Ashwin Shenai
180156

Homework 1

ESO207A
August 19, 2019

Problem 1. Order Notation and Recurrence Relations

Part (a)

$$f(n) = \mathcal{O}(g(n)) \text{ iff } \exists\ \mathcal{M} > 0, n_0 \in \mathbb{N} \text{ such that } |f(n)| \leq \mathcal{M}g(n)\ \forall\ n \geq n_0$$
$$f(n) = \Omega(g(n)) \text{ iff } \exists\ \mathcal{M} > 0, n_0 \in \mathbb{N} \text{ such that } |f(n)| \geq \mathcal{M}g(n)\ \forall\ n \geq n_0$$
$$f(n) = \Theta(g(n)) \text{ iff } \exists\ c_1, c_2 > 0, n_0 \in \mathbb{N} \text{ such that } c_1 g(n) \leq |f(n)| \leq c_2 g(n)\ \forall\ n \geq n_0$$
$$f(n) = \Theta(g(n)) \text{ iff } f(n) = \Omega(g(n)) \text{ and } f(n) = \mathcal{O}(g(n))$$

(i) Using the definition, let $\mathcal{M} = 1$ and $n_0 = 4$. Then $\forall\ n \geq n_0$, we have

$$|f(n)| \leq \mathcal{M}g(n) \implies n^2 \leq 2^n$$

Taking logarithm of both sides with base 2,

$$2log(n) \leq n \qquad \text{(since } log(n) \geq 0 \text{ for } n \geq 1\text{)}$$

Then for $n_0 = 4$,

$$2log(4) = 4$$

Since $logn$ grows slower than n,

$$\forall\ n > 2,\ 2log(n) < n$$

Hence the definition holds with $\mathcal{M} = 1$ and $n_0 = 2$.
Therefore $n^2 = \mathcal{O}(2^n)$ and the statement is **TRUE**

(ii) Using the definition $n^3 = \mathcal{O}(n^4 log(n))$ iff $n^3 \leq \mathcal{M}n^4 log(n)\ \forall\ n \geq n_0$ for some $\mathcal{M}$, n_0
Let us assume there exists such an $\mathcal{M}$ and n_0. Then $\forall\ n \geq n_0$,

$$n^3 \leq \mathcal{M}n^4 log(n) \implies \mathcal{M}nlog(n) \geq 1 \qquad \text{(since } n^3 > 0\ \forall\ n \geq 1\text{)}$$

Choosing $\mathcal{M} = 1$, we must find n_0 such that

$$nlog(n) \geq 1 \qquad \forall\ n \geq n_0$$

For $n_0 = 2$,

$$2log(2) > 1$$

Note that $xlogx$ is monotonically increasing $\forall\ x > 1$. So $\forall\ n > 2,\ nlog(n) > 1$
Hence the definition holds with $\mathcal{M} = 1$ and $n_0 = 2$.
Therefore $n^3 = \mathcal{O}(n^4 log(n))$ and the statement is **TRUE**

(iii) Using the definition $n^2 = \Omega(n^3)$ iff $n^2 \geq \mathcal{M}n^3\ \forall\ n \geq n_0$ for some $\mathcal{M}$, n_0
Let us assume there exists such an $\mathcal{M}$ and n_0. Then $\forall\ n \geq n_0$,

$$n^2 \geq \mathcal{M}n^3 \implies \mathcal{M}n \leq 1 \qquad \text{(since } n^2 > 0\ \forall\ n \geq 1\text{)}$$

Choosing $\mathcal{M} = 1/k$ where $k > n_0$, we get

$$n \leq k \qquad \forall\ n \geq n_0$$

However since infinite natural numbers exist we can find an $m > k > n_0$ for which the above would not hold.
Thus the above statement leads to a contradiction and there does not exist such an $\mathcal{M}$ and n_0.
Therefore $n^2 \neq \Omega(n^3)$ and the statement is **FALSE**

