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 $\{a_1, ..., a_i\}$ are 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 $m_{a_p} < m_{b_q}$.

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

Because a_{p+1} is defined to not cover a_p, we know that $y_{a_p}(x_{a_{p-1},a_p}) > y_{a_{p+1}}(x_{a_{p-1},a_p})$.
Therefore, for $x \leq x_{a_{p-1},a_p}$, a_{p+1} is covered by a_p because $m_{a_p} < m_{a_{p+1}}$.
We now need to show that a_{p+1} is covered for $x > x_{a_{p-1},a_p}$.
Because b_q is covering a_p, $y_{b_q}(x) > y_{a_p}(x)$ for $x \geq x_{a_{p-1},a_p}$.
This means that $y_{b_q}(x_{a_{p-1},a_p}) > y_{a_p}(x_{a_{p-1},a_p}) > y_{a_{p+1}}(x_{a_{p-1},a_p})$. In other words, b_q covers both a_p and a_{p+1} where a_p and a_{p-1} intersect.
Therefore, for $x > x_{a_{p-1},a_p}$, a_{p+1} is covered by b_q because $m_{b_q} > m_{a_{p+1}}$.
$\therefore p + 1$ is covered for all x.
$\therefore p + 1$ is invisible if p is invisible.
This follows for all p.
Then let $p = i + 1$.
$\therefore \{a_1, ..., a_i\}$ are visible and $\{a_p, ..., a_t\}$ are invisible.

Prove that $\{b_j, ..., b_s\}$ are visible

A similar proof can be provided for the lines in B.
Because all elements in B were defined to be visible with respect to each other, any covering line a_q must be from A.
Given that $m_{a_t} < m_{b_1}$, a covered line $b_p \in B$ has $m_{a_q} < m_{b_p}$.

Prove that for any covered line b_p, all lines to the left of p in B are also covered:

Because b_{p-1} is defined to not cover b_p, we know that $y_{b_p}(x_{b_{p+1},b_p}) > y_{b_{p-1}}(x_{b_{p+1},b_p})$.
Therefore, for $x \leq x_{b_{p+1},b_p}, b_{p-1}$ is covered by b_p because $m_{b_p} < m_{b_{p-1}}$.
We now need to show that b_{p-1} is covered for $x > x_{b_{p+1},b_p}$.
Because a_q is covering b_p, $y_{a_q}(x) > y_{b_p}(x)$ for $x \geq x_{b_{p+1},b_p}$.
This means that $y_{a_q}(x_{b_{p+1},b_p}) > y_{b_p}(x_{b_{p+1},b_p}) > y_{b_{p-1}}(x_{b_{p+1},b_p})$. In other words, a_q covers both b_p and b_{p-1} where b_p and b_{p+1} intersect.
Therefore, for $x > x_{b_{p+1},b_p}, b_{p-1}$ is covered by a_q because $m_{a_q} > m_{b_{p-1}}$.
$\therefore p-1$ is covered for all x.
$\therefore p-1$ is invisible if p is invisible.
This follows for all p.
Then let $p = j - 1$.
$\therefore \{b_1, ..., b_i\}$ are visible and $\{b_p, ..., b_t\}$ are 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:

