Tips

- To get faster run-time (in case of lots of i/o bound input) add ios_base::sync_with_stdio(false); cin.tie(null). Read on when to use;
- To get an idea of overflow solve - link
- c++ vector changes its address on reallocation so don't rely your algo on vector's pointer or iterator.
- greedy is a solution, where you never go back to improve and keep focussing on whatever is the present viable solution for each iteration.
- check overflow

```
bool check_overflow_add(int x, int y) {
  //Case x is +ve y +ve
  if (x > 0 && y > 0){
    if (x > INT_MAX - y) return true;
  }
  else if (x < 0 && y < 0){
        if (x < INT_MIN - y) return true;
  }
  return false;
  }


bool check_overflow_multiply(int x, int y) {
    if (x < 0) x = -1 * x;
    if (y < 0) y = -1 * y;
    if(x == 0 ) return false;
    if (y > INT_MAX / x) return true;
    return false;
}
```

- Questions like find the third max number (in 1,2,2,3) max is 3.
    - Use a set(bst) in c++ cause it is sorted and takes care of duplicates. link
    - If duplicates are allowed use a priority queue. link
- Sometimes bruteforce is the best solution - link
- max of 3 numbers is max(max(a,b),c);
- Sieve of Eratosthenes (To find all the prime numbers less than n). In this method, you first set the prime value ofall numbers to be true. Then, for a variable i starting from 2 to the square root of limit(the number given), you set the value of prime of all the multiples of i to be false.

```
int countPrimes(int n) {
    if(n ==0) return 0;
    if(n == 1) return 0;
    if (n == 2) return 0;
    if (n == 3) return 1;
    vector<int> v(n+1,0);
    v[0] = 1;
    v[1] = 1;
    for(int i=2;i<sqrt(n);i++){
        for(int j=i+i;j<=n;j=i+j)
            v[j] = 1;
    }
    int count =0;
    for(int i=0;i<n;i++){
        if(v[i] == 0) count++;
    }
    return count;

}
```

Now, all the numbers left whose prime value is true are prme numbers and can be displayed.

- Time complexity for a sqrt function is log(n);

```
1) Start with ‘start’ = 0, end = ‘x’,
2) Do following while ‘start’ is smaller than or equal to ‘end’.
    a) Compute ‘mid’ as (start + end)/2
    b) compare mid*mid with x.
    c) If x is equal to mid*mid, return mid.
    d) If x is greater, do binary search between mid+1 and end. In this
case, we also update ans (Note that
     we need floor).
    e) If x is smaller, do binary search between start and mid
  Note: The Binary Search can be further optimized to start with ‘start’ = 0
and ‘end’ = x/2.
  Floor of square root of x cannot be more than x/2 when x > 1.
```

- c++ strings are mutable.
- vector's iterator are bidirectional
- list's iterator can't move in ++ or --.
- umap's iterator can't move in ++ or --.
- pair is very useful when you want a container for 2 data types without writing an additional class for it.
    - First element is stored as "first"
    - second as "second"
    - You can create one as {first_int, second_int} too;

- You can insert into a unordered_map<int,pair<int,int>> as

```
umap[key] = {val1,val2};
```

- it is helpful to use _ before name for member variables.
- Use typedef for long names.
- Use vector instead of map in dealing with characters.
- Use vector instead of stack wherever possible.
    - vector.push_back (s.push())
    - vector.pop_back (s.pop())
    - vector.back() (s.top())
    - This trick is useful when dealing with string and you need to process them later. Example
- IMPORTANT -> While sorting objects in c++
    - Remember to use the following constraint/requirement. The compare requirement requires that If comp(a,b)==true then comp(b,a)==false. But if you write you it returns true if b==a thus comp(b,a) == comp(a,b) which does not fulfil the requirement.