(iv) Note that $x! > 3^x \ \forall \ x \geq 7$. Therefore $3^n = \mathcal{O}(n!)$
Using the definitions $3^n = \Omega(n!)$ iff $3^n \geq \mathcal{M}n! \ \forall \ n \geq n_0$ for some $\mathcal{M}$, n_0
Let us assume there exists such an $\mathcal{M}$ and $n_0(> 6)$. Then

$$\forall \ n \geq n_0, \ 3^n \geq \mathcal{M}n!$$

From the above statement, $\exists k$ such that

$$3^{(n_0+k)} \geq \mathcal{M}(n_0+k)!$$

Assuming $3^{n_0} = \mathcal{M}n_0!$ (otherwise let n_0 be such that $3^{n_0} < \mathcal{M}n_0 < 3^{n_0+1}$ and replace n_0 with $n_0 + 1$ in the definition), since $n_0 > 6$, we get

$$n_0 + i > 3 \ \forall \ i > 0,$$
$$\implies 3^k < (n_0+k)(n_0+k-1)...(n_0+1)$$

Multiplying both sides by $n_0!$,

$$3^k n_0! < (n_0+k)!$$

Thus $\mathcal{M}(n_0+k)! > \mathcal{M}3^k n_0! = 3^{n_0+k}$, which contradicts the above statement.
Thus there does not exist such an $\mathcal{M}$ and n_0 Hence, $3^n \neq \Omega(n!) \implies 3^n \neq \Theta(n!)$.
The statement is thus **FALSE**

(v) Using the definition, let $c_1 = 2^{-(c+1)}$, $c_2 = 2^{-(c-1)}$ and $n_0 = 1$. Then $\forall \ n \geq n_0$, we have

$$c_1 g(n) \leq |f(n)| \leq c_2 g(n)$$
$$\implies 2^{n+c} * 2^{-(c+1)} \leq 2^n \leq 2^{n+c} * 2^{-(c-1)}$$
$$\implies 2^{n-1} \leq 2^n \leq 2^{n+1}$$

Then for $n_0 = 1$, we have $1 \leq 2 \leq 4$. Since 2^n increases monotonically for $n > 0$,

$$\forall \ n > 1, \ 2^{n-1} \leq 2^n \leq 2^{n+1}$$

Hence the definition holds with $c_1 = 2^{-(c+1)}$, $c_2 = 2^{-(c-1)}$ and $n_0 = 1$.
Thus $2^n = \Theta(2^{n+c})$ and the statement is thus **TRUE**

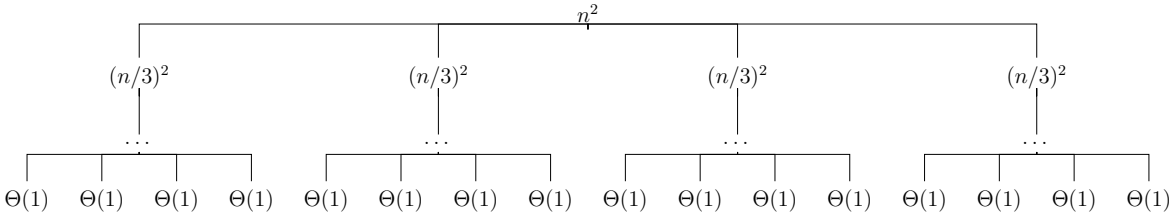
Part (b)

1. $T(n) = T(n/2) + n$; $T(1) = \Theta(1)$
   Assuming n = 2^k, we get the following recursion tree with $k = log_2(n)$ levels:

   n — n/2 — n/4 — n/8 — ... — $\Theta(1)$

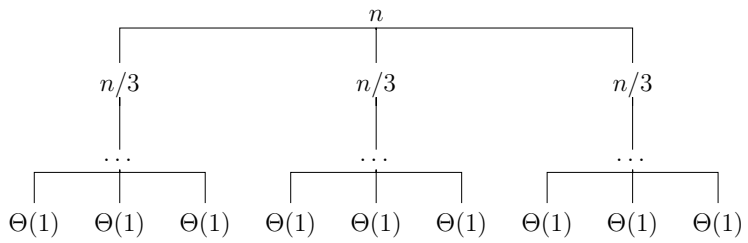
   Adding up the time taken at each level, we get
   $T(n) = n + n/2 + n/4 + n/8....n/2^{k-1} + \Theta(1)$
   $\implies T(n) = n(1 + 1/2 + 1/4 + 1/8....1/2^{k-1}) + \Theta(1)$
   $\implies T(n) = n(1 - 1/2^k)/(1 - 1/2) + \Theta(1)$
   $\implies T(n) = n(2^k - 1) * 2/2^k + \Theta(1)$
   $\implies T(n) = 2n(n-1)/n + \Theta(1) \implies T(n) = n - 1 + \Theta(1)$
   Since $n + \Theta(1) = \mathcal{O}(n)$ and $n + \Theta(1) = \Omega(n)$, we have $T(n) = \Theta(n)$

