Comp202 HW4.2 Report

Time and space complexity analysis for each function with pseudocodes:

getMin:

Pseudocode:

```
static int getMin(int array[],boolean isKnown[]){
  int min =10000;
  int index=-1;
  for(int i=0; i<array.length;i++){
    if (array[i]<min && isKnown[i]==false){
    min=array[i];
    index=i;
    }
  }
  return index;
  }
```

Time Complexity Analyisis:

There's one for loop which iterates over the input array and in the loop, operations are constant so time complexity is O(n) for n element array.

Space Complexity Analysis:

Nothing is created so space complexity is O(1).

totalTransitTime:

Pseudocode:

```
int sum=0;
  int adjMatrix[][] = graph.asArray(true);

  boolean isKnown[]= new boolean[adjMatrix[0].length];
//if i used PriorityQueue, it's impossible to acces by indexing and every time i want to access
an element i should search and its O(n) so i prefered array.

  int dist[]= new int[adjMatrix[0].length];
```

```
1 for(int k=0;k<adjMatrix[0].length;k++){                O(n^3)

    for(int i=0;i<adjMatrix[0].length;i++){
                                                O(n)
      dist[i]=10000;
    }

    for(int i=0;i<adjMatrix[0].length;i++){     O(n)
      isKnown[i]=false;
    }
                                                              O(n^2)
    dist[k]=0;
    int current=0;

2   for(int i=0;i<adjMatrix[0].length;i++){                 O(n)
    current=HW4.getMin(dist,isKnown);
    isKnown[current]=true;

3    for(int j=0;j<adjMatrix[0].length;j++){
       if(adjMatrix[current][j]!=0){
         if(isKnown[j]==false){                        O(1)
           if(dist[j]>dist[current]+adjMatrix[current][j]){
             dist[j]=dist[current]+adjMatrix[current][j];
           }
         }
       }
     }
    }

    for(int i=0;i<dist.length;i++){
       sum+=dist[i];
     }
}
                         return sum;
         }
```

Time Complexity Analysis:

The three for loops I indicated above determines the time complexity because others are operations like initializing Boolean array or distance array, the most outer for loop (1) iterates over all vertices of graph (n times), second for loop also iterates over all vertices (n times), in second loop, there's an operation (getMin()) which is O(n) and above that there's the third for loop which also iterates over all vertices (n times) and in that for loop there's nothing but constant operations so in second for loop, there's n + O(n) = O(n) operations (n from getMin(), O(n) from 3. for loop) and second loop iterates n times so until second for loop, time complexity is O(n^2), but there's still first for loop which includes second and third loop and iterates also n times overall time complexity is O(n^3).

Comparing To Floyd-Warshall:

Floyd-Warshall's algorithm's time complexity is O(n^3) but to fulfill the given task, it should be applied over all vertices of graph so overall time complexity would be O(n^4), in my implementation, its O(n^3).

Space Complexity Analysis:

There's been created an adjacency matrix(O(n^2)) + a Boolean array size of vertex number (O(n)) + the distance array size of vertex number(O(n)) = O(n^2) overall space complexity.

cheapestTransitTime:

Pseudocode:

```
int cheapestTransitTime(Graph graph) {

        return totalTransitTime(cheapestNetwork(graph));

    }
```

Time Complexity:

It's returning totalTransitTime O(n^3) with a graph that generated by my mst implementation from HW4.1(mst code is modified to give a graph but time complexity is still same) with cheapestNetwork method(O(n^2 + m*logm)) and they should be added because totalTransitTime() starts after cheapestNetwork() so overall time complexity is O(n^3) + O(n^2 + m*logm) = O(n^3).

Space Complexity:

cheapestNetwork() is a little more costy in terms of space complexity, because I added this lines:
Graph newGraph=new Graph();
int array_graph[][]= graph.asArray(false);
int latency_graph[][]= graph.asArray(true);

and they cost O(m+n) + O(n^2) + O(n^2) = O(n^2) and there's also a O(n^2) from totalTransitTime() function so it's O(n^2).

timeIncrease:

Pseudocode:

