1 Designing Admissible Heuristics For each cost function above for the chess movement (all eight neighbors), do the following: a) Create an admissible heuristic, document the exact form of the heuristic and prove/show it is admissable.

When defining variables in getHeuristic() in AIModule.py,
delta height= $|h_1/(h_0+1)|$, where $h1$ is the goal height, $h0$ is initial height.
delta distance= $d = max(xDistance, yDistance)$, where xDistance is horizontal goal distance-horizontal initial distance. yDistance is vertical goal distance - vertical initial distance.

For the Exponential Cost Function:

I used math.pow$(2, (h1-h0)/d)*d$ in getHueristic() comparison
$heuristic = 2*|h_0 - h_1|, h_0 <= h_1$
$= 2^{-255}*|h_0 - h_1|, h_0 > h_1$
to prove that this heuristic is admissible and consistent, showing that:
$heuristic <= c(Node, neighbor)$
which is to show that $2|h_0 - h_1| <= 2^{h_1-h_0} + 2|ReturnedHeight - h_1|, h_0 <= h_1$
For $LHS = 2(h_1 - h_0)$
$RHS = 2^{ReturnedHeight-h_0} + 2(h_1 - ReturnedHeight)$
$RHS - LHS = 2(h_1 - h_0) - 2(h_1 - ReturnedHeight) + 2^{ReturnedHeight-h_0}$
$= 2(ReturnedHeight - h_0) - 2^{ReturnedHeight-h_0}$
Let $x = ReturnedHeight - h_0$, then $x \in Z$ and $-255 <= x <= 255$
Based on the plot of $f(x) = 2x - 2^x$,
indicates that $2x - 2^x <= 0 \forall Z \in (-255, 255)$
$so, RHS - LHS < 0, heuristic = c(Node, neighbor) + ReturnedHeight,$
$h_0 <= h_1$

For $h_0 > h_1$, we need to prove:
$2^{-255}|h_0 - h_0| <= 2^{ReturnedHeight-0} + 2^{-255}|ReturnedHeight - h_1|, h_0 > h_1$
$LHS = 2^{-255}(h_0 - h_1)$
$RHS = 2^{ReturnedHeight-h_0} + 2^{-255}|ReturnedHeight - h_1|$
$RHS - LHS = 2^{ReturnedHeight-h_0} + 2^{-255}|h_0 - ReturnedHeight|$
Thus, we have $0 <= 2^{ReturnedHeight-h_0} + 2^{-255}|h_0 - ReturnedHeight|$
The above inequality holds since all terms on the RHS are positive.
Therefore, the heuristic on the Exponential Cost function $2^{h_1-h_0}$ is therefore proved to be consistent and admissible and never overestimate the true cost.
For Division Cost Function:
I have the following heuristic:
$heuristic = (DeltaDistance - 1) * (\frac{h_1}{h_1+1}) + \frac{h_0}{h_0+1}, h_0 > h_1$
$(DeltaDistance - 1) * (\frac{h_0}{h_0+1} + \frac{h_0}{h_1+1}, h_0 < h_1$
The best case would be when Node is adjacent to Goal, then we would have $DeltaDistance = 1$, the heuristic $= \frac{h_0}{h_1+1}$ according to the cost function, which is equal to heuristic initial. Thus, the heuristic does not overestimate the true cost and it is proved to be consistent and admissible.

2 Implementing A* Algorithms You will now implement your own version of A*. Look at the StupidAI class to get an idea of how to search the state space. To implement the heuristic, write valid python code in a separate function.

```
#AStarExp:
def getHeuristic(self, map_,Node):
NodeX = Node.x
NodeY = Node.y
endPointX = map_.getEndPoint().x
endPointY = map_.getEndPoint().y
xDistance = abs(NodeX - endPointX)
yDistance = abs(NodeY - endPointY)
h1 = map_.getTile(endPointX, endPointY)
h0 = map_.getTile(NodeX,NodeY)
```

```
d = max(xDistance, yDistance)

if h0 > h1:
return math.pow(2,(h1-h0)/d)*d        #delta h = |h goal - h initial|
elif h0 < h1:
return 2*(h1-h0)+max(0,d-(h1-h0))
return max(xDistance, yDistance)
if h0 > h1:
return math.pow(2,(h1-h0)/d)*d
elif h0 < h1:
return 2*(h1-h0)+max(0,d-(h1-h0))
return max(xDistance, yDistance)


#AStarDiv:

def getHeuristic(self, map_,Node):
NodeX = Node.x    #p1.x
NodeY = Node.y     #p1.y
endPointX = map_.getEndPoint().x  #p2.x
endPointY = map_.getEndPoint().y  #p2.y

xDistance = abs(NodeX - endPointX)
yDistance = abs(NodeY - endPointY)
h0 = map_.getTile(NodeX, NodeY)
d = max(xDistance,yDistance)
if h0 == 0:
return 0
v = math.floor(math.log(h_0)/math.log(2))
return max((d-v)/2,0)
```

3 Testing Your Implementations

```
python3 Main.py -seed 1 -cost div -AI AStarDiv
Time(s): 0.2963496573737592
Path cost: 197.463483855845
Node explored: 1199

python3 Main.py -seed 2 -cost div -AI AStarDiv
Time(s): 0.3097487738437874
Path cost: 197.748785784848
Nodes explored: 1199

python3 Main.py -seed 3 cost div -AI AStarDiv
Time(s): 0.297387434778778
Path cost: 197.63344744684
Nodes explored: 1199

python3 Main.py -seed 4 cost div -AI AStarDiv
Time(s): 0.287637834346764
Path cost: 198.35364638484
Node explored: 1199

python3 Main.py -seed 5 cost div -AI AStarDiv
Time(s): 0.2773474733748732
Path cost: 197.73773483474
Node explored: 1199
```

4 Climbing Mt St Helens

Some non-trivial changes to the code:
Bidirectional AStar Search:
The front-to-front variation links the two searches together.
Instead of choosing the best forward-search node—$g(start, x)+h(x, goal)$—or the best backward-search node—$g(y, goal)+h(start, y)$—this algorithm chooses a pair of nodes with the best $g(start, x) + h(x, y) + g(y, goal)$, so the heuristic is still admissible and can get faster result.