**Search Analysis**

The available positions are n^2 – n [the same as the formula given] and this will be used as a way to determine the amount of steps some processes use. When we generate the available positions [before finding the heuristic value] one function will execute certain code by n^2 – n. What we have done is calculate all the formulas for each of the other classes for the preparation of the final analysis.

**Position Matching Section [algorithm to produce actions]:**

1. generateAvailablePositions

9n^2 + 9n + 1

1. generateTopPositions

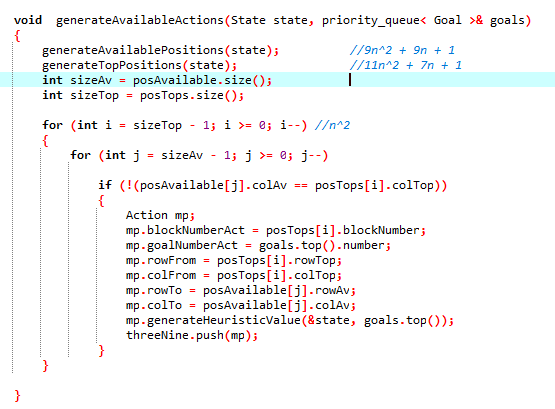
11n^2 + 7n + 1

1. Common Execution before

4n^2 +3n + 4

1. position matching

(n^2-n) \* (9n^2 + 9n + 40 + logn)

  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Total algorithm:**

9n^4 + 55n^2 + n^2logn – 20n - nlogn

**Algorithmic analysis of Total Complexity**

The algorithm itself is more complicated than giving it one stand alone mathematical function. There are three options that it can move through; and we are analyzing the most prominent part of the algorithm [the longest path]. At the same time, the actual formula for the algorithm will change from iteration to iteration, depending upon the circumstances [examples]:

1. The size of the containers
2. The particular condition of the state
3. The functions which is to be called
4. Whether the state has been used before [the amount used before]
5. The size of the used states
6. Whether a goal has been found
7. The amount of goals given
8. The difficulty of the goals to be achieved
9. The orientation of the initial state

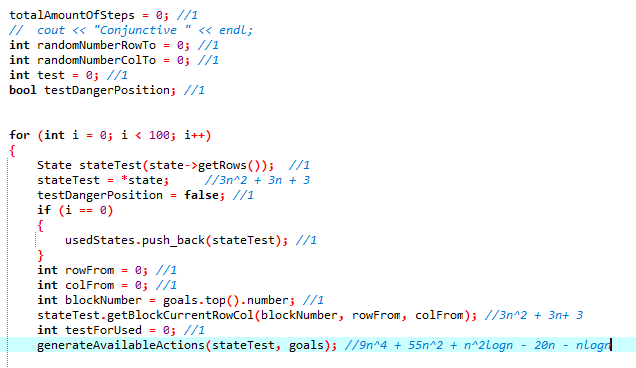
This suggests that there are probably 100 possible variations of the formula which could be analyzed. This can be demonstrated when we place a variable [variable called used] to determine how many times it has been called; the variation in the output of the variable is an indication that there isn’t one formula that can explain the algorithm (that there are many). Even though that this might be the case, for most situations, the algorithm itself will not be that much different from g(n); and we wouldn’t expect to see great variations in the iteration of the variable(used) also.

All these concepts are demonstrated below [three different cases target is bottom right to bottom left; example 1; move 5 to bottom left]:

|  |  |
| --- | --- |
| ***Initial State*** | **End State** |
|  |  |
|  |  |
|  |  |

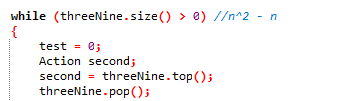
We note that we can see that when the circumstances are similar the output is much rather the same, however, when the circumstances are different, the number of steps used changes [indicating a variation in the formula]. In reality, there should be variation between the two states which have the same number used.   
  
Final Algorithm Analysis  
  
The main driver algorithm is split into four sections; which includes the section explained above [available positions]. All the formulas for the algorithms used in the state, action and goal class were calculated before executing this task.

