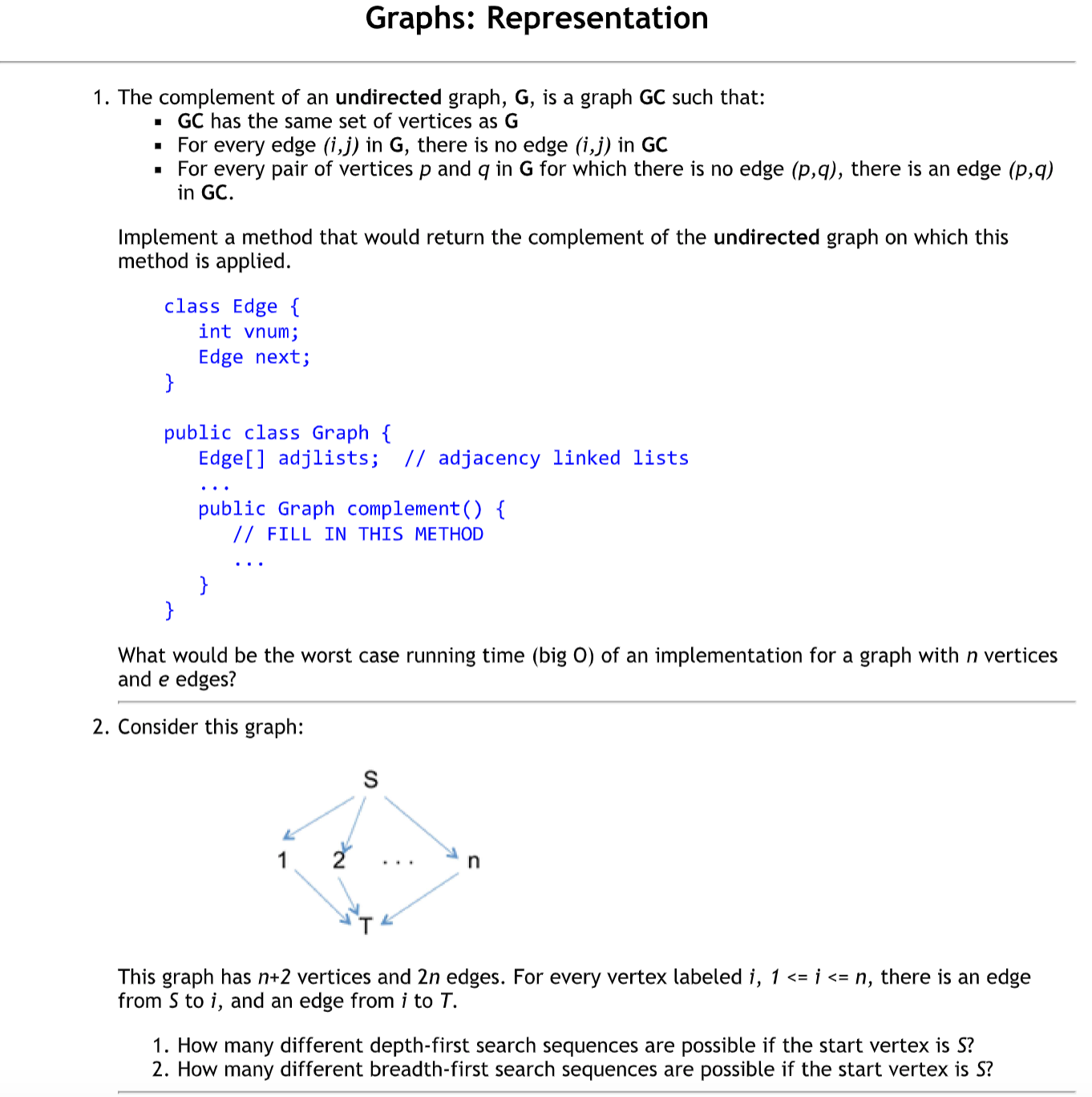
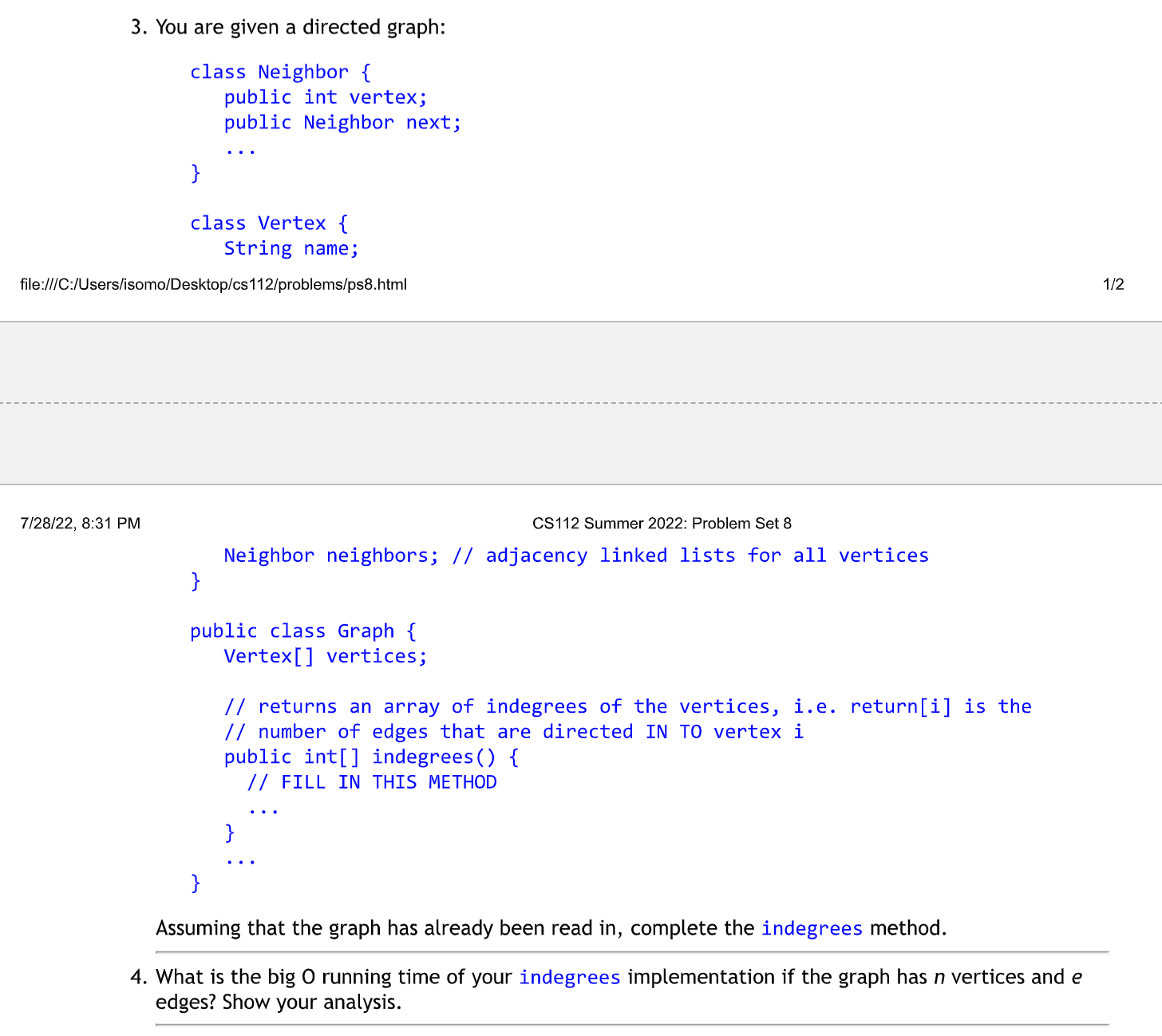
****

