CSCC73 A1

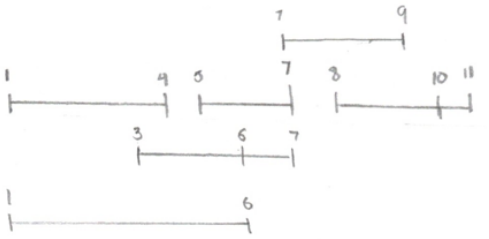
Saad Makrod

September 2022

Q1 Minimum Cover of Closed Interval Sequence

Part A:

The proposed greedy algorithm does not always find a minimum cover for $\mathcal{I}$. Consider the following counter-example:



Total of 8 intervals
{[1, 4], [1, 6], [3, 6], [5, 7]
[6, 7], [7, 9], [8, 10], [10, 11]}
= $\mathcal{I}$

In this counter-example $\mathcal{I}$ contains 8 intervals, as follows $\mathcal{I} = \{[1,4],[1,6],[3,6],[5,7],[6,7],[7,9],[8,10],[10,11]\}$.

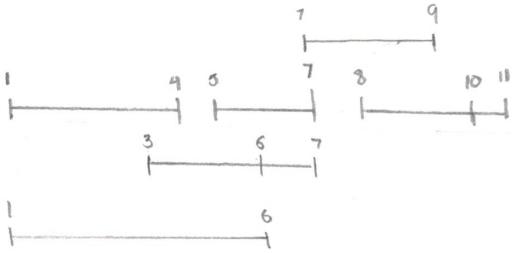
When the algorithm is run on this counter-example the first c that is chosen must be 6. This is because 6 is contained by 4 intervals: $\{[1,6],[3,6],[5,7],[6,7]\}$, and the next possible c values are contained by a maximum of 3 intervals. Once the intervals that contain 6 are removed we are left with $\{[1,4],[7,9],[8,10],[10,11]\}$. From here we can pick c such that there are two different outcomes, one choice is $c = 10$ which will remove $\{[8,10],[10,11]\}$ and the other is $c \in [8,9]$ which will remove $\{[7,9],[8,10]\}$.

In the case where $c = 10$ is chosen the remaining intervals are $\{[1,4],[7,9]\}$. From here it is obvious that two more cs will have to be chose to cover $\mathcal{I}$. One being some $c \in [7,9]$ and the other being some $c \in [1,4]$. Overall the set C will contain 4 values.

In the case where $c \in [8,9]$ is chosen the remaining intervals are $\{[1,4],[10,11]\}$. From here it is obvious that two more cs will have to be chose to cover $\mathcal{I}$. One being some $c \in [1,4]$ and the other being some $c \in [10,11]$. Overall the set C will contain 4 values.

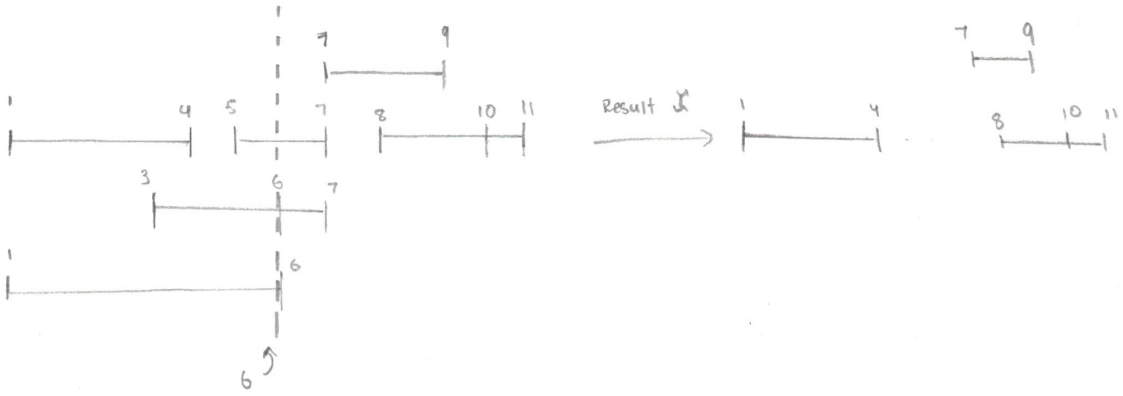
Thus in any case the algorithm results in $|C| = 4$.