2. $T(n) = 4T(n/3) + n^2$; $T(1) = \Theta(1)$
   Assuming $n = 3^k$, we get the following recursion tree with $k = log_3 n$ levels:



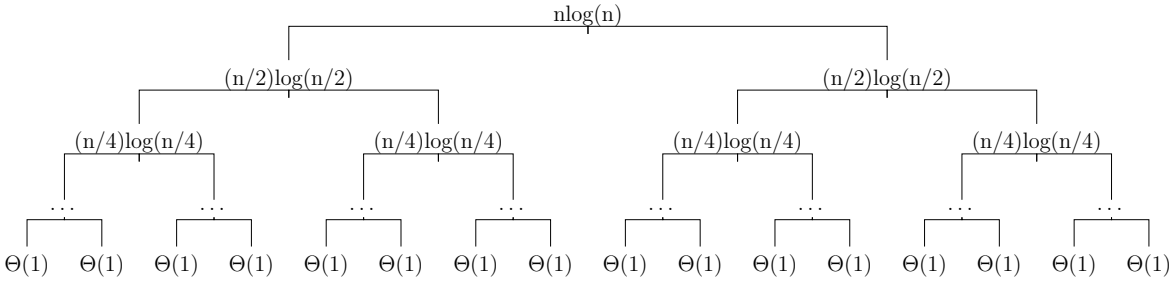
   Adding up the time taken at each level, we get:
   $T(n) = n^2 + 4(n/3)^2 + 4^2(n/9)^2 + ... + 4^{k-1}(n/3^{k-1})^2 + 4^k\Theta(1)$
   $\implies T(n) = n^2(1 + 4/9 + (4/9)^2 + ... + (4/9)^{k-1}) + 4^k\Theta(1)$
   $\implies T(n) = n^2(1 - (4/9)^k)/(1 - 4/9) + 4^k\Theta(1)$
   $\implies T(n) = 9n^2(3^{2k} - 2^{2k})/5.3^{2k} + 4^k\Theta(1)$
   $\implies T(n) = 9n^2(n^2 - n^{log_3 4})/5n^2 + n^{log_3 4}\Theta(1)$
   $\implies T(n) = 9n^2/5 + n^{log_3 4}(\Theta(1) - 9/5)$
   Since $2 > log_3 4, n^{log_3 4} = \mathcal{O}(n^2)$. Therefore we have T(n) = $\mathcal{O}(n^2)$

3. $T(n) = 3T(n/3) + n$
   Assuming $n = 3^k$, we get the following recursion tree with $k = log_3 n$ levels:



Adding up the time taken at each level, we get:
$T(n) = n + 3(n/3) + 3^2(n/9) + ... + 3^{k-1}(n/3^{k-1}) + 3^k\Theta(1)$
$\implies T(n) = n + n + n + ... + n + 3^k\Theta(1)$
$\implies T(n) = n(k-1) + n\Theta(1) \implies T(n) = nlog_3n - n + n\Theta(1)$
$\implies T(n) = nlog_3n - n + \Theta(n)$
Since $n = \mathcal{O}(nlog_3n)$, we have $T(n) = \mathcal{O}(nlog_3n)$

4. $T(n) = 2T(n/2) + nlog(n)$; $T(1) = \Theta(1)$
Assuming n = 2^k, we get the following recursion tree with $k = log_2(n)$ levels:



Adding up the time taken at each level, we get
$T(n) = nlog(n) + 2(n/2)log(n/2) + 4(n/4)log(n/4) + ... + 2^{k-1}(n/2^{k-1})log(n/2^{k-1}) +$
$2^k\Theta(1) \implies T(n) = n(log(n) + log(n/2) + log(n/4) + ... + log(n/2^{k-1})) + n\Theta(1)$
$\implies T(n) = n(klogn - (log2 + log4 + ...log2^{k-1})) + \Theta(n)$
$\implies T(n) = nklogn - n(1 + 2 + ...k - 1) + \Theta(n)$
$\implies T(n) = n(logn)^2 - nk(k-1)/2 + \Theta(n)$
$\implies T(n) = (n(logn)^2 + nlogn)/2 + \Theta(n)$
$n(logn)^2 - nlogn = \mathcal{O}(n(logn)^2)$, therefore we have $T(n) = \mathcal{O}(n(logn)^2)$

Problem 2. Non-dominated points

Definitions

1. A two-dimensional point (x, y) is said to **dominate** another two-dimensional point (u, v) iff $x \geq u$ and $y \geq v$.