****

1.

public Graph complement() {

Graph complement = new Graph();

complement.adjlists = new Edge[this.adjlists.length];

boolean exists = false;

for (int i = 0; i < this.adjlists.length; i++) {

Edge current = this.adjlists[i];

while (current != null) {

exists = false;

for (int j = 0; j < adjlists.length; j++) {

if (this.adjlists[j].vnum == current.vnum) {

exists = true;

}

}

if (exists == false) {

Edge newEdge = new Edge();

newEdge.vnum = current.vnum;

Edge temp = complement.adjlists[i];

complement.adjlists[i] = newEdge;

newEdge.next = temp;

}

current = current.next;

}

}

return complement;

}

Worst case: O(n^2) as you would have to iterate through the vertices and the edges of each vertex.

2.

* Depth-first search sequences: n possible.
* Breadth-first search sequences: 1 possible.

3.

public int[] indegrees() {

int[] indeg = new int[this.vertices.length];

for (int i = 0; i < this.vertices.length; i++) {

Neighbor current = this.vertices[i].neighbors;

while (current != null) {

indeg[current.vertex]++;

current = current.next;

}

}

return indeg;

}

4.

The Big O of indegrees() would be O(n+e). While going through the front of a vertex's neighbors list, update the vertex’s indegree, and then access the neighbor of a vertex are counted as each unit time operation. Since there are “e” neighbors for all, including for all vertices, the neighbor access contributes to “e” units of time. By accessing the front of a vertex's neighbors list, this is done n times in total (once per vertex). As there are e indegree updates, one per edge exists. The total is e + n + e = n + 2e, which should result to O(n + e).