A diagram of the algorithm can be seen below:



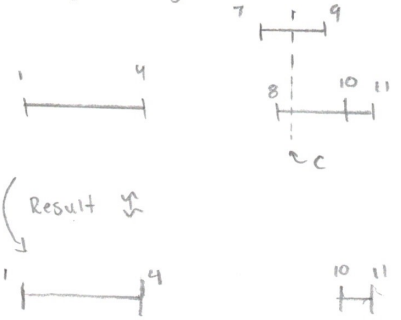
Total of 8 intervals
$\{[1,4], [1,6], [3,6], [5,7]$
$[6,7], [7,9], [8,10], [10,11]\}$
$= I$

By Algorithm First c Chosen is 6 (contained in 4 intervals)



By Algorithm Next $c \in [8,9]$ or $c = 10$ (both contained by 2 intervals)

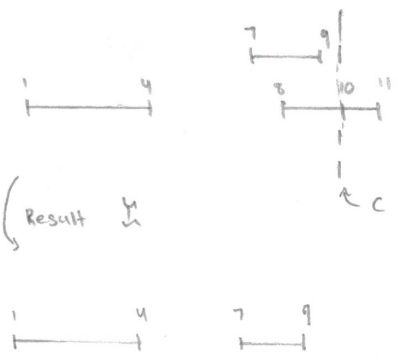
$c \in [8,9]$



From this point it takes two choices of c one where $c \in [1,4]$ and one where $c \in [10,11]$

