Names: Alejandro Ramos(alejandroramosh27@csu.fullerton.edu)
Alexander Zavaleta(A_zavaleta@csu.fullerton.edu)

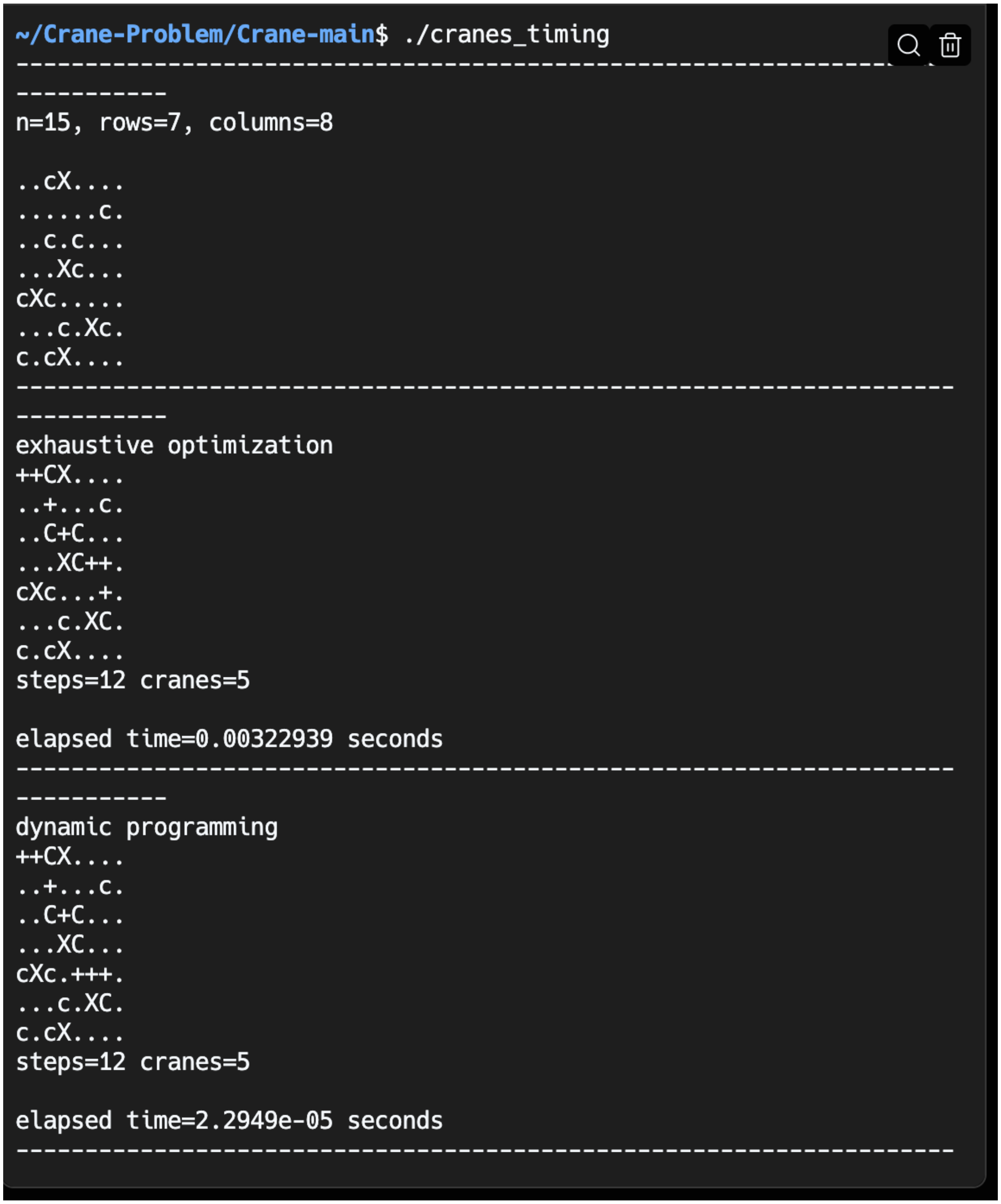
I. Create a Document with the following:

