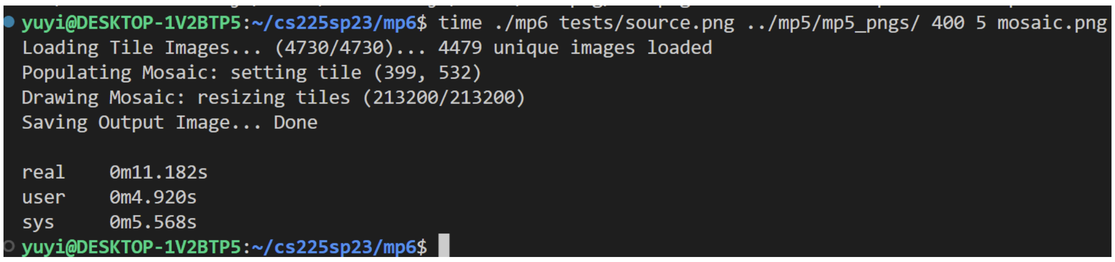
It takes w'*h' to time for each tilesimage because it draw it pixel by pixel. So in total O(w*h*w'*h').

I use time ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png to see running time.

2.analysis of memory allocation and running time in your new MP6 implementation



```
yuyi@DESKTOP-1V2BTP5:~/cs225sp23/mp6$ time ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png
Loading Tile Images... (4730/4730)... 4479 unique images loaded
Populating Mosaic: setting tile (399, 532)
Drawing Mosaic: resizing tiles (213200/213200)
Saving Output Image... Done

real    0m33.818s
user    0m29.187s
sys     0m4.152s
```



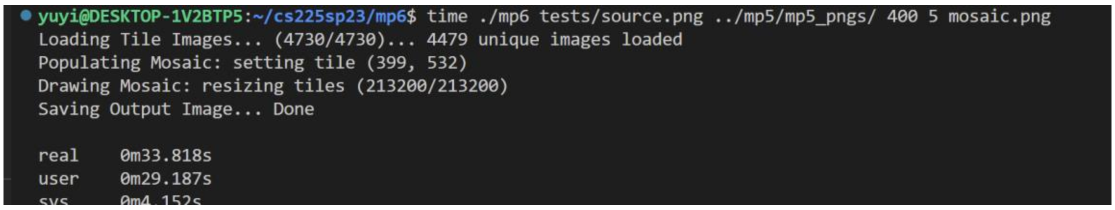
```
yuyi@DESKTOP-1V2BTP5:~/cs225sp23/mp6$ time ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png
Loading Tile Images... (4730/4730)... 4479 unique images loaded
Populating Mosaic: setting tile (399, 532)
Drawing Mosaic: resizing tiles (213200/213200)
Saving Output Image... Done

real    0m11.182s
user    0m4.920s
sys     0m5.568s
yuyi@DESKTOP-1V2BTP5:~/cs225sp23/mp6$
```

I use time ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png to see running time. Both the memory allocation and running time is the same as MP5 implementation.
I think for running time it is impossible to realize the goal O($w*h+n*w'*h'$), because we do need draw w*h tileimages and each image has w'*h' pixels. I cannot think of a way better than that.

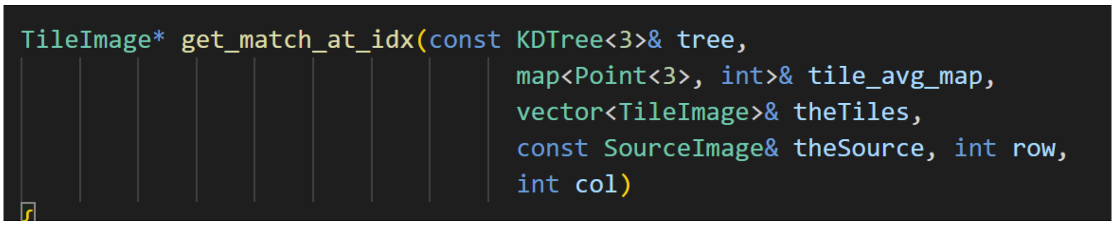
3.description of changes made to reduce memory footprint and running time

I think there is no way to reduce memory footprint or running time since it is good enough. However, I find it possible to make my code run faster.

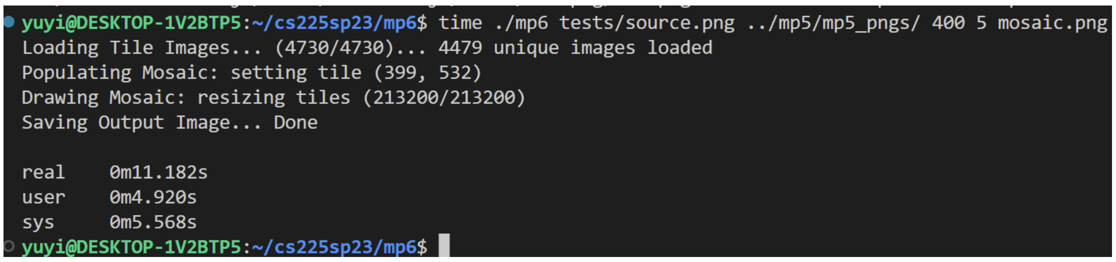
In the mapTiles function, I find previous helper function "get_match_at_idx" pass the map by value. I change the function so it pass the map by reference and it run much faster.



```
yuyi@DESKTOP-1V2BTP5:~/cs225sp23/mp6$ time ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png
Loading Tile Images... (4730/4730)... 4479 unique images loaded
Populating Mosaic: setting tile (399, 532)
Drawing Mosaic: resizing tiles (213200/213200)
Saving Output Image... Done

real    0m33.818s
user    0m29.187s
sys     0m4.152s
```



```
TileImage* get_match_at_idx(const KDTree<3>& tree,
                            map<Point<3>, int>& tile_avg_map,
                            vector<TileImage>& theTiles,
                            const SourceImage& theSource, int row,
                            int col)
```



```
yuyi@DESKTOP-1V2BTP5:~/cs225sp23/mp6$ time ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png
Loading Tile Images... (4730/4730)... 4479 unique images loaded
Populating Mosaic: setting tile (399, 532)
Drawing Mosaic: resizing tiles (213200/213200)
Saving Output Image... Done

real    0m11.182s
user    0m4.920s
sys     0m5.568s
yuyi@DESKTOP-1V2BTP5:~/cs225sp23/mp6$
```