Thus $|C| = 4$

$c = 10$

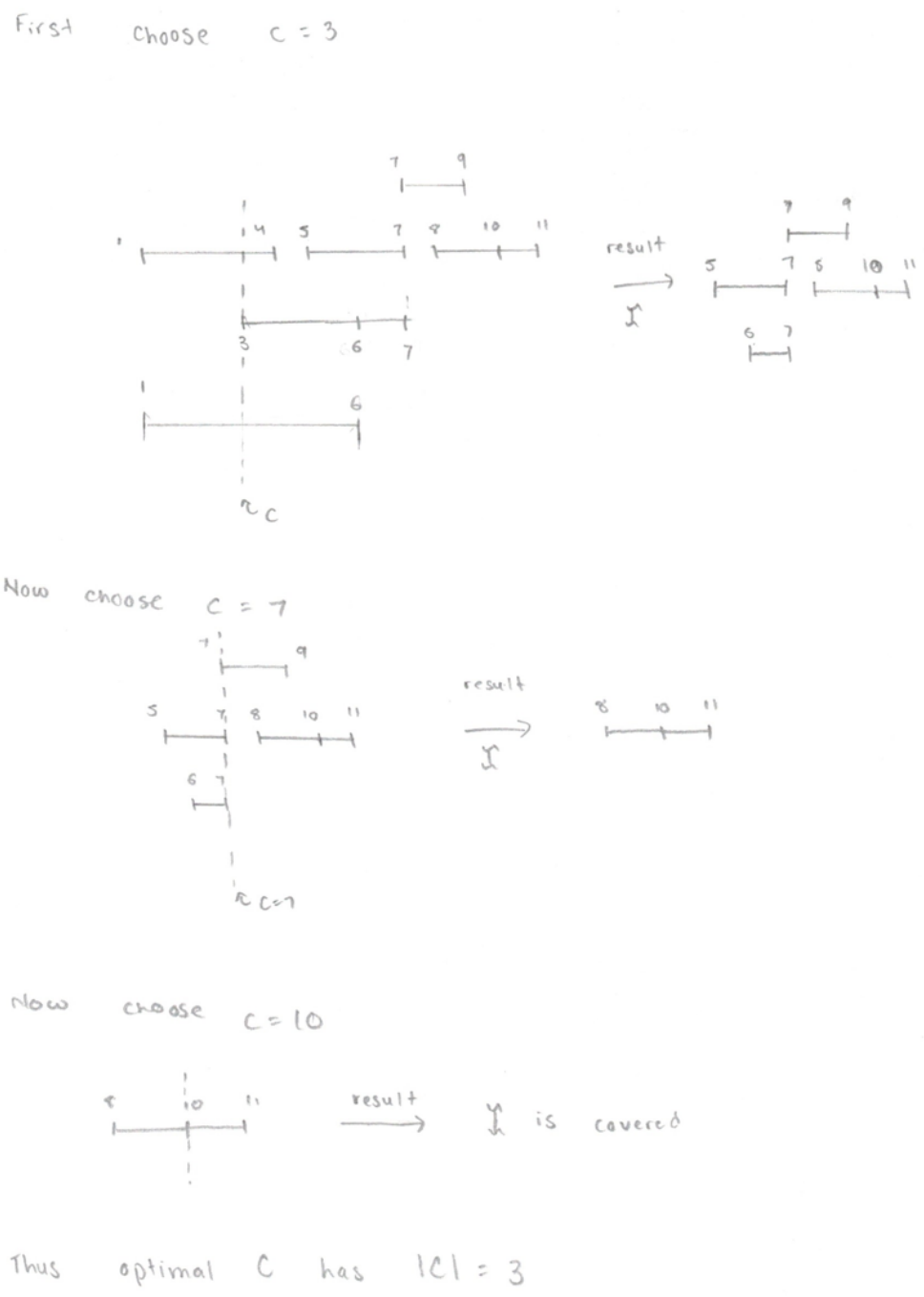


From this point must pick two more c, where $c \in [1,4]$ and $c \in [7,9]$

Thus $|C| = 4$

$\therefore$ in all cases need 4 values to cover I by algorithm.

However this is not the optimal solution. The figure below shows the optimal solution actually results in $|C| = 3$:



Note that this is not the only solution which leads to $|C| = 3$. Thus we can see that the optimal solution for this example is when $|C| = 3$.

Therefore the proposed greedy algorithm cannot find always find an optimal solution.

Part B:

Consider the following greedy algorithm:

sort $\mathcal{I}$ in increasing order of the s value of each interval

$C := \emptyset$

while $\mathcal{I} \neq \emptyset$

let i be the first interval in $\mathcal{I}$

let $c \in i$ be the smallest number that is contained by the largest amount of consecutive intervals in $\mathcal{I}$

$C := C \cup \{c\}$

delete from $\mathcal{I}$ all intervals that contain c

return C

Assume that when two or more intervals have the same s value the smaller interval is chosen to be i. In other words, if two or more intervals have the same s value they are sorted in increasing order of their f values.

Before explaining the idea behind the algorithm I should first make clear what I mean by "let $c \in i$ be the smallest number that is contained by the largest amount of consecutive intervals in $\mathcal{I}$". This means that when the algorithm is choosing a $c \in i$ once it encounters an overlap with an interval, c must be contained by that interval. So for example if we have the ordered $\mathcal{I}$ such that $\mathcal{I} = \{[1, 20], [2, 19], [17, 19], [20, 21], [20, 25], [20, 100]\}$ then the c chosen would be $c = 17$. This is because $\{[1, 20], [2, 19], [17, 19]\}$ is the largest number of consecutive intervals in $\mathcal{I}$ that overlap and 17 is the smallest number that is contained by all of them. Even though choosing $c = 20$ would result in 4 intervals being removed it is not chosen since $\{[1, 20], [20, 21], [20, 25], [20, 100]\}$ are not consecutive in the ordered $\mathcal{I}$.

The intuitive idea here is that as we are moving left to right we are trying to clear as many intervals as possibles without leaving any interval behind. From the counter-example I noticed that the proposed algorithm failed when I forced it to clear many intervals in the middle but leave several on the edges. As a result, the algorithm I proposed moves left to right and clears as many intervals as possible in one go without leaving any behind.

Proof

Note that for this proof I will refer to the sorted $\mathcal{I}$ as $\mathcal{I}$ for simplicity.

We will prove this algorithm works with the "Greedy Stays Ahead" method. Consider the set of numbers C constructed by the algorithm. By the algorithm, intervals are only deleted from $\mathcal{I}$ if there is a $c \in C$ which is contained by them, thus C is a feasible solution. Let $c_1, c_2, \ldots, c_k$ be the list of these numbers. It is important to note that these numbers are sorted in increasing order. This is because the algorithm moves in left-to-right order so each c added to the set will be larger than the one which preceded it. Let C^* be any optimal covering set of $\mathcal{I}$ ordered in increasing order. Let $c_1^*, c_2^*, \ldots, c_m^*$ denote the set of numbers in C^*. Note C^* is feasible since as stated earlier it is an optimal covering set. Note that there are many optimal solutions so instead of proving $C = C^*$, I will prove that $|C| = |C^*|$. If I can show that $k = m$ then I know that C is optimal.

Before proving the correctness of the algorithm I will prove the following Lemma:

For every t, $1 \leq t \leq k$ $\sum_{i=1}^{t}$ *number of intervals covered by* $c_i^* \leq \sum_{i=1}^{t}$ *number of intervals covered by* c_i

Lemma Proof

The proof will be done by induction on t.