2. A point $p = (x, y) \in \mathcal{P}$ is said to be a **non-dominated point** of the set $\mathcal{P}$ iff there exists no point $q \in \mathcal{P}$ such that q dominates p.

Objective

Given a set of arbitrary points $\mathcal{P}$, find the set of non-dominated points in $\mathcal{P}$.

Assumptions

No two points in the given set $\mathcal{P}$ have the same x or y-coordinate. In other words, each point has a distinct x-coordinate and a distinct y-coordinate.

(Note: The algorithm given below does not use this assumption.)

Intuition

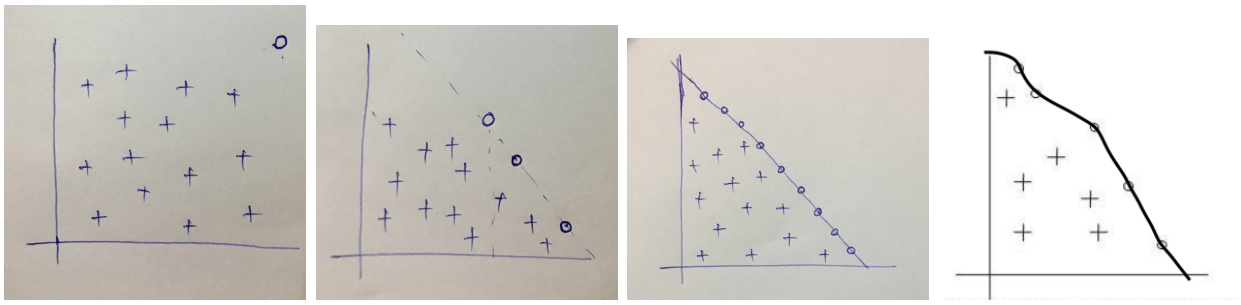


Figure 1: Visualization of a few sample sets of points

One can observe that in the set of non-dominated points (circles), as x increases, y decreases.

Lemmas

1. (x, y_1) is not dominated by (x, y_2) iff $y_1 > y_2$
   Proof: The above statement is just the negative of Definition 1. It can be easily proved by contradiction. Assuming $y_1 > y_2$ and(x, y_2) dominates (x, y_1), we must have $y_2 \geq y_1$, which isn't true. So (x, y_1) is not dominated by (x, y_2).

2. If all the points in $\mathcal{P}$ have the same x-coordinate x_0, then the set of non-dominated points contains only $(x_0, \texttt{max}(y(x_i)))$
   Proof: This can be proved by applying Lemma 1. Since a non-dominated point of $\mathcal{P}$ must not be dominated by any point in P by Definition 2, given that all the x-coordinates are equal the non-dominated point must have the maximum possible y-coordinate.

3. Given $x_1 < x_2$, the point (x_1, y_1) is not dominated by (x_2, y_2) iff $y_1 > y_2$
   Proof: Analogous to Lemma 1, this statement can be proved in the same way. Assuming $y_1 < y_2$ and (x_2, y_2) dominates (x_1, y_1), we must have $y_2 \geq y_1$ **and** $x_2 \geq x_1$, of which the first isn't true. Since the condition is **and**, (x_1, y_1) is not dominated by (x_2, y_2).

Algorithm

Following are the steps of the algorithm `FIND_NON_DOM`$(\mathcal{P}, n)$

1. Sort all the points in $\mathcal{P}$ in increasing order of x-coordinate. If two points have the same x-coordinate, we sort them in increasing order of y-coordinate.

2. Once the array is sorted x-wise, traverse the array from the reverse while keeping a running maximum in the y-coordinate. If at any point the y-coordinate is greater than the current maximum, that point is added to the result set. Also, only one element is checked per value of x-coordinate. This is done by keeping track of a running x-coordinate value, elements are skipped until it is updated.

