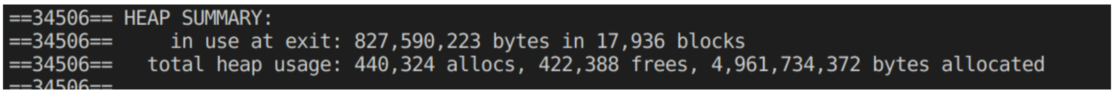
Report
Fang Junzhou, 3210115452
junzhou.21; junzhou5

## Space Complexity

In MP5, the space complexity of the code has already been optimized. All the tile images are loaded into a vector in getTiles(), occupying n*w'*h' memories. Then when matching tile images to the base picture, in mapTile() and other called functions, I use a pointer, TileImage*, rather than copying the tile image, to link corresponding tile image to each pixel. Therefore, the total space complexity is optimized to w*h+n*w'*h'+n*c, where the first two terms are unavoidable for loading base and tile images, and the last term refers to when populating tiles, using a pointer to represent each tile image.

As a result, the code will take approximately 800 MB when running on a base png of size 605*453 with 4730 tile images with each of size 75*75. This should be a reasonable number. Testing by valgrind with *valgrind ./mp6 tests/source.png mp5_pngs/ 400 5 mosaic.png*



```
==34506== HEAP SUMMARY:
==34506==     in use at exit: 827,590,223 bytes in 17,936 blocks
==34506==   total heap usage: 440,324 allocs, 422,388 frees, 4,961,734,372 bytes allocated
==34506==
```

## Time Complexity

The time complexity of drawing the mosaic is already O(w*h+n*w'*h') if take input resolution as a constant, where

w*h: two for-loops in drawMosaic, traversing through each row and column of the mosaic.

n*w'*h': each of the tile image needs to be resized

Therefore, the total time complexity is O(w*h+n*w'*h'). Although it can be optimized to O(w*h) for this part by resizing all the tile images at loading step, but this will not contribute to an optimization to the general time complexity. Therefore, I consider it meaningless.

However, the time complexity of mapTiles() can be improved. Here a map of size n that mapping the average value to the tile index is generated, and when the function calls get_match_at_idx(), it passes this map as an object. This means each time when get_match_at_index() is called, it will copy the whole map. It will run for extra O(n*w*h).

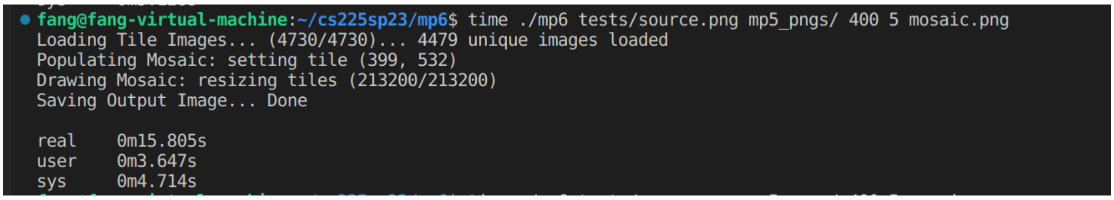
Therefore, in MP6, I change the signature of get_match_at_idx() as

```
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)
```

In such way the map will only be generated once, the theoretical time complexity of get_match_at_idx() will be a constant. What's more, I deleted the error detection code in get_match_at_idx(), which previously is

```
// Check to ensure the point exists in the map
        map< Point<3>, int >::iterator it = tile_avg_map.find(nearestPoint);
        if (it == tile_avg_map.end())
                cerr << "Didn't find " << avgPoint << " / " << nearestPoint << endl;
```

This also take an O(n) traversal. By deleting this part the running time gets further improved. Testing with time with *time ./mp6 tests/source.png mp5_pngs/ 400 5 mosaic.png*



```
● fang@fang-virtual-machine:~/cs225sp23/mp6$ time ./mp6 tests/source.png 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    0m15.805s
  user    0m3.647s
  sys     0m4.714s
```