The base case $t = 1$ is clear by the algorithm. I know that c_1 and c_1^* must both be contained by the interval with the earliest start time. This is because they are the smallest values in their sets respectively. Assume for contradiction, that *number of intervals covered by* $c_1^* >$ *number of intervals covered by* c_1, this is indeed possible but these intervals cannot be consecutive in $\mathcal{I}$ since if they were the algorithm would choose a c that is contained by them as well. This means that the optimal set C^* is leaving at least one interval behind such that c_1^* is greater than the f value for this interval, we will refer to this f value as f^*. Note

that it is impossible for this interval to have already been covered since c_1 and c_1^* are the first values in their respective sets. However, since C^* is feasible it must cover this interval too, which implies that there exists a $c_i^* \in C^*$ that covers the interval. I know that this $c_i^* \leq f^*$ since it must be contained by the interval that was left behind. Thus I have $c_i^* \leq f^* < c_1^*$. This is impossible since the set C^* is sorted in increasing order, so there cannot exist a c_i^* that is less than c_1^*. This implies that C^* is not feasible since it cannot cover the left behind interval. This is a contradiction since C^* is both feasible and optimal. Thus it is indeed true that *number of intervals covered by* $c_1^* \leq$ *number of intervals covered by* c_1 and it follows that $\sum_{i=1}^{1}$ *number of intervals covered by* $c_i^* \leq \sum_{i=1}^{1}$ *number of intervals covered by* c_i. Thus the basis holds.

For the induction step, suppose that $\sum_{i=1}^{t}$ *number of intervals covered by* $c_i^* \leq \sum_{i=1}^{t}$ *number of intervals covered by* c_i. I want to prove that $\sum_{i=1}^{t+1}$ *number of intervals covered by* $c_i^* \leq \sum_{i=1}^{t+1}$ *number of intervals covered by* c_i. Suppose for contradiction, that $\sum_{i=1}^{t+1}$ *number of intervals covered by* $c_i^* > \sum_{i=1}^{t+1}$ *number of intervals covered by* c_i. Note that since we have $\sum_{i=1}^{t}$ *number of intervals covered by* $c_i^* \leq \sum_{i=1}^{t}$ *number of intervals covered by* c_i, this means that *number of intervals covered by* $c_{t+1}^* > \sum_{i=1}^{t}$ *number of intervals covered by* $c_i - \sum_{i=1}^{t}$ *number of intervals covered by* c_i^* + *number of intervals covered by* c_{t+1}. There are two cases, when $\sum_{i=1}^{t}$ *number of intervals covered by* $c_i^* = \sum_{i=1}^{t}$ *number of intervals covered by* c_i and when $\sum_{i=1}^{t}$ *number of intervals covered by* $c_i^* < \sum_{i=1}^{t}$ *number of intervals covered by* c_i.

In the case where $\sum_{i=1}^{t}$ *number of intervals covered by* $c_i^* = \sum_{i=1}^{t}$ *number of intervals covered by* c_i we need *number of intervals covered by* $c_{t+1}^* >$ *number of intervals covered by* c_{t+1}. Note $\sum_{i=1}^{t}$ *number of intervals covered by* $c_i^* = \sum_{i=1}^{t}$ *number of intervals covered by* c_i means that the current intervals covered by C and C^* are equal. This is because C and C^* are ordered, so $c_1, c_2, \ldots, c_t$ and $c_1^*, c_2^*, \ldots, c_t^*$ cover intervals left-to-right, if their values are equal then they both cover the same number of intervals from the left. However this causes a very similar issue to the base case. c_{t+1} and c_{t+1}^* must both be contained by the next interval which is not covered. Again this is due to the ordered property of C and C^*. It is indeed possible that *number of intervals covered by* $c_{t+1}^* >$ *number of intervals covered by* c_{t+1} but these intervals cannot be consecutive in $\mathcal{I}$ since if they were the algorithm would choose a c that is contained by them as well. This means that the optimal set C^* is leaving at least one interval behind such that c_{t+1}^* is greater than the f value for this interval and c_t^* is less than the s value for this interval, we will refer to these as f^* and s^* respectively (note this is different from the earlier f^*). Also note that it is impossible for this interval to have already been covered since C and C^* have currently covered the same exact intervals. However, since C^* is feasible it must cover this interval too, which implies that there exists a $c_i^* \in C^*$ that covers the interval. I know that this $c_t^* < s^* \leq c_i^* \leq f^*$ since it must be contained by the interval that was left behind. Thus I have $c_t^* < s^* \leq c_i^* \leq f^* < c_{t+1}^*$. This is impossible since the set C^* is sorted in increasing order, so there cannot exist a c_i^* that is less than c_{t+1}^* and greater than c_t^*. This implies that C^* is not feasible since it cannot cover the left behind interval. This is a contradiction since C^* is both feasible and optimal. Thus in this case it is impossible for *number of intervals covered by* $c_{t+1} <$ *number of intervals covered by* c_{t+1}^* and it follows that $\sum_{i=1}^{t+1}$ *number of intervals covered by* $c_i^* \leq \sum_{i=1}^{t+1}$ *number of intervals covered by* c_i.