Exhaustive Algorithm Solution Pseudocode

```
for size in range max_steps:
    for i in range i < (1 << steps):
        if( i < (1 << steps)):
        path candidate(setting)
        for j in range steps:
            if(candidate is valid to the east):
                move east
            end if
            else:
            break
            end else
        end if
        else:
            //move south
            if(candidate is valid to the SOUTH):
                move south
            end if:
            else:
                break
            end else:
        end else:
        end j:
        if total_cranes() > best.total_cranes:
            best = candidate
        end if;
    close for loop i;
close steps loop
```

Time complexity function of Exhaustive Algorithm

We know that the Exhaustive program run time is 0(2^(N +M) + O(N)). The N and the M are the rows and columns that we use to traverse the paths. Also, this takes up more time because it finds all the possible paths so it takes a while.
When we run ./cranes_timing. We are given the inputs 7 = rows and 8 = columns.
So if we paste the input into the run time we get 0(2^(7+8) + 0(7))



```
~/Crane-Problem/Crane-main$ ./cranes_timing
-----------------------------------------------------------------------
-----------
n=15, rows=7, columns=8

..cX....
......c.
..c.c...
...Xc...
cXc.....
...c.Xc.
c.cX....
-----------------------------------------------------------------------
-----------
exhaustive optimization
++CX....
..+...c.
..C+C...
...XC++.
cXc...+.
...c.XC.
c.cX....
steps=12 cranes=5

elapsed time=0.00322939 seconds
-----------------------------------------------------------------------
-----------
dynamic programming
++CX....
..+...c.
..C+C...
...XC...
cXc.+++.
...c.XC.
c.cX....
steps=12 cranes=5

elapsed time=2.2949e-05 seconds
-----------------------------------------------------------------------
```

Dynamic Algorithm Solution Pseudocode:

```
if(column > 0 and (r,c-1) != cell building
        Left = A[r][c-1]
```

