# CS325 - Project 2

Group #6
William Jernigan, Alexander Merrill, Sean Rettig

October 28, 2014

# Correctness

## Proof of Claim 3: Direct Proof

Claim 3: If $\{z_1, z_2, ..., z_t\}$ and $\{z'_1, z'_2, ..., z'_s\}$ are two visible sets of lines (each ordered by increasing slope), then the visible subset of $\{z_1, z_2, ..., z_t\} \cup \{z'_1, z'_2, ..., z'_s\}$ is $\{z_1, ..., z_i\} \cup \{z'_j, ..., z'_s\}$ for some $i \geq 1$ and $j \leq s$.

Let $A = \{a_1, a_2, ..., a_t\}$ be the set $\{z_1, z_2, ..., z_t\}$ and let $B = \{b_1, b_2, ..., b_s\}$ be the set $\{z'_1, z'_2, ..., z'_s\}$ for improved clarity.

### Prove that $\{a_1, ..., a_i\}$ is visible:

Because all elements in A were defined to be visible with respect to each other, any covering line $b_q$ must be from B.
Given that $m_{a_t} < m_{b_1}$, a covered line $a_p \in A$ has $m_p < m_q$.

#### Prove that for any covered line $a_p$, all lines to the right of p in A are also covered:

Because $a_p$ is defined to be invisible, then by Claim 2 in the P1 Visible Lines Handout:
$y_{a_{p-1}}(x_{a_{p-1},b_q}) > y_{a_p}(x_{a_{p-1},b_q})$, where $x_{a_{p-1},b_q}$ is the x coordinate where $a_{p-1}$ and $b_q$ intersect.
Because $a_{p+1}$ is defined to not cover $a_p$, we know that $y_{a_p}(x_{a_{p-1},b_q}) > y_{a_{p+1}}(x_{a_{p-1},b_q})$.
For $x < x_{a_{p-1},a_p}$, $a_{p+1}$ is covered by $a_p$.
We need to show that $a_{p+1}$ is covered for $x \geq x_{a_{p-1},a_p}$.
Because $b_q$ is covering $a_p$, $y_{b_q}(x) > y_{a_q}(x)$ for $x \geq x_{a_{p-1},b_q} \geq x_{a_{p-1},a_p}$.
$\therefore p + 1$ is invisible if p is invisible.
This follows for all p.
Then let $p = i + 1$.
$\therefore \{a_1, ..., a_i\}$ is visible and $\{a_p, ..., a_t\}$ is invisible.

### Prove that $\{b_j, ..., b_s\}$ is visible

Because all elements in B were defined to be visible with respect to each other, any covering line $a_o$ must be from A.
Given that $m_{a_t} < m_{b_1}$, a covered line $b_r \in B$ has $m_o < m_r$.

#### Prove that for any covered line $b_r, r = s - 1$, all lines to the left of r in B are also covered:

Because $a_p$ is defined to be invisible, then by Claim 2 in the P1 Visible Lines Handout:
$y_{b_{r+1}}(x_{b_{r+1},a_o}) > y_{b_r}(x_{b_{r+1},a_o})$, where $x_{b_{r+1},a_o}$ is the x coordinate where $b_{r+1}$ and $a_o$ intersect.

Because $b_{r-1}$ is defined to not cover $b_r$, we know that $y_{b_r}(x_{b_{r+1},a_o}) > y_{b_{r-1}}(x_{b_{r+1},a_o})$.
For $x < x_{b_{r+1},b_r}$, $b_{r-1}$ is covered by $b_r$.
We need to show that $b_{r-1}$ is covered for $x \geq x_{b_{r+1},b_r}$.
Because $a_q$ is covering $b_r$, $y_{a_o}(x) > y_{b_o}(x)$ for $x \geq x_{b_{r+1},a_o} \geq x_{b_{r+1},b_r}$.
$\therefore$ $r-1$ is invisible if r is invisible.
This follows for all r.
Then let $r = j-1$.
$\therefore$ $\{b_j, ..., b_s\}$ is visible and $\{b_1, ..., b_r\}$ is invisible.

## Proof of MergeVisible: Direct Proof

### Prove that for each line that MergeVisible checks, it correctly determines the visibility of the line:

For each line at $i$ that MergeVisible checks, it chooses $j, k$ with $j < i < k$ and marks the line as invisible if $y_j(x_{j,k}) > y_i(x_{j,k})$.
Therefore, by the proof of Claims 2 and 3 in the "Visible Line Notes" handout, each line it checks and marks as invisible is actually invisible.
Additionally, for each line that MergeVisible checks the visibility of, it also checks to make sure that the previous line was not covered by the current line (using the same check), thereby exhausting all possible combinations of $j, k$.
Therefore, there is no $i$ such that $j < i < k$ and $y_j(x_{j,k}) > y_i(x_{j,k})$.
Therefore, by the proof of Claims 2 and 3 in the "Visible Line Notes" handout, every line marked as visible is actually visible.