Let n be $\sum_{i=1}^{t}$ *number of intervals covered by* $c_i - \sum_{i=1}^{t}$ *number of intervals covered by* c_i^*. In the case where $\sum_{i=1}^{t}$ *number of intervals covered by* $c_i^* < \sum_{i=1}^{t}$ *number of intervals covered by* c_i we need *number of intervals covered by* $c_{t+1}^* > n+$ *number of intervals covered by* c_{t+1}. Note that this is impossible. Due to the order property of C and C^*, c_{t+1}^* must cover the n intervals that were covered earlier in C. Thus c_{t+1}^* is bounded by the earliest finish time of these n intervals. We will again use f^* to denote this (again note this is a different f^* then both previous ones). If the intervals covered by c_{t+1} overlap with the n intervals then it is indeed possible for *number of intervals covered by* $c_{t+1}^* = n+$ *number of intervals covered by* c_{t+1}, however it can be no greater than this. This is because if this were possible then this would mean that the algorithm would have chosen a c that also covers them since they would overlap with the intervals that c_{t+1} currently is covering. In other words, the s values for these intervals are all greater than f^* which means that a c_{t+1}^* cannot be chosen which would cause them to overlap without breaking the order property of C^*. Thus we have come to a contradiction it is impossible for *number of intervals covered by* $c_{t+1}^* > n+$ *number of intervals covered by* c_{t+1} and it follows that $\sum_{i=1}^{t+1}$ *number of intervals covered by* $c_i^* \leq \sum_{i=1}^{t+1}$ *number of intervals covered by* c_i.

Since a contradiction was found in both cases the induction step is complete.

QED Lemma 1 Proof

Algorithm Proof

Since C is feasible, $k \geq m$. Suppose, for contradiction, that C is not optimal; i.e., $k > m$. So C contains a number c_{m+1}. This means that if the algorithm choose to add c_{m+1} there must be another interval that was not covered by the m other cs. By the Lemma $\sum_{i=1}^{m}$ *number of intervals covered by* $c_i^* \leq \sum_{i=1}^{m}$ *number of intervals covered by* c_i. Since a c_{m+1} was added we have $\sum_{i=1}^{m}$ *number of intervals covered by* $c_i^* < \sum_{i=1}^{m+1}$ *number of intervals covered by* c_i. This means that C^* is not feasible since there is an interval covered by C but not by C^*. This is a contradiction since C^* is both optimal and feasible. Thus $k = m$.

QED Algorithm Correctness

Q2 Minimum Doubles and Increments to Reach n

Consider the following algorithm:

```
FewestSteps(n)
    create a new stack s
    while n is not 1
        if n is even
            n = n / 2
            add 'D' to s
        else
            n = n - 1
            add 'I' to s
    return s
```

This algorithm is feasible since it works down from n to 1 by dividing and decrementing. So to get from 1 to n the only thing you have to do is repeat the opposite operation in the reverse order. So instead of dividing by 2 you double and instead of decrementing you increment. Since the algorithm returns a stack the order of the operations is simply in the same order you get when popping them from the stack. Thus the sequence returned is a correct sequence.

The idea behind the algorithm is to perform as many doubles as possible. This is because doubles save more steps than increments. So by working down from n, the algorithm looks for any opportunity to divide by 2 and only if it cannot divide by 2 then it will decrement by 1.

If n is a power of 2 then this algorithm clearly runs in $log_2(n)$ time. In the the general case however it is not so clear what the running time of the algorithm is. Since the algorithm is designed to divide by 2 as much as possible the number of divisions is clearly $\lfloor log_2(n) \rfloor$. It is hard to determine the exact number of decrements but we can determine a bound. Since n is decremented only when it is odd we know that each decrement leads to division (since any odd number decremented by 1 is even). However, it is not necessarily the case that the a division leads to a decrement (see the case where n is a power of 2). Thus the number of decrements is strictly bounded by the number of divisions. We know that the number of divisions is $\lfloor log_2(n) \rfloor$. Thus *total complexity* $=$ *number of decrements* $+$ *number of divisions* $\leq$ $log_2(n) + log_2(n) = 2log_2(n) \in O(log_2(n))$. Therefore the running time of the algorithm is $O(log_2(n))$.