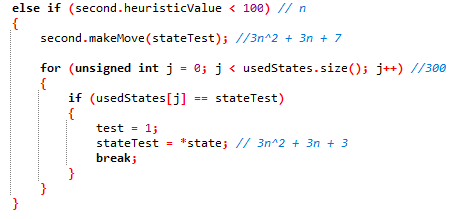
**1st section:**

****

9n^4 + 61n^2 – 15n + n^2logn – nlogn

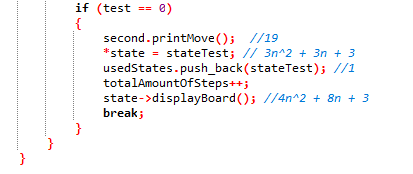
**2nd section:**

****

****

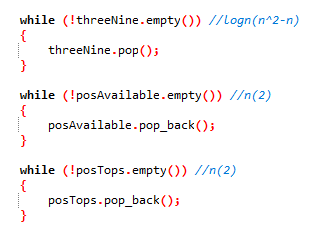
8n^4 + 307n^2 – 310n + n^2logn

**3rd section:**

****

7n^2 + 11n + 27

**4th section:**

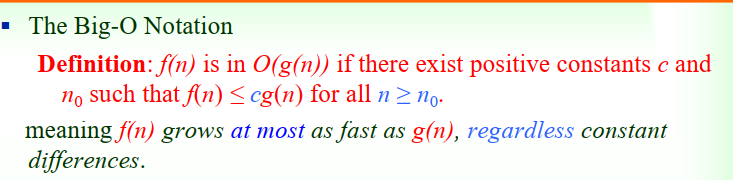


n^2logn – nlogn + 4n

**Final Algorithm [one iteration]:**

17n^4 + 3n^2logn + 375n^2 – 310n – 2nlogn + 27

**BigO analysis**

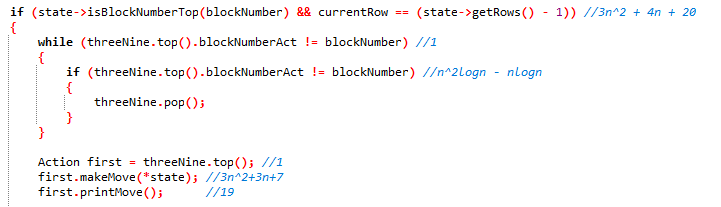
****

By definition of the BigO Notation; we can identify a constant when multiplied by g(n), would >= f(n); and thus the complexity of the algorithm should be bigO(n^4).

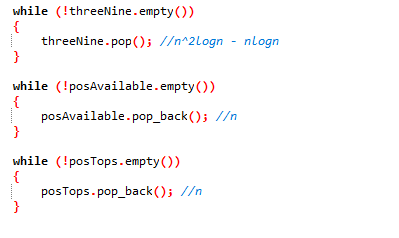
Algorithmic Switching

My particular algorithm switches to another algorithm if it identifies that the desired goal block number is at the bottom of the row. When it detects this, it will switch to another algorithm which will move the desired block number from the bottom row and fill it with some other value [through blocking]. The algorithm for this has also been analyzed and this is demonstrated in 3 sections.

**1st Section**

  
  
  
6n^2 + n^2logn + 7n – nlogn + 48

**2nd Section**  
  
n^2logn + 2n – nlogn



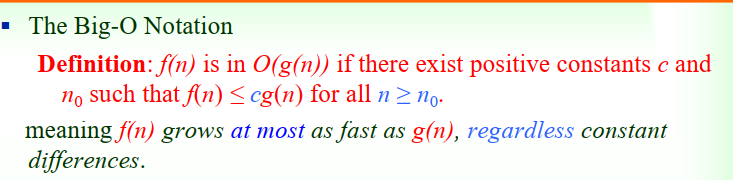
**3rd Section**

  
  
  
9n^4 + 69n^2 + 8n^2logn + 10n – 3nlogn + 35

**Total Complexity Algorithm**

9n^4 + 75n^2 + 10n^2logn -5nlogn + 19n + 83

**BigO analysis**



O(n^4)