      NOT CORRECT WAY

```
//NOT CORRECT WAY
if (da.x() > db.x())
    return false;
else
  return true;
```

      CORRECT WAY

```
//NOT CORRECT WAY
return da.x()<db.x();
```

        - Refer https://stackoverflow.com/questions/2627166/difference-between-const-reference-and-normal-parameter
        - https://en.cppreference.com/w/cpp/named_req/Compare

Tokenizing a string in c++

Using stringstream

```
```

```
string s = "Hello how are you";
stringstream ss(s);
string token;
while(getline(ss,token,' ')){
  cout<<token<<endl;
}
/*Prints->
  Hello
  how
  are
  you
*/
```

Operator Precedence in c++

- Bit operator precedence is less than equality operator.

Precedence	Operator	Description	Associativity
1	++ -- () [] . -> (*type*){*list*}	Suffix/postfix increment and decrement Function call Array subscripting Structure and union member access Structure and union member access through pointer Compound literal(C99)	Left-to-right
2	++ -- + - ! ~ (*type*) * & sizeof _Alignof	Prefix increment and decrement[note 1] Unary plus and minus Logical NOT and bitwise NOT Cast Indirection (dereference) Address-of Size-of[note 2] Alignment requirement(C11)	Right-to-left
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional[note 3]	Right-to-Left
14[note 4]	= += -= *= /= %= <<= >>= &= ^= |=	Simple assignment Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Traverse While Deleting

Vector/ String

- Use erase-remove idiom (Optimized)
    - Motivation behind this idiom is in vector/ string calling erase multiple times on the same container generates lots of overhead from moving the elements.
    - So a solution is to use remove/remove_if which do not remove elements from the container, but move all elements that don't fit the removal criteria to the front of the range, keeping the relative order of the elements. This is done in a single pass through the data range. remove returns an iterator pointing to the first of these tail elements so that they can be deleted using a single call to erase.
    - Doing the same using only erase results in as many passes as there are elements to remove. For each of these passes, all elements after the erased element have to be moved, which is more time-consuming than shifting elements in a single pass.
    - Can't be used for set

```
v.erase(std::remove_if(v.begin(), v.end(), IsOdd), v.end());
```

- Using just erase (Not optimized)

```
while(it!=v.end()){
  if(toDelete(it)){
    it = v.erase(it);
  }
  else{
    it++;
  }
}
```

Set/List

- Using just erase (Not optimized)

```
while(it!=s.end()){
  if(toDelete(it)){
    it = s.erase(it);
  }
  else{
    it++;
  }
}
```

Dynamic Programming

- Unique Path - 2
    - You can just create an additional row and column to deal with boundary case and that will simplify the implementation a lot.
- Look at the Boolean Parenthesis problem.
    - Similar Problems
    - Valid Parenthesis String
    - Matrix Multiplication
- Space can be heavily optimized in case in dp solution the present state only depends on the previous state. You only have 2 arrays and toggle between them using flag = flag^1
    - https://www.youtube.com/watch?v=UFMOzkUFEW4
    - https://leetcode.com/problems/uncrossed-lines/
- Not apparent problems
    - https://leetcode.com/problems/minimum-swaps-to-make-sequences-increasing/
        - https://leetcode.com/problems/minimum-swaps-to-make-sequences-increasing/discuss/192341/Super-Intuitive-solution(recursive-%2B-memoization)

STL

Strings

Iterators (Random Access Iterators)

- Iterator to the beginning `s.begin()`
- Iterator to the past-the-end `s.end()`
- Iterator to the i index `s.begin()+i`

- **You can perform addition in these iterators like it+3 because it is random access iterator**

Get

- Value from the index - O(1) `s[1];`
- Iterator from index - O(1) `s.begin()+i`
- Value from iterator `(*it)`
- Size `s.size()`'

- **Get iterator of a character in string `std::find(s.begin(), s.end(), 'x')` Remember it is a char not string.**

- Substring (Important)

    - `s.substr(firstIndex,length)` returns substring for [pos, pos+len).
    - `s.substr(index)` returns substring for pos to end.

Add

Single element

- In middle using the iterator(before which to insert) `s.insert(it,val)`
- There is no push front.

Multiple element

- ```
  string x = "Manish";
  string y = "sdf"
  x.insert( x.begin() + 2, y.begin(), y.end());
  ```
- Add at the end or beginning.
- Use + operator `s+x`

Erase

- To clear all the elements `s.clear()`
- To erase an element using iterator `s.erase(it)`
- To erase a range of elements using iterator `s.erase(it_first, it_last)`

- Very Important erase-remove idiom.