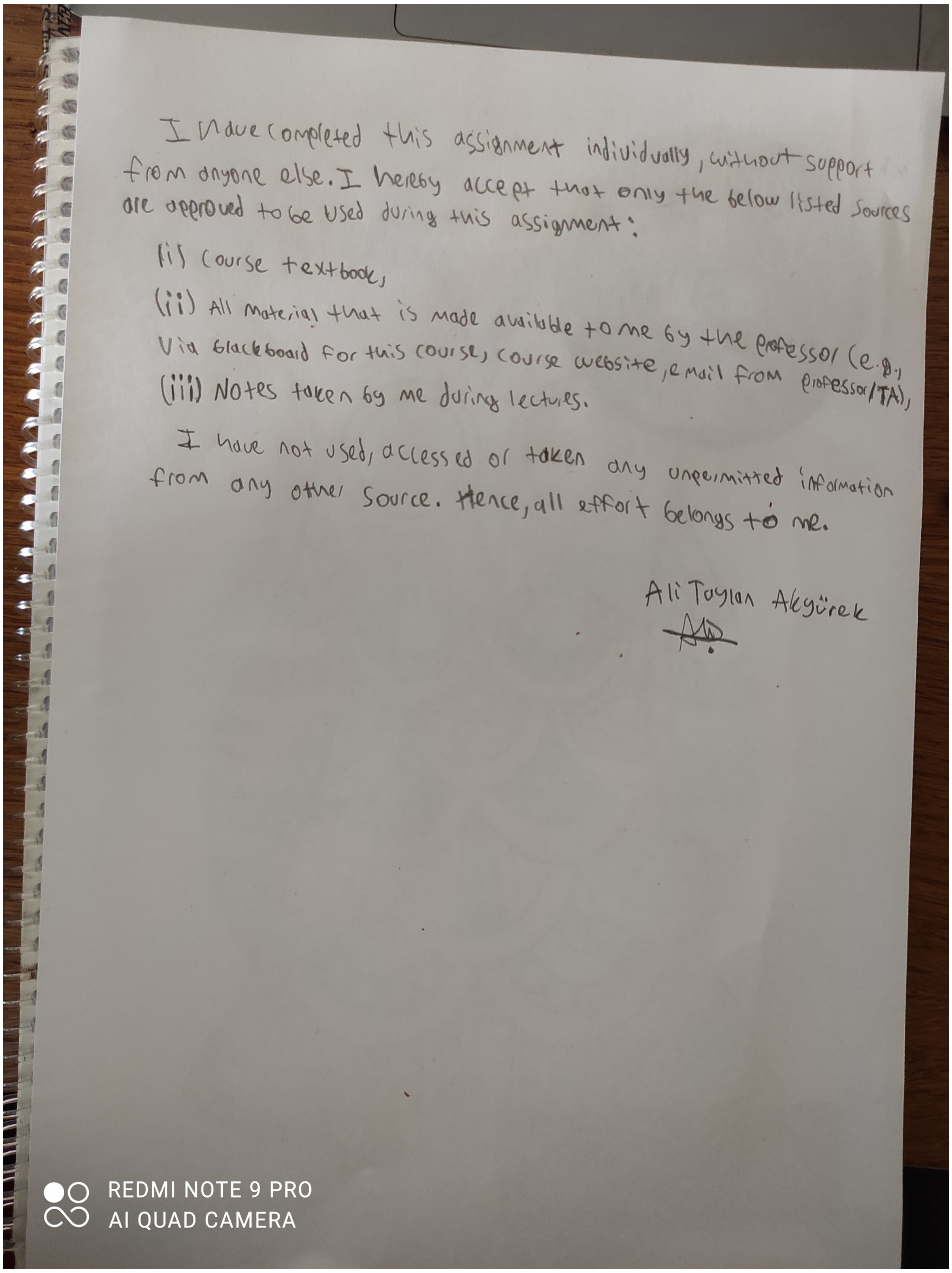
```
int timeIncrease(Graph graph) {

        return totalTransitTime(cheapestNetwork(graph))-totalTransitTime(graph);
    }
```

Time Complexity:

Both first and second functions are called so it’s $O(n^3) + O(n^3) = O(n^3)$.

Space Complexity:

Both first and second functions are called so it’s $O(n^2) + O(n^2) = O(n^2)$.



I have completed this assignment individually, without support from anyone else. I hereby accept that only the below listed sources are approved to be used during this assignment:

(i) Course textbook,

(ii) All material that is made available to me by the professor (e.g., via Blackboard for this course, course website, email from professor/TA),

(iii) Notes taken by me during lectures.

I have not used, accessed or taken any unpermitted information from any other source. Hence, all effort belongs to me.

Ali Taylan Akyürek

[illegible]