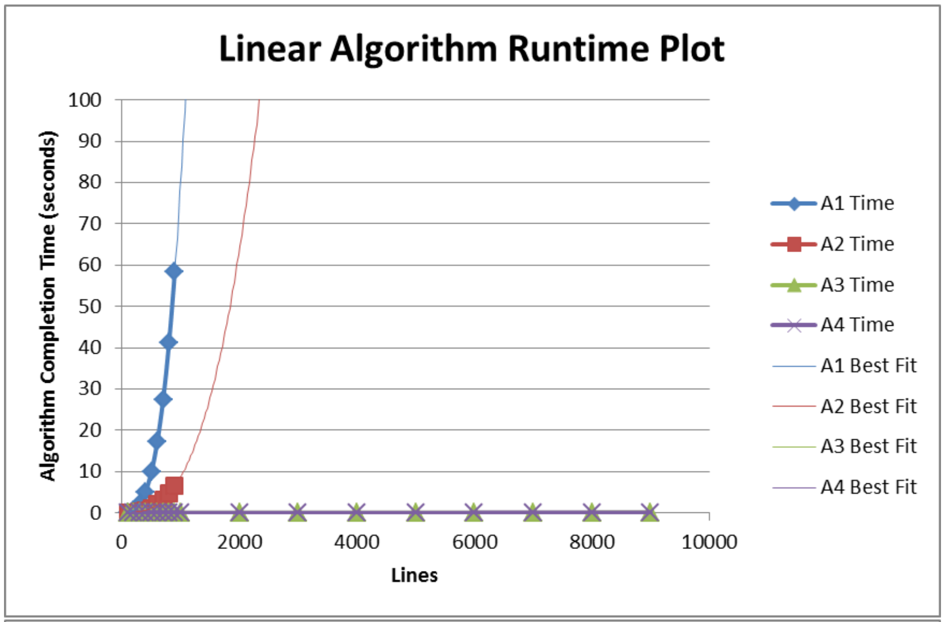
Because Algorithm 4 subdivides the given set of lines Y into sets of size 1 (which are visible according to the base case) before passing them to MergeVisible, we know that the result of each first MergeVisible call is correct.
Because Algorithm 4 then only passes the results of correct previous MergeVisible 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.

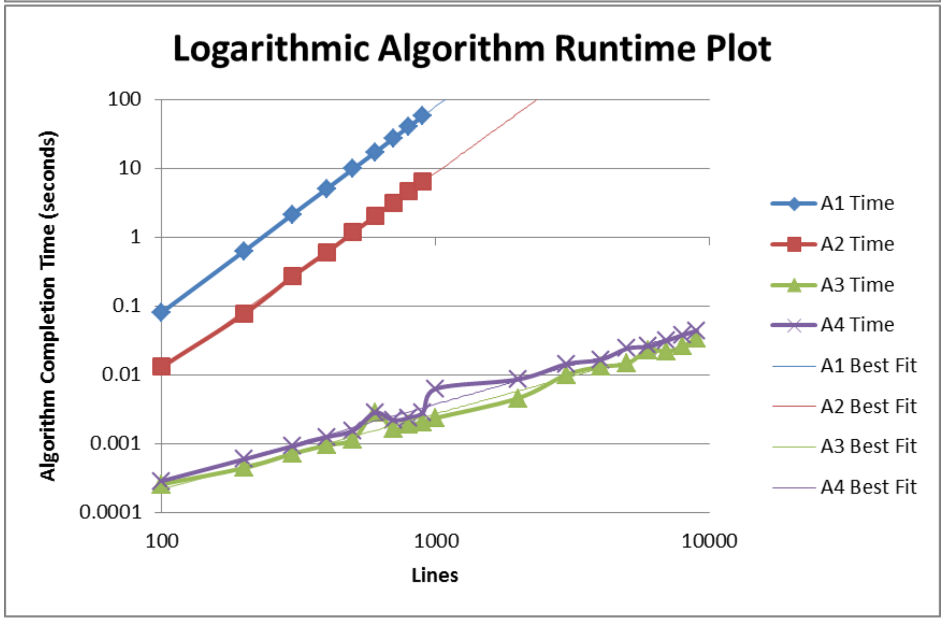
Experimental and Asymptotic Run Time Analysis