- Look at vector explaination.

Modify

Single Element

- `s[index] = character_element`
- **Using iterator** `(*it) = val;`

Iterators

- Iterators are used to step through the elements of collections of objects. Offer a common interface for all the container types.
- Just like template make algorithm independent of datatype, iterator make algorithm independent of container type.
- Iterator actually denote a certain position in a container.

Common Operations in iterator

- `*Iterator` -> Returns element of current position.
  ```

- `Iterator++` -> Returns next iterator
- `Iterator1 == Iterator2` or `Iterator1 != Iterator2` -> If 2 iterators points to same location.
- `Iterator =` -> assigns an iterator to the position of element it refers to.

Iterator vs Pointer

- Iterator are not same as pointer, they can be thought of a special pointer which is designed for a specific container and can iterate efficiently for that container. Their internal behaviour is defined by the container over which they iterate. In fact each container defines its iterator type as a nested class.

The logic behind the half open range of iterator i.e [begin(),end())

- We have a simple end criterion for loops that iterate over the elements: They simply march through until they meet end().
- It avoids special case handling for empty ranges, where begin() == end().

Types of Iterators (Important)

Input iterator

- The term input is used from the viewpoint of a program. Information going from the container to the program is considered input
- An input iterator is one that a program can use to read values from a container.
- It does not allow us to alter the value. So algorithms that require an input iterator are algorithms that don't modify values of the container elements.

Output iterator

- They are just like input iterator but not for accessing elements, but for being assigned elements.
- Single-pass and write-only iterator.

Forward Iterators

- They are higher in hierarachy than input and output iterators, and contain all the features present in these two iterators.
- They also can only move in forward direction and that too one step at a time.
- Unlike input and output iterators, however, it necessarily goes through a sequence of values in the same order each time we use it

Bidirectional Iterator

- they can move in both the directions, that is why their name is bidirectional.

Random Access Iterator

- They are the most powerful iterators. They are not limited to moving sequentially, as their name suggests, they can randomly access any element inside the container. They are the ones whose functionality is same as pointers.

ITERATORS	PROPERTIES				
	ACCESS	READ	WRITE	ITERATE	COMPARE
Input	->	= *i		++	==, !=
Output			*i=	++	
Forward	->	= *i	*i=	++	==, !=
Bidirectional		= *i	*i=	++, --	==, !=,
Random-Access	->,[ ]	= *i	*i=	++, --, +=, -==, + ,-	==, !=, <,>,<=,>=

CONTAINER	TYPES OF ITERATOR SUPPORTED
Vector	Random-Access
List	Bidirectional
Deque	Random-Access
Map	Bidirectional
Multimap	Bidirectional
Set	Bidirectional
Multiset	Bidirectional
Stack	No iterator Supported
Queue	No iterator Supported
Priority-Queue	No iterator Supported

Functions on iterators

- `dist(Iterator1, Iterator2)`
- `advance(Iterator1, count)`
- `prev(Iterator, count=1)` **Used in prev(container.end())**
- `next(Iterator, count=1)`

Misc

- Sort and binary search algorithm requires a random access so it can swap 2 non adjacent elements. So linkedlist iterator cannot be used. (That's why linked list provides a sort method in itslef)

- Find needs a ++ operator, it doesnot need a write access but needs a read access.
- You cannot do advance(container.begin(),12);
- You need to first assign container.begin() to a iterator and then use that iterator to advance.
- Compare iterator with a null, it is not straight forward. You need to do `it != container.end()`

Tuple

A tuple is an object that can hold a number of elements. The elements can be of different data types. The elements of tuples are initialized as arguments in order in which they will be accessed.