Proof of Correctness

The first step of the algorithm is to sort the input array with respect to the x-coordinates. This is done so that Lemma 3 can be applied to consecutive elements.

For points with the same x-coordinate, we sort them with respect to the y-coordinate. This is done so that Lemma 1 can be applied to consecutive elements.

Again to apply Lemma 3, we initialize a running maximum in y and traverse the array from the end. When the maximum updates, the point is added to the result set, otherwise it is

ignored. While traversing the array from the end, since it has been sorted with respect to the x-coordinates, x decreases as we move towards the first element. The point with the maximum x-coordinate is a non-dominated point - this can be proved by interchanging x and y in Lemma 2. This point is the first to be added to the result set. As x decreases, we can check the y-coordinate ($= y_1$) with the running maximum ($= y_m$). If $y_1 > y_m$, by Lemma 3 the current point would be non-dominated - in this case the maximum is updated and the point is added to the result set. By this process all the non-dominated points are collected in the result set.

Also, by applying Lemma 2 we can further reduce the number of elements to be checked. Since for the same value of x, we have sorted the array in increasing order of y, the first element of that x value encountered when traversing the array from the end is the one with the maximum y value. By Lemma 2, other points with the same value of x cannot be non-dominated, hence they need not be checked at all. Thus only the maximum y-coordinate point needs to be checked for each distinct x-coordinate.

Pseudocode

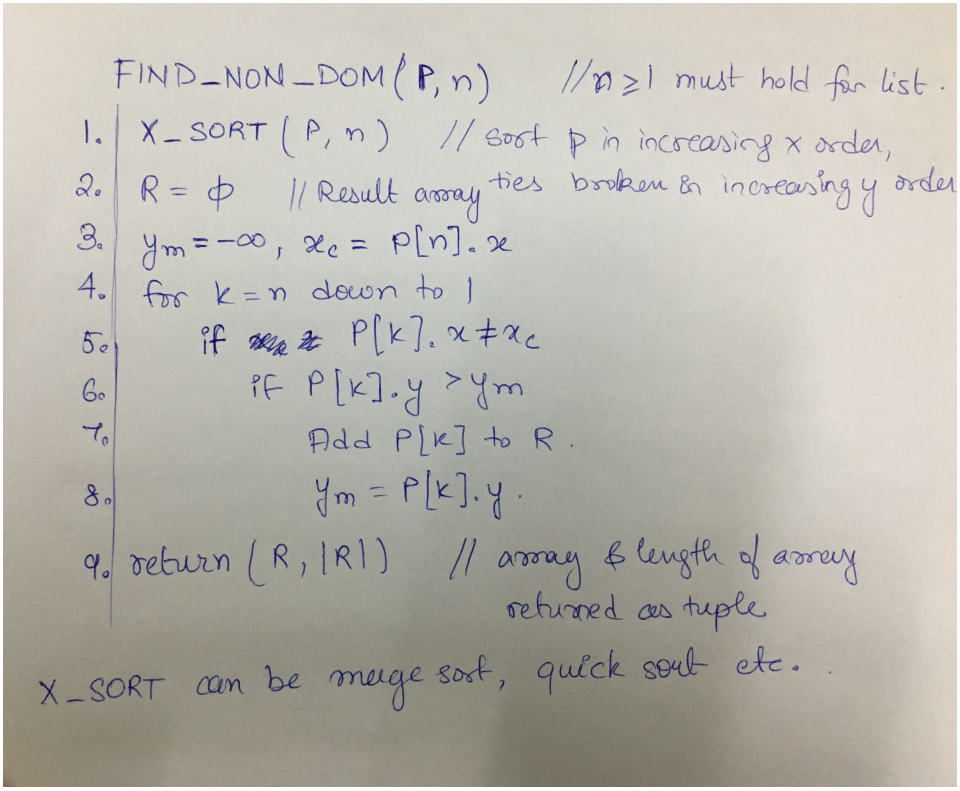


Figure 2: Pseudocode for the algorithm

Time Complexity Analysis

For sorting the array with respect to the x-coordinate we can use algorithms like merge sort (as above) or quick sort, etc.