### Prove that each line in the rest of each set is also invisible if an invisible line is found:

By the proof of Claim 3, we know that if a line in $A$ with index $p$ is invisible, then all lines in $A$ with index $> p$ are also invisible.
By the proof of Claim 3, we know that if a line in $B$ with index $r$ is invisible, then all lines in $B$ with index $< r$ are also invisible.

## Proof of Algorithm 4: Proof by Induction

Let $Y$ be a list of $n$ lines sorted in ascending order by slope. We claim that Algorithm 4 will return the visible subset of $Y$.

### Base case:

For any set of size $< 2$, we know that each line in the set is trivially visible because no other lines exist to cover the single line in the set.

### Inductive hypothesis:

By our proof of MergeVisible, we know that if MergeVisible is passed two sets of visible lines, that it will correctly return the merged set of visible lines.

### Applying the axiom of induction:

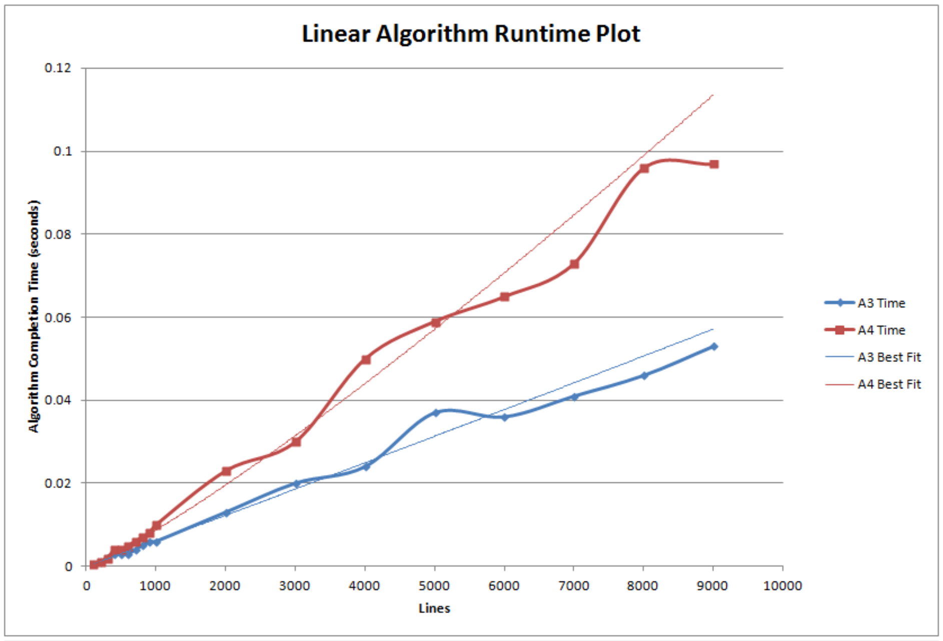
Because Algorithm 4 subdivides the given set of lines $Y$ into sets of size 1 (which are visible according to the base case) before passing them to MergeVisible, we know that the result of each first MergeVisible call

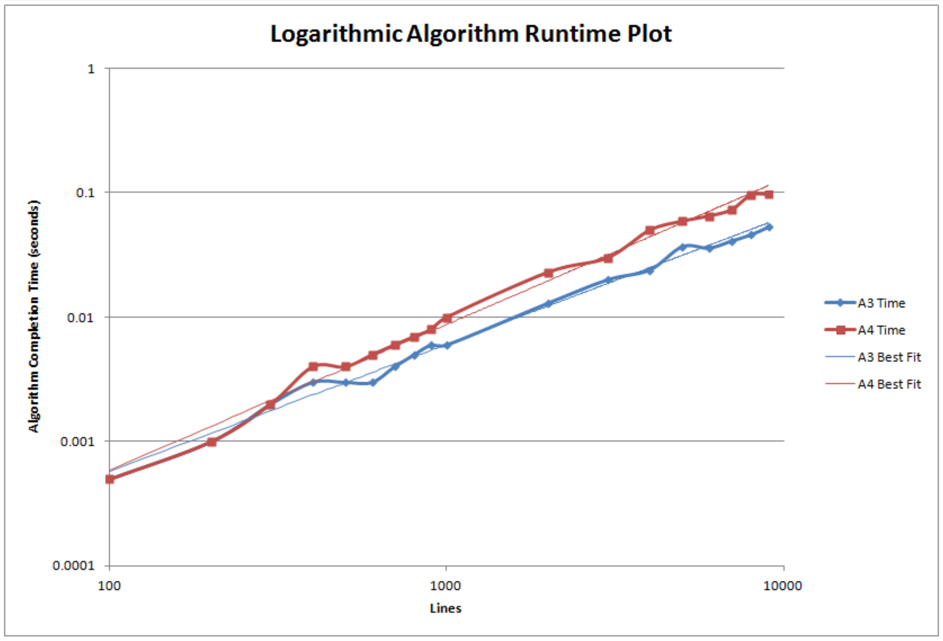
is correct.
Because Algorithm 4 then only passes the results of correct previous MergeVisible calls to MergeVisible, each subsequent call of MergeVisible also returns a correct result.
Therefore, once all subsets of $Y$ have been merged using MergeVisible, we know that the result of the final MergeVisible call is correct.
Therefore, because Algorithm 4 returns this result, we know that Algorithm 4 correctly returns the set of visible lines given any set of lines $Y$.