```
tuple <int,char,float> tup1(20,'g',17.5);
tie(i_val,ch_val,f_val) = tup1;
cout << i_val << " " << ch_val << " " << f_val;
```

Use tuple instead of creating structs

Can also use `make_tuple()` is used to assign tuple with values.

tie can also be used to unpack pair<X,Y>;

Priority Queue

- Need to `#include<queue>`
- In C++ if the element is in the form of pairs/ tuple, then by default the priority of the elements is dependent upon the first element. Very useful to know.

Containers

Vector

Iterators (Random Access Iterators)

- Iterator to the beginning `v.begin()`
- Iterator to the past-the-end `v.end()`
- Iterator to the i index `v.begin()+i`
- **You can perform addition in these iterators like it+3 because it is random access iterator**

Get

- Value from the index - O(1) `v[1];`
- Iterator from index - O(1) `v.begin()+i`
- Value from iterator `(*it)`

- Size `v.size()`'
- **Get item with a specific value** `std::find(vec.begin(), vec.end(), item)`

Add

Single element

- `v.push_back()`
- In middle using the iterator(before which to insert) `v.insert(it,val)`
- There is no push front.

Multiple element

- ```
  int myarray [] = { 501,502,503 };
  myvector.insert (myvector.begin(), myarray, myarray+3);
  ```

Erase

- To clear all the elements `v.clear()`
- To erase an element using iterator `v.erase(it)`
- To erase a range of elements using iterator `v.erase(it_first, it_last)`
- Remove from the end `pop_back()`
- There is no pop front.

- ## Very Important erase-remove idiom.

- To erase multiple elements from an vector with a certain value, it is not recommended to use for loop and erase each element. Since this is a n^2 solution.
- Instead we should use erase with stl's remove to get O(n) solution.
- The way it works is first we call remove on the entire array. Remove doesn't erase the elements from array, it simply shifts the elements with the value to the end of the vector and returns the iterator to the tail of the remaining vector(non inclusive).
- You can then call erase on the remaining part and erase by range.

- `v.erase(std::remove(v.begin(),v.end(),7), v.end())` to erase all occurances of 7 from the vector. **This is called erase-remove idiom of c++**. It is also applicable for strings.

Modify

Single Element

- `v[index] = val`
- **Using iterator** `(*it) = val;`

Multiple elements

```
```
```

- ```
  std::vector<int> first;
  first.assign (7,100); / 7 ints with a value of 100
  ```

Initialize

- Fill with a value `vector<int> v(size,val)`

Unordered Map

IMP -> YOU CANNOT CREATE A MAP OF A MAP (Map as key) (Group Anagrams) by `unordered_map<unordered_map<string,int>,int> umap();` -> You can get around by using a string representation of the map. Also holds for unordered_set

IMP -> YOU CANNOT CREATE A MAP OF A Pair of T (pair<T,T> as key) by `unordered_map<pair<int,int>,int> umap();` -> You can get around by using a string representation of the pair. Also holds for unordered_set

Iterator (Useful for looping through elements)

- **Notice that an unordered_map object makes no guarantees on which specific element is considered its first element.**
- Iterator to the beginning `v.begin()`
- Iterator to the past-the-end `v.end()`

Get

- Value from key - O(1) `v[key]`
- Iterator from key - O(1) `v.find(key)`
- Value from iterator `it->second` or `(*it).second` since map contains Pair objects.
- Key from iterator `it->first` or `(*it).first`

Add

- `umap.insert(make_pair(1,1))`

Erase

- To clear all the elements `v.clear()`
- **To erase by key** `mymap.erase ("France");`
- To erase an element using iterator `mymap.erase ( mymap.begin() );`
- To erase a range of elements using iterator `mymap.erase ( mymap.find("China"), mymap.end() );`

Modify
  ```

- `umap[key] = value`
- Using iterator `it->second = value`

List (Doubly linked list)

Iterator (Bidirectional Iterators)

- get the iterator to the head `l.begin()`
- get the iterator to the back `l.end()`

- **You cannot perform addition in these iterators like it+3 only it++ or it-- because it is bidirectional iterator.**

Get

- get the front element `l.front()`
- get the back element `l.back()`
- (*it)

- To get value at an index - Important