Proof

Let S be the sequence proposed by the algorithm and $s_1, s_2, \ldots, s_k$ be the steps of the sequence. From above I know that S is a feasible solution. Before proving the correctness of the algorithm I will prove the following Lemma 2:

Let S^* *be any optimal sequence of doubles and increments. Let* $s_1^*, s_2^*, \ldots, s_m^*$ *denote the sequence of steps in* S^**. If n is an even number greater than 2 then* s_m^* *is double*

Lemma Proof

The proof will be done by contradiction. Suppose this is not the case and s^*_m is increment. Note that if this is the case s^*_{m-1} is also increment. This is because $n-1$ is odd so there is no way to double to $n-1$. Now suppose that s^*_{m-2} is double. This is because any further increments will only increase the number of steps. So the steps $s^*_{m-2}, s^*_{m-1}, s^*_m$ represent D, I, I. Performing these steps in reverse $(n-1-1)/2 = n/2-1$. Thus from this sequence it takes three steps to get from $n/2-1$ to n.

However, if s^*_m was double then it would only take 2 steps to get from $n/2-1$ to n (increment then double). So we can see that by having s^*_m as increment we do not have an optimal sequence. This is a contradiction since S^* is optimal.

By a similar argument, if instead of just s^*_{m-1}, s_m being increments, suppose that $s^*_i, s^*_{i+1}, \ldots, s^*_m$ are increments, where $1 \leq i \leq m$. Note that $i - m = 2x$ where x is some positive number. This is because we cannot have an odd number of increments since we cannot double to an odd number. Now there are two cases there are no doubles or s^*_{i-1} is double.

If there are no doubles then clearly S^* is only increments which cannot be optimal unless $n \leq 3$. We do not have to worry about this since this lemma applies to even numbers greater than 2. So we have a contradiction since S^* is not optimal.

If s^*_{i-1} is double then we can evaluate $(n-2x)/2 = n/2 - x$. So from $n/2-x$ the optimal sequence chose to double and then increment $2x$ times. However this is clearly not optimal. Instead we can choose to increment x times from $n/2-x$ and then double which leads to $(n/2 - x + x) * 2 = n$. This is $x+1$ steps instead of the $2x+1$ steps that was chosen by S^*. Thus we have a contradiction again since S^* must be optimal.

Thus s^*_m must be double.

QED Lemma 2 Proof

Algorithm Proof

This proof will use complete induction on n. The goal of this proof is to show that S is optimal for all n.

The basis is trivial. Let $n = 1$, since we start at 1 there is no need to increment or double, thus the optimal behavior is to return nothing. The algorithm proposed will return nothing so it performs optimally.

For the induction step, suppose that the algorithm is optimal for 1 to n, I want to show it is optimal for $n+1$. Suppose for contradiction, that my algorithm is not optimal for $n+1$. Note that there are two cases, $n+1$ is even or $n+1$ is odd.

If $n+1$ is even then my algorithm will choose to divide by 2 and add D onto the stack. By Lemma 2 this is the optimal move if $n > 2$. Thus there are two cases, when $n = 2$ or $n > 2$. If $n = 2$ this is an optimal choice since the optimal sequence to get to 2 is either D or I. Since both take one step, either are valid. Thus in this case there is a contradiction since S is optimal. In the case where $n > 2$, D is the optimal choice for the last step by Lemma 2. This means that something must have went wrong when the algorithm was determining the optimal sequence for $(n+1)/2$. However, this is a contradiction, since it was assumed that the algorithm was optimal from 1 to n. Clearly $1 \leq (n+1)/2 \leq n$. Thus if $n+1$ is even the algorithm is optimal.

If $n+1$ is odd my algorithm will add I onto the stack. It is trivial that this is the optimal move since you cannot double to an odd number so the last step must always be an increment. Since this is the optimal move something must have went wrong when the algorithm was determining the optimal sequence for n. However, this is a contradiction, since it was assumed that the algorithm was optimal from 1 to n, it must be optimal for n. Thus if $n+1$ is odd the algorithm is optimal.

Since there was a contradiction in both cases this completes the induction step. Thus clearly the algorithm is optimal and feasible.

QED Algorithm Correctness