## Experimental and Asymptotic Run Time Analysis

### Experimental Run Time Data

## Experimental Run Time Plots





## Experimental Run Time Analysis

Algorithm 3: $y = 5 \times 10^{-6} x^{1.0234}$
Algorithm 4: $y = 3 \times 10^{-6} x^{1.1695}$

Given these equations, we can use both the slopes and our code to analyze the run times of the algorithms. We can also determine the biggest instance that can be solved with each algorithm in an hour.

## Asymptotic Run Time Analysis

Algorithm 1: $\Theta(n^3)$
Algorithm 2: $\bigcirc(n^3)$
Algorithm 3: $\bigcirc(n^2), \Omega(n)$
Algorithm 4: $\bigcirc(nlogn), \Omega(logn)$
Algorithm 4

## Extrapolation and Interpretation

### Estimated Number of Lines Per Hour

Algorithm 3: $y = 5 \times 10^{-6} x^{1.0234}$
Algorithm 4: $y = 3 \times 10^{-6} x^{1.1695}$

#### Discrepancies

We note that the slopes from our experimentally-derived equations are within the asymptotic run-time range of $\bigcirc(n^2)$, $\Omega(n)$ and $\bigcirc(nlogn)$, $\Omega(logn)$ however Algorithm 4, which is theoretically faster than Algorithm 3 actually runs slower. This discrepancy many have many possible contributing factors, including a small sample size, a low timing resolution (particularly for Algorithm 3), how the compiler and system handle arrays and operations, and randomness in run-time present. This randomness in run-time is due to both Algorithms not performing a fixed number of operations, but rather change what operations they run depending on the input lines given. The randomness and how the compiler and system handle the our code are likely the primary causes of these differences.

## Pseudocode

```
algorithm1(lines):
    for j in lines[0 ... n]:
        for i in lines[j+1 ... n]:
            for k in lines[i+1 ... n]:
                Xjk, Yjk = intersection(j, k)
                Yi = i.slope * Xjk + i.intercept
                if Yjk > Yi:
                    i.visible = False
    return lines

algorithm2(lines):
    for j in lines[0 ... n]:
        for i in lines[j+1 ... n]:
            for k in lines[i+1 ... n]:
                if i.visible:
                    Xjk, Yjk = intersection(j, k)
                    Yi = i.slope * Xjk + i.intercept
                    if Yjk > Yi:
                        i.visible = False
    return lines

algorithm3(lines):
    vlines = []
    for i in lines:
        vlines.append(i)
        removeCovered(vlines)
    return lines

algorithm4(lines):
    if len(lines) <= 1:
        return lines
    else:
        left = algorithm4(first half of lines)
        right = algorithm4(second half of lines)
    merged = mergeVisible(left, right)
    return merged

def mergeVisible(a, b):

    # Don’t check the ends for visibility because they are always visible.
    i = 1
    j = len(b)-2

    while checking A’s or checking B’s:
        if checking A’s:

            # a[i] is the next line in A.
            # b[j] is the next line in B.
            # a[i-1] is the previous line in A.
            # b[j+1] is the previous line in B.

            # Check the next line in A.
            x*, y* = intersection(a[i-1], b[j+1])
            testLineY = a[i].slope * x + a[i].intercept
            if y* > testLineY:
                stop checking A's (because the rest of them are invisible)
            else:
                i += 1 # Get ready to check the next line in A on the next iteration

                # Now we check to make sure that we didn't just cover the last line in 'b'.
                if there is a previously added line in j:
                    intersectionY = a[i-1].slope * (a[i-1].intercept - b[j+2].intercept) + a[i-1].inter
                    testLineY = b[j+1].slope * (a[i-1].intercept - b[j+2].intercept) + b[j+1].intercept
                    if intersectionY > testLineY:
                        checkBs = False
                        j += 1

        if checkBs and j >= 0:
            intersectionY = a[i-1].slope * (a[i-1].intercept - b[j+1].intercept) + a[i-1].intercept * (
            testLineY = b[j].slope * (a[i-1].intercept - b[j+1].intercept) + b[j].intercept * (b[j+1].s
            if intersectionY > testLineY:
                checkBs = False
            else:
                j -= 1

                # Now we check to make sure that we didn't just cover the last line in 'a'.
                intersectionY = a[i-2].slope * (a[i-2].intercept - b[j+1].intercept) + a[i-2].intercept
                testLineY = a[i-1].slope * (a[i-2].intercept - b[j+1].intercept) + a[i-1].intercept * (
                if intersectionY > testLineY:
                    checkAs = False
                    i -= 1

    vlines = a[:i] + b[j+1:]
    return vlines
```