Time complexity of sorting step: $\mathcal{O}(nlogn)$
In the traversal step, all comparisons, running value updates, counter updates take $\Theta(1)$ time. Since there are maximum n such sets of constant-time operations to be made,

Time complexity of traversal step: $nc_i\Theta(1) = \mathcal{O}(n)$
(where c_i is an element dependent constant)
Hence, **time complexity of both steps combined** $= \mathcal{O}(nlogn) + \mathcal{O}(n)$
Since asympotically $n < nlogn$, $\mathcal{O}(n) = \mathcal{O}(nlogn)$
Therefore **time complexity of algorithm** $= \mathcal{O}(nlogn)$

Problem 3. The Court Trial

Case	W1 says	W2 says	Conclusion
1	W2 is true	W1 is true	Both true or both false
2	W2 is true	W1 is false	At least one is false
3	W2 is false	W1 is true	At least one is false
4	W2 is false	W1 is false	At least one is false

Table 1: Truth Table

Part (a)

To show that if there are more than $N/2$ false witnesses, the judge cannot necessarily pick out the true witnesses irrespective of the pairwise test strategy used, it is sufficient to show that there exists a conspiracy that the false witnesses can concoct to fool the judge. If all the false witnesses participate in this conspiracy, there would be no way for the judge to thwart it without prior information about the conspiracy or the witnesses.
One strategy the false witnesses could follow is to pass off any false witness as a true witness and any true witness as a false witness in each of the judge's pairwise tests. Since the false witnesses are able to conspire, they must know which witness is false and which true and hence this strategy can be executed.
In a test, the following scenarios would arise:

Both true: Case 1 conclusion

Both false: Case 1 conclusion

One false and one true: Case 4 conclusion

Thus under this strategy the judge witnesses a symmetric sort of behaviour, with two separate sets of people saying that the other set is false. Since the number of false witnesses is more than $N/2$, irrespective of the pairwise strategy used we would get a completely consistent system.
If the judge assumes W1 to be true, they will be able to completely distinguish between those witnesses that agree with W1 and those that do not. But the judge cannot in any way determine whether W1 is actually a true witness, as for each witness who says 'W1 is true', there would be another witness that says 'W1 is false'. Thus the judge cannot necessarily find out the good witnesses in this case.

Part (b)

Since we have more than $N/2$ true witnesses and we must find one true witness through a divide-and-conquer approach, let us consider the simple invariant that the number of true

witnesses always remains greater than the number of false witnesses in each iteration. In this way, if the invariant holds throughout our algorithm, at the final step we would have 1 true witness and 0 false witnesses left.

Algorithm

If N is even, we make $N/2$ pairs and test them pairwise. If N is odd, we make $\lfloor N/2 \rfloor$ pairs at random, leaving one witness out to be included in the subsequent iteration of pairwise tests. Depending on the conclusions, we then take the following actions for each pairwise test:

Case	Conclusion	Action
1	Both true or both false	Remove one at random
2	At least one is false	Remove both
3	At least one is false	Remove both
4	At least one is false	Remove both

Table 2: Action Table

Proof of Correctness

We now prove that the above algorithm leaves us with one true witness at the end. It is sufficient to prove that the above-mentioned invariant holds for all N, as then it is evident that at the end of the last iteration we would be left with 1 true witness and 0 false witnesses.
If N is **even**:
The number of true witnesses is greater than $N/2$, so invariant is initially valid. The difference between number of true and false witnesses must thus initially be 2. For conclusions corresponding to cases 2, 3 and 4, we always remove at least one false witness (either both false witnesses are removed or one true and one false witness is removed). So the difference between the number of true and false witnesses would possibly increase or remain the same in actions 2,3 and 4. So the invariant would still hold for tests ending in those conclusions, and the difference would be at least 2. In Case 1, we know both witnesses are either true or false, hence by removing one we would either remove a true witness or a false witness. The invariant evidently still holds if a bad witness is removed. Since the difference is at least 2, even if a good witness is removed the invariant would still hold. Hence for all tests, the invariant holds for even N.
If N is **odd**:
The original invariant still holds true at the beginning. After making $\lfloor N/2 \rfloor$ pairs, we are left with one witness. Same as the case for even N, we apply our actions corresponding to the conclusion of each test. Removing both witnesses in actions 2, 3, 4 does not affect the invariant as proved above. Removing a false witness in Case 1 also does not affect the invariant. When a true witness is removed under action 1, if the number of false witnesses

