Lee

1. the worse case.

put N keys to 2N size.

the worse case is for linear probing is they have all same numbe

like we have N keys Hashing is 1.

that meaning. for the first one is 0.

second one is 1.

third is 2.

⋮

end second (n-2)

end one (n-1).

So the total numbe of comparison is

$= 0+1+2+3+4+5+6+\dots+(N-2)+(N-1)$

$= \frac{(0+N-1)\cdot N}{2} = \frac{(N-1)\cdot N}{2} = \frac{N^2-N}{2}$

the worse case need $\frac{N^2-N}{2}$ time.

this only happen when everyone key hashing is same.

```
import java.util.*;
class qu2 {
    private final int V;
    private final LinkedList<Integer> adj[];
    private String list="";
    qu2(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }
    void addEdge(int v,int w) {
        adj[v].add(w);
    }
    void BFS(int s) {
        boolean visited[] = new boolean[V];
        LinkedList<Integer> BFS_list = new LinkedList<Integer>();
        visited[s]=true;
        BFS_list.add(s);
        int buffer;
        while (BFS_list.size() != 0){
            buffer = BFS_list.poll();
            list=list+buffer+"-->";
            Iterator<Integer> adj_list = adj[buffer].listIterator();
            while (adj_list.hasNext()) {
                int next_int = adj_list.next();
                if (!visited[next_int]){
                    visited[next_int] = true;
                    BFS_list.add(next_int);
                }
            }
        }
        if (Objects.equals(list, s + "-->")){
            System.out.println("the graph of "+s+" is acyclic, So its girth is infinite.");
        }
        else System.out.println(list);
    }
    public static void main(String args[]) {
        Random r =new Random();
        qu2 g = new qu2(r.nextInt(100)+100);
        for (int i=0;i<100;i++){
            g.addEdge(r.nextInt(100),r.nextInt(100));
        }
        g.BFS(r.nextInt(10));
    }
}
```

3.↵
DFS algorithm runs in O(|V|+|E|)↵
At the first, we should know DFS have two choice one is directed another one is undirected, but them are same for big O theorem.↵
I use directed in my code.↵
V meaning vertices and E meaning edges.↵
We must check each one vertex, so the big O of V is O(|V|), and the for the edges is O(|E|), so the big of DFS algorithm is O(|V|+|E|)↵

```
import java.util.Iterator;
import java.util.Random;
import java.util.*;
public class qu3 {
        private final int V;
        private final LinkedList<Integer> adj[];
        private String list="";

        public qu3(int v){
            this.V = v;
            adj = new LinkedList[V];
            for (int i = 0; i < v; ++i)
                adj[i] = new LinkedList();
        }
        public void addEdge(int x,int y){
            adj[x].add(y);
        }
        public void DFS (int v){
            boolean check[]=new boolean[V];
            DFS_inside_check(v,check);
            System.out.println(list);
        }
        public void DFS_inside_check(int v,boolean[] check){
            check[v]=true;
            list=list+v+"-->";
            Iterator<Integer> list_of_tf = adj[v].listIterator();
            while(list_of_tf.hasNext()){
                int number= list_of_tf.next();
                if (!check[number]){
                    DFS_inside_check(number,check);
                }
            }
        }
        public static void main(String args[]){
            Random r =new Random();
            qu3 g = new qu3(r.nextInt(100)+100);
            for (int i=0;i<50;i++){
                g.addEdge(r.nextInt(100),r.nextInt(100));
            }
            g.DFS(1);
        }
    }
```

3. DFS's the algorithm runs in $O(|V|+|E|)$.

At the first for DFS we can understand two choice { directed / undirected

I use directed in my code.

Topological sorting

{ each one vertices show one time
from vertices all with less edges than behind V.

4. DAG (S, t).

① At the first, we need check S, t vertices both are no-input vertices.

② make a count set, for each one vertices have how many edges in come edges. ⇒ count[w]

③ Put S and t to a list name like S.

④ While S not empty do. {

Remove v vertices from S.
add vertices to an answers list.
for all edges of v to some w do.
{
count[w]--
if count[w]=0. then add w to answer list.
}
}

all outline logic is depend on check each one vertices input edges numbers. to check.

So the answers all with output the no input vertices first (S and t). and then will be the one input vertices, two..... to the n input vertices.

we need check each one vertices to that is $O(|V|)$
each one edges that is $O(|E|)$
add together will be $O(|V|+|E|)$