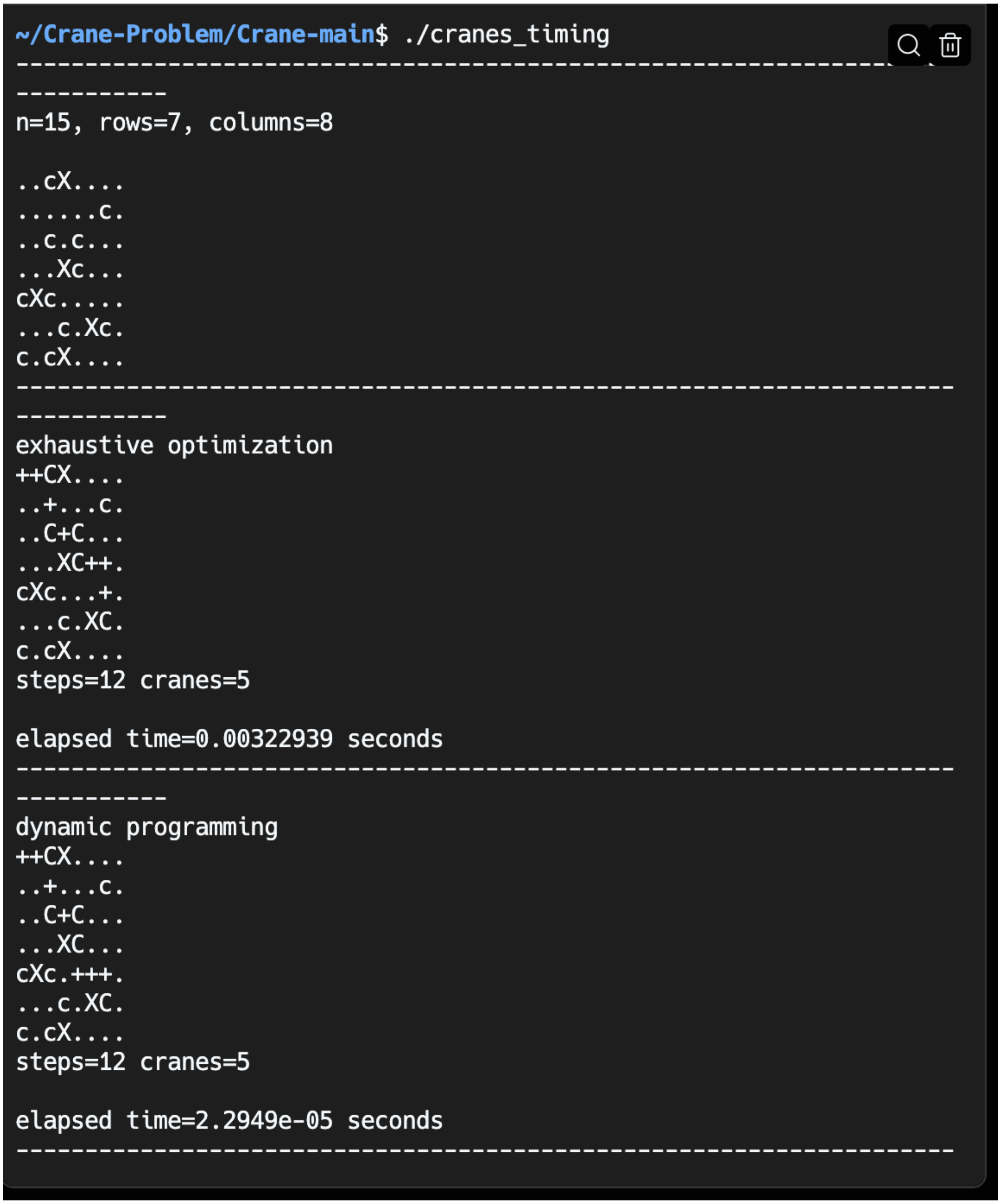
```
    if(left has value)
        add step(STEP_DIRECTION_EAST)
    end if
end if
If (row > 0 and get(r-1,c) != cell building)
    Above = A[r-1][c]
    If (above has value)
        add step(STEP_DIRECTION_SOUTH)
    End if
End if
if(from above and left have no value):
    A[r][c].reset();
    continue
end if
if(above has value && left has no value)
    A[r][c] = above
End if
else if (above has no value && left has value)
    A[r][c] = left
end if
else if (above has value && left has value && above.crane > left.crane)
    A[r][c] = above
end if
Else
    A[r][c] = left

for coordinate r in range setting.row()
    for coordinate c in range setting.columns()
        if(A[r][c].has_value() && A[r][c].value().total_cranes() > best.total_crane
            Best = A[r][c].value
        end if
    end for
end for
return best;
```

Time analysis

We know that the Dynamic Algorithm program run time is 0(3^(N)).

When we run ./cranes_timing. We are given the inputs n = 15 which is our rows and columns. In this problem, the reason why this would be O(n^3) is that we need to traverse the rows and columns. We then need to traverse the graph again and check to see if first, the spot we are currently on has a crane, and if it does it compares it to the best. crane and if it’s more, it will insert it into “best”.
So if we paste the input into the run time we get 0(3^(15)).