Experimental Run Time Data

Lines	A1 Time	A2 Time	A3 Time	A4 Time
100	0.08167	0.01316	0.00026	0.00029
200	0.63689	0.07770	0.00045	0.00060
300	2.16108	0.27030	0.00072	0.00093
400	5.07531	0.60974	0.00096	0.00126
500	9.93557	1.18310	0.00117	0.00157
600	17.1548	2.03575	0.00282	0.00285
700	27.4515	3.22775	0.00165	0.00222
800	41.0484	4.71010	0.00189	0.00241
900	58.4698	6.63860	0.00208	0.00289
1000			0.00239	0.00634
2000			0.00463	0.00871
3000			0.01014	0.01446
4000			0.01323	0.01675
5000			0.01474	0.02439
6000			0.02322	0.02608
7000			0.02192	0.03189
8000			0.02613	0.03847
9000			0.03415	0.04436

Experimental Run Time Plots





Experimental Run Time Analysis

Algorithm 1: $y = 8 \times 10^{-8} x^{3.0026}$
Algorithm 2: $y = 4 \times 10^{-8} x^{2.7892}$
Algorithm 3: $y = 1 \times 10^{-6} x^{1.0921}$
Algorithm 4: $y = 2 \times 10^{-6} x^{1.1231}$

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: $\Theta(n^3)$
Algorithm 3: $O(n^2), \Omega(n)$
Algorithm 4: $O(n \log n), \Omega(\log n)$

Extrapolation and Interpretation

Estimated Number of Lines Per Hour

Algorithm 1: $\sim 3,532$
Algorithm 2: $\sim 8,460$
Algorithm 3: $\sim 562,848,000$
Algorithm 4: $\sim 174,111,000$

Discrepancies

We note that the slopes from our experimentally-derived equations are within the asymptotic run-time range of $O(n^2)$, $\Omega(n)$ and $O(n \log n)$, $\Omega(\log n)$. However, our implementation of Algorithm 4 actually seems to run slightly slower compared to our Algorithm 3, despite Algorithm 4 having smaller minimum and maximum asymptotic runtimes. This discrepancy has many possible contributing factors, including a small sample size, differences in how the compiler and system the operations performed in each algorithm, the randomness of the input data sets. This randomness is due to both algorithms not performing a fixed number of operations, but rather changing what operations they run depending on the input lines given. It is possible that our tests are biased toward data sets that provide an advantage to Algorithm 3 (or a disadvantage to Algorithm 4). This randomness and compiler/system code handling are likely the primary causes of these differences.

Pseudocode

```
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):
    # Note: A indices go from left to right, while B indices go from right to left.
    # Note: We skip the first line of each element because the lines with least and most slope are alwa
    A_prev = A[0]
    A_new = A[1]
    B_prev = B[0]
    B_new = B[1]
    while checking A's or checking B's:
        if checking A's:
            # Check the next line in A.
            x*, y* = intersection(A_prev, B_prev)
            testLineY = A_new.slope * x* + A_new.intercept
            if y* > testLineY:
                stop checking A's (because the rest of them are invisible)
            else:
                A_prev++, A_new++ # Prepare for 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:
                    x*, y* = intersection(A_prev, B_prev)
                    testLineY = B_new.slope * x* + B_new.intercept
                    if y* > testLineY:
                        stop checking B's (because the rest of them are invisible)
                        B_prev--, B_new-- # Backtrack because we overstepped.

        if checking B's:
            # Check the next line in B.
            x*, y* = intersection(B_prev, A_prev)
            testLineY = B_new.slope * x* + B_new.intercept
            if y* > testLineY:
                stop checking B's (because the rest of them are invisible)
            else:
                B_prev++, B_new++ # Prepare for the next iteration.

                # Now we check to make sure that we didn't just cover the last line in 'A'.
                if there is a previously added line in j:
                    x*, y* = intersection(B_prev, A_prev)
                    testLineY = A_new.slope * x* + A_new.intercept
                    if y* > testLineY:
                        stop checking A's (because the rest of them are invisible)
                        A_prev--, A_new-- # Backtrack because we overstepped.
```

```
vlines = a[:A_new] + b[B_new:]
return vlines
```