Assignment 4

I have done abstract modeling to calculate big-O, I didn't include all constants and lines, just the important ones affecting the growth analysis

ArrayListMatrix class

Method construct in ArrayListMatrix class

Ln#	Function construct	Cost	times	comments
1	`while (scan1.hasNext() && i<listSize) {`	Constant will keep it O(1)	N+1 times(N successes and 1 fail)	List size= nom of neighboring pixels
2	`for (int z = 0; z < matSize; z++) {`	1+(n+1)+2n=3n+2	N times	Matsize= size of neighbor(3/5/7/9/11) will represent as 'n'
3	`for (int j = 0; j < line.length; j++){`	3n+2	N(3n+2) times	Line length=size of neighbor usually(3/5/7/9/11) Will represent as 'n'
4	`if(line[j].length()>0)`	3	N(3n+2)(3n+2)	3 accesses and a comparison
5	`t[z][j] = Integer.parseInt(line[j]); } }`	4 (3 accesses and an arithmetic)	N(3n+2)(3n+2)	End of the nested for loop
5	`matrixList.add(t);`	N(insert in arraylist is of cost O(N),	N times	
6	`i++ }`	2 (arithmetic and assigning)	N times	End of while loop

F(n)=N+1+ N(3n+2)+ N(3n+2)(3n+2)+ N(3n+2)(3n+2)*3+N(3n+2)(3n+2)*4+N*N+N*2

In our assignment, number of neighboring pixels N is way larger than size of neighbor n, hence my f(N) can be concluded to(taking n worst case possible =11) :

F(N)construct=N^2+35N+1225N+3675N+4900N+1= N^2+9835N+1 abstractly

Therefore, big-O of construct is O(N^2)

Adj Matrix class

Method populate in AdjMatrix class

Ln#	Function populate	Cost	Times	comments
1	`for (int i = 0; i < adjMatSize; i++)`	(N+1)+1+N=2N+2	1	adjMatSize= nom of neighboring pixels in the data file=N
2	`for (int j = 0; j < adjMatSize; j++)`	2N+2	N	
3	`Integer[][] mat1 = mat.getAt(i)`	1	N^2	mat.getAt(i) ; get method of arrat list has cost 1
4	`Integer[][] mat2 = mat.getAt(j)`	1	N^2	
5	`int diff = getDifference(mat1, mat2);`	F(n) diff	N^2	
6	`adjMatrix[i][j] = (diff);`	3	N^2	2 accesses and one arithemtic

F(N) populate=2N+2+n(2N+2)+N^2+N^2+f(n)getdiff *N^2+3N^2 =8N*N+4N

F(n)getDiff is of complexity O(1), thereby we can conclude our F(N) to be of time complexity O(N^2)

Method get Difference

Ln#	Function getDiference	Cost	Times	Comments
1	`for (int i = 0; i < mat1.length; i++)`	2n+2	1	Mat1.length= size of the neighbouring pixel(3/5/7..) =n
2	`for (int j = 0; j < mat2.length; j++)`	2n+2	n	
3	`diff += mat1[i][j] - mat2[i][j];`	7	n^2	4 accesses, 1 assignment, 2 arithmetics
4	`return java.lang.Math.abs(diff)`	1	1	

F(n)getDiff =2n+2+2n^2+2n+7n^2+1

But since the size of the pixel we have taken will be relatively small, we can conclude our method to have constant complexity of O(1)

Method writeToFile

Ln#	Function writeToFile	Cost	Times	Comments
1	`writer = new PrintWriter(new FileWriter(outFileName));`	Constant cost	1	Initializing my writer
2	`for (int i = 1; i <adjMatSize + 1; i++)`	2N+2	1	N : size of the adjacency matrix, or the number of neighboring pixels in the data file
3	`for (int j = 1; j < adjMatSize + 1; j++)`	2N+2	N	
4	`writer.println(i + " - " + j + "\t" + adjMatrix[i - 1][j - 1])`	constant	N^2	I have 2 accesses, and the cost of printing to the counsel

F(N)WriteToFIle= constant+2N+2+2N^2+2N+constant *N^2=3N*N+4N

Thereby bigO of F(N) is O(N^2)

MST class

Method primMST

Ln#	Function primMST	Cost	Time	comments

1	`for (int i = 0; i < V; i++) {`
	`key[i] = Integer.MAX_VALUE`
	`mstSet[i] = false;`
	`}`
		.2N+2
		.2(assigning +access)
		. 2(assigning +access)
			.1
			.N
			.N
				V=N= number of vertices= nom of neighboring cells

2	`for (int count = 0; count < V - 1; count++) {`
	`int u = minKey(key, mstSet);`
	`mstSet[u] = true;`
	`for (int v = 0; v < V; v++){`
	`if (graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {`
	`parent[v] = u;`
	`key[v] = graph[u][v];`
	`}`
	`}`
		.2N+2
		. 2(assign, function return)
		.2(assign, access)
		.2N+2
		.9(6 access, 3 comparison)
		.2(access + assign)
		4(3 access+1 assign)
			.1
			.N
			.N
			.N
			.N*N
			. N*N
			N*N	

3	`printMST(outputFileName,parent, V, graph)`	F(n) printMST	1	Calling out to another function

F(N)primMST= 2N+2+2N+2N+2N+2+2N+2N+(2N+2)*N+9*N*N+2*N*N+4*N*N+F(N)printMST

F(N)= 2N+2+2N+2N+2N+2+2N+2N+2N*N+2N+9*N*N+2*N*N+4*N*N+5N

=21N+17N*N+ constant

Therefore, bigO of primMST is O(N^2)

<u>Method printMST</u>

Ln#	Function printMST	Cost	Times	comments
1	`out1 = new PrintWriter( new FileWriter(name));`	Constant cost	1	Initializing my writer
2	`for (int i = 1; i < V; i++) {`	2N+2	1	For loop
3	`out1.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);`	3	N	3 accesses

F(N) of my printMST= 2N+2+3N+constant= 5N+cst

Big-O of printMST= O(N)

Overall F(N)=F(N)construct+f(N)populate+f(N)writeToFIle+f(N)primMST+f(N)printMST=

N*N+9835N+1+8N*N+4N+3N*N+4N+21N+17N*N+5N=29N^2 +9852N+constant

The overall complexity of my code appeared to be quadratic of O(N^2)

References:

-CPSC 319 lecture slides

-https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

-Tutorial lecture slides: https://pages.cpsc.ucalgary.ca/~mdmamunur.rashid1/CPSC319-W19.html