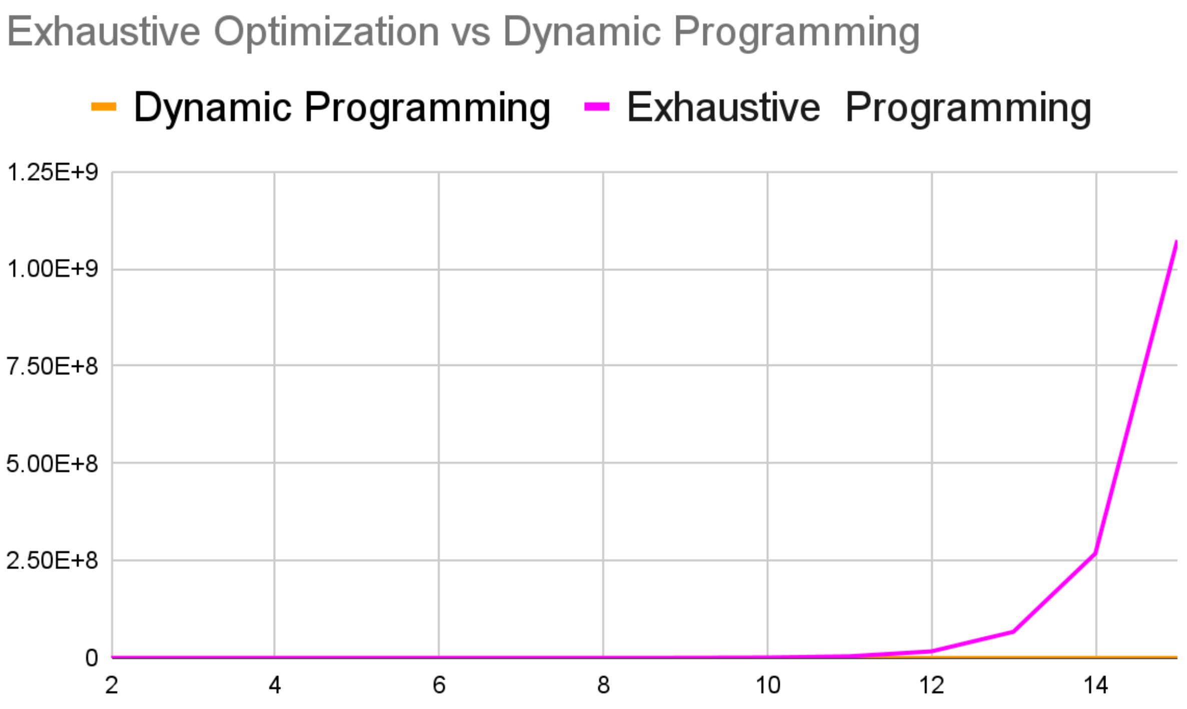
```
~/Crane-Problem/Crane-main$ ./cranes_timing
----------------------------------------------------------------------
-----------
n=15, rows=7, columns=8

..cX....
......c.
..c.c...
...Xc...
cXc.....
...c.Xc.
c.cX....
----------------------------------------------------------------------
-----------
exhaustive optimization
++CX....
..+...c.
..C+C...
...XC++.
cXc...+.
...c.XC.
c.cX....
steps=12 cranes=5

elapsed time=0.00322939 seconds
----------------------------------------------------------------------
-----------
dynamic programming
++CX....
..+...c.
..C+C...
...XC...
cXc.+++.
...c.XC.
c.cX....
steps=12 cranes=5

elapsed time=2.2949e-05 seconds
----------------------------------------------------------------------
```

Exhaustive Optimization vs Dynamic Programming Graph



1. Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?

   Yes, there is a noticeable difference between exhaustive optimization and dynamic programming. The dynamic programming program is faster than the optimization by approximately 0.00320644 seconds. Ya, it did surprise us by how fast it was, I was astonished.

2. Are your empirical analyses consistent with your mathematical analyses? Justify your answer

   From what I see from the empirical analyses, it seems that dynamic programming is much faster than exhaustive optimization. From the graph up above we see that around 12 inputs exhaustive programming begins to increase exponentially in time, while dynamic programming relatively remains the same. Making empirical analyses consistent with the mathematical analyses.

3. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.

Yes, by the evidence that we have seen, dynamic programming is much faster than exhaustive optimization. You can see this in the picture we have displayed at the top. It shows the time and as you can see, exhaustive optimization takes a lot more time than dynamic programming.

4. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.

   It is inconsistent because, from the images and the information we have, exhaustive optimization is much slower than dynamic optimization. Also, dynamic programming is much more efficient as well.

Video of code:

https://drive.google.com/file/d/13h17k9WeDTPX8pFX1ZFmwyifKbkH34qV/view?usp=sharing