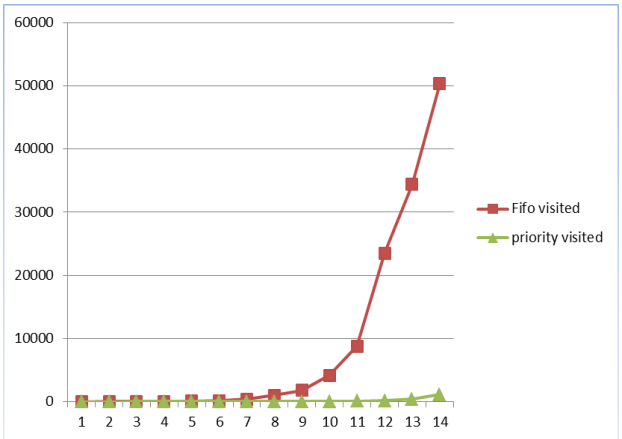
Supratim Baruah
Smb4

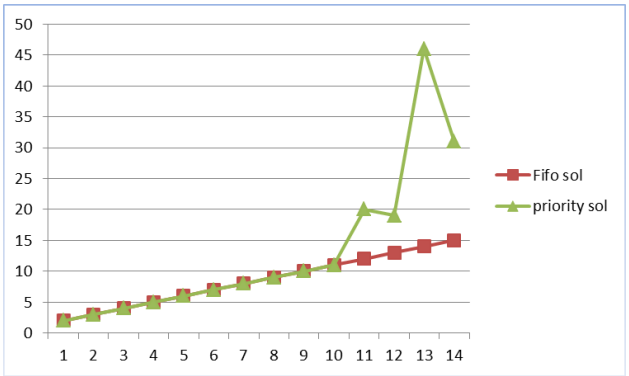
1) From the data collected for N moves from 1 to 14:
   FIFO uses far more space and time to calculate the solution as the N increases. Whereas priority Queue takes a lot less space and is faster.
   But as the N increases the priority queue solution path for the solution exceeds the solution path with FIFO.

I divides the graph in 3 graphs as FIFO visited node count was far greater than other data.

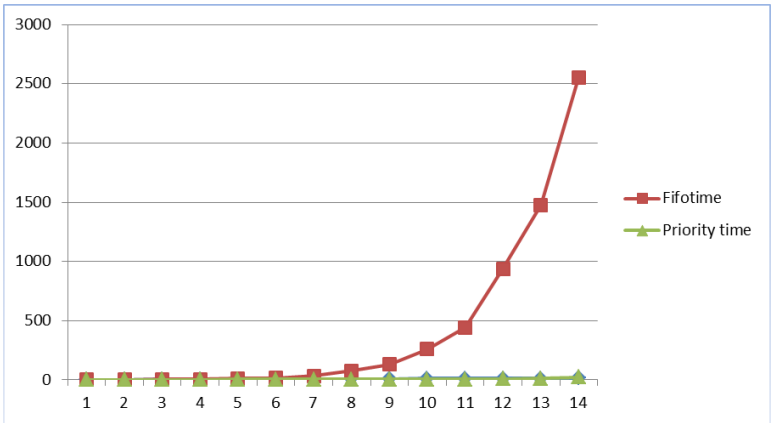
VISITED Count DATA:



SOLUTION count:



TIME TAKEN:



2) I have 5 classes:
   1. State : stores the puzzle .
   2. Node: stores the puzzle and its parent that calculated it as a neighbor
   3. BFS: takes the state and solves the puzzle
   4. Solver: takes the input for the puzzle
   5. HashTable: to store the visited states.

Supratim Baruah
Smb4

Random Puzzle Generator

```
public class PuzzleMaker {

	State goalState;
	Node goalNode;
	Node puzzle;
	Node lastNode;

	public PuzzleMaker(String[] goal)
	{
		goalState = new State(goal);
		goalNode = new Node(goalState, null);
	}

	public Node make(int moves)
	{
		//moves equals N
		Node cur = goalNode;
		lastNode = cur;
		boolean next = false;
		for(int i = 0; i < moves; i++) //loops until the puzzle takes N
steps
		{
			next = false;
			ArrayList<State> temp = cur.getCurState().neighbours();

			while(next != true) //runs until it finds the next move
			{
				int direction = getDirection(0, temp.size() - 1);

				Node tempNode = cur;
				State curState = temp.get(direction);
				tempNode = new Node(curState, cur);

				if(!lastState(tempNode)) //checks if it is not going
to undo last move
				{
					cur = tempNode;
					lastNode = cur.getParent();
					next = true;
				}
			}
		}
		puzzle = cur;
		return puzzle;

	}

	/**
	 * Helper function to make sure it is not going to Undo last move
	 * @param n
	 * @return true if it undos the last move
	 */
	public boolean lastState(Node n)
	{
```

```
		if(lastNode == null)
		{
			return false;
		}
		else if(lastNode.equals(n))
		{
			return true;
		}
		else
		{
			return false;
		}

	}

	/**
	 * Helper function to get the random direction it will move.
	 * @param min to get range
	 * @param max to get range
	 * @return direction
	 */
	public int getDirection(int min, int max)
	{
		int range = (max - min) + 1;
		int direction = (int) (Math.random()*range);
		return direction;
	}
}
```