```
auto it = l.begin();
advance(it,index);
(*it)
```

- **To find if an element exist's in a list `std::find(listOfElements.begin(), listOfElements.end(), element);`**

Add

- To add at the front `l.push_front(val)`
- To add at the back `l.push_back(val)`

- To add in the middle `l.insert(it,val)`

Erase

- To clear all the elements `l.clear()`
- **To erase an element using iterator `l.erase(it)`**
- To erase a range of elements using iterator `l.erase(it_first, it_last)`
- To remove the last element `l.pop_back()`
- To remove the first element `l.pop_front()`
- **Removes from the container all the elements that compare equal to val. `l.remove(val)`**

- The remove is O(N) but erase is O(1) so be careful when to use what.

- remove if a condition is met `l.remove_if(condition)`

Modify

Multiple Elements

- ```
  std::list first;
  first.assign(10,100);
  ```

Useful Function

The reason these algo are the part of this class and not stl algorithm is because the iterator in list is a bidirectional iterator and stl algo like sort and reverse don't work in them.

- `l1.merge(l2)`
- `l.reverse()` -> Important
- `l.sort()` -> Important
- `l.unique()` -> Important

Priority Queue

By default min queue. If you want max queue.

```
std::priority_queue<int, std::vector<int>, std::greater<int> > q2;
```

Get

- top of the queue -> pq.top()
- size of the queue -> pq.size()

Add

- pq.push()

Erase

- pq.pop()

Stack

Get

Add

Erase
```

Modify

Ordered Map

Get

Add

Erase

Modify

Queue (Use List rather than this.)

Misc

- Difference b/w queue, list and deque
  - queue can only add at the back. and remove at front.
  - deque (Double ended queue) can add and remove at front and back.
  - list can add and remove anywhere.
  - A list has to allocate memory every single time something is added, and deallocate it when it goes away.
  - A deque, on the other hand, allocates in chunks. It will allocate less often than a list. Think of it as a list, but each memory chunk can hold multiple nodes.
- forward_list is a singly linked list. where as list is a doubly linked list.
- 1 and 7 are the only happy numbers.

Upper Bound, Lower Bound and Binary Search

For a simple binary search always use the below structure.

```
int start = start_of_search_range;
int end = end_of_search_range;
while(start<=end){  //For normal binary search you need to use start<=end else it
will be
//complicated.
  int mid = start + (end-start)/2; //To prevent overflow
  if(x==mid) return mid;
  else if(x>mid) start = mid+1; //Answer is atleast bigger than mid;
  else end = mid -1;  //Answer is atleast smaller than mid;
}
return -1; //Can't find x
```

For Lower Bound (First element greater or equal to the given element).

```
int start = start_of_search_range;
int end = end_of_search_range + 1 //### Important cause we also need to take care
of case
//where the required answer is more than last item in the array.
while(start<end) // You can use start<= end but then you need to write condition
// for the return value.
{
  int mid = start + (end-start)/2;
  if(mid<x){
    start=mid+1;//If x is greater than mid. The search space cannot include mid
    // and is higher than mid.
  }
  else{
    end = mid;//If mid is equal or greater than x, then elements to right of mid
are
    //not the first since, mid is already greater or equal.  But mid can be the
upper bound.
  }
}
return start; //Takes care of case where lower bound isn't even present in array/
```

For upper Bound (First element greater than the given element)

```
int start = start_of_search_range;
int end = end_of_search_range + 1 //### Important cause we also need to take care
of
// case where the required answer is more than last item in the array.
while(start<end) // You can use start<= end but then you need to
//write condition for the return value.
{
  int mid = start + (end-start)/2;
  if(mid<=x){
    start = mid+1;//If x is greater than or equal to mid. The search space cannot
include
    //mid and is higher than mid.
  }
  else{
    end = mid; //If mid is greater than x, then elements to right of mid are not
    //the first since, mid is already greater than x. But mid can be the upper
bound.
  }

}
return start; //Takes care of case where upper bound isn't even present in array
```

Note: In both Upper and lower bounds, the high index is set to end_of_search_range+1 instead of end_of_search_range. These functions can return an index which is one beyond the

bounds of the array. I.e., it will return the size of the array if the search key is not found and it is greater than all the array elements.

Floyd's Tortoise and Hare (Cycle Detection)

- Used to detect a cycle in linked list.

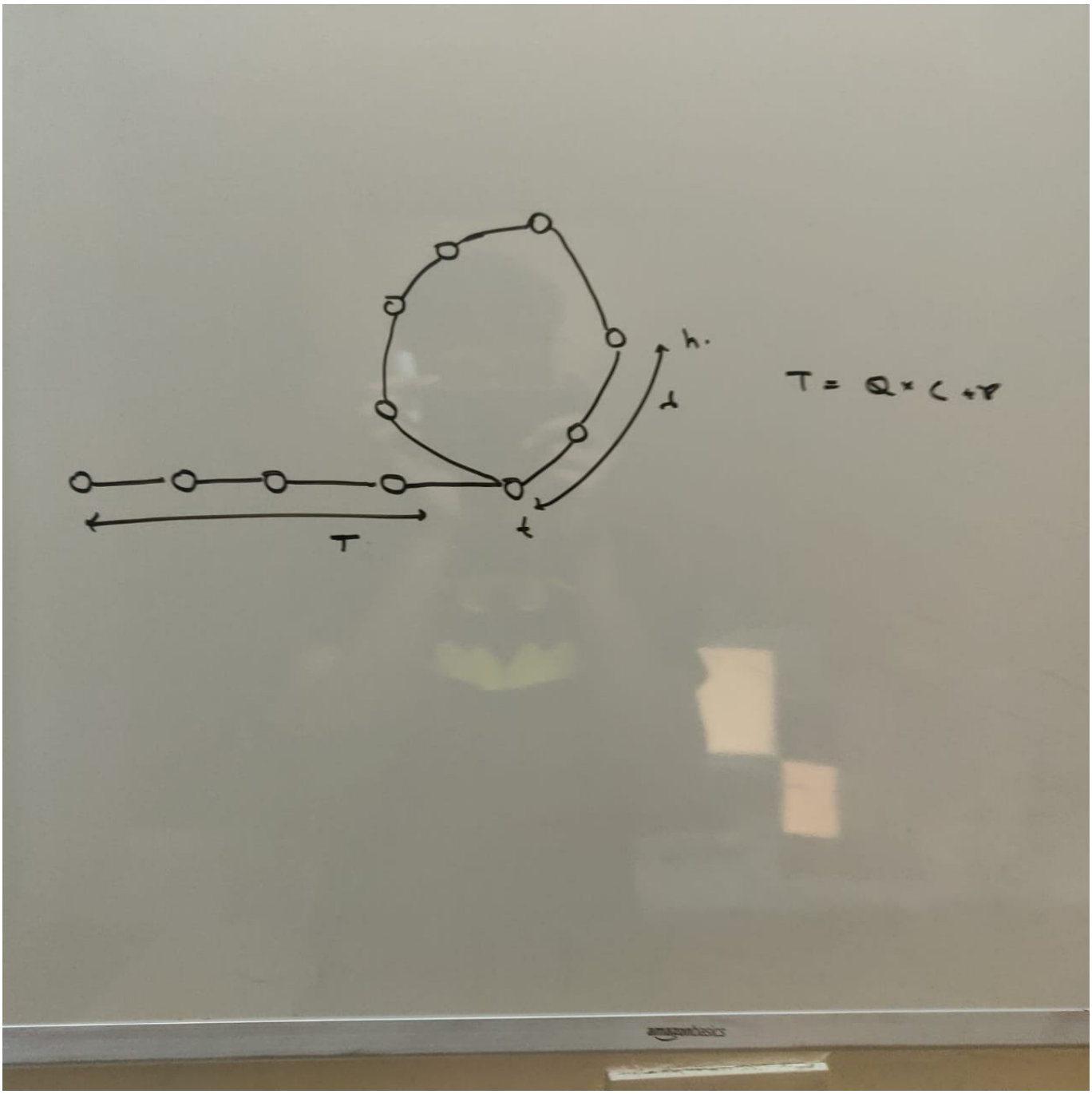
Theorem -> If you have a hare and tortoise both starting at first node in the linked list. hare moving at 2(node unit) times the speed of tortoise. They both eventually meet.

Physical Proof