becomes equal to the number of true witnesses, the excluded witness must be a true witness for the invariant to hold initially as cases 2, 3 and 4 always remove more false witnesses than true. Hence adding that true witness back to the testing pool restores our invariant. If the number of true witnesses is greater than the number of false witnesses after our actions, the difference in their numbers must at least be 2 as seen in the even case. Then even if the excluded witness is false, adding them back to the testing pool will not affect the invariant. Hence for all tests, the invariant holds for even N.
Since in both case we have made $\lfloor N/2 \rfloor$ pairs and tested them pairwise, it is thus proved that in $\lfloor N/2 \rfloor$ the problem is reduced to a size less than N/2

Pseudocode

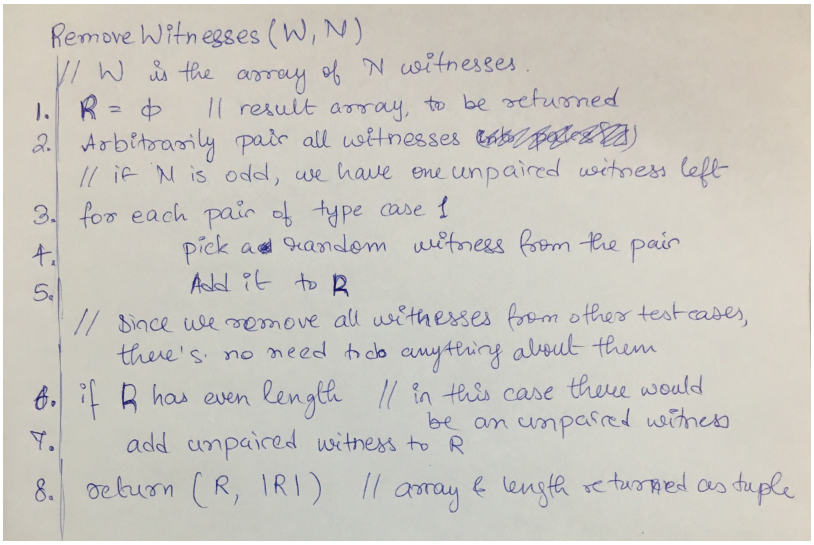


Figure 3: Pseudocode for the reducing number of witnesses

Part (c)

Pseudocode

By recursively applying the procdeure explained in part (b), we can keep reducing the number of witnesses and thus arrive at one good witness. Since at each step we must perform at most N/2 pairwise tests, each iteration runs in O(n) (assuming that each test takes constant time). The running time for N witnesses would be given as:
$T(n) \leq T(n/2) + \mathcal{O}(N)$
Hence the worst-case time complexity for finding one true witness would be given by the recurrence relation:

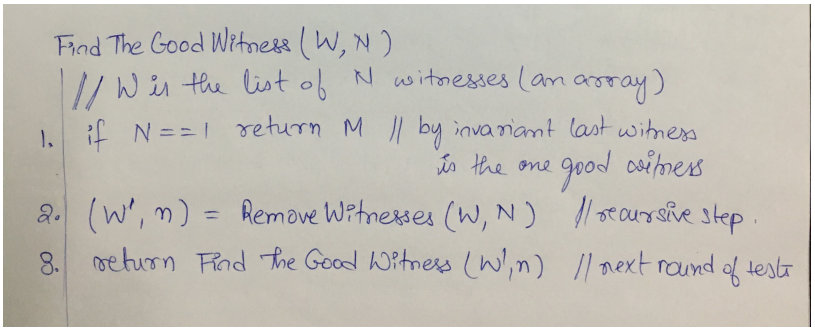


Figure 4: Pseudocode for finding one true witness

$T(n) = T(n/2) + \mathcal{O}(N)$
From the solution of Problem 1, part(b), subpart 1 (Page 3) we know that the solution to this recurrence relation is $T(n) = \Theta(n)$.
Once a true witness is found, we can pairwise test all the other witnesses with that true witnesses to find all the true witnesses. Doing so would also take $\Theta(n)$ time. Hence the number of pairwise tests, and the time taken (assuming each test takes constant time) to find all true witnesses from N witnesses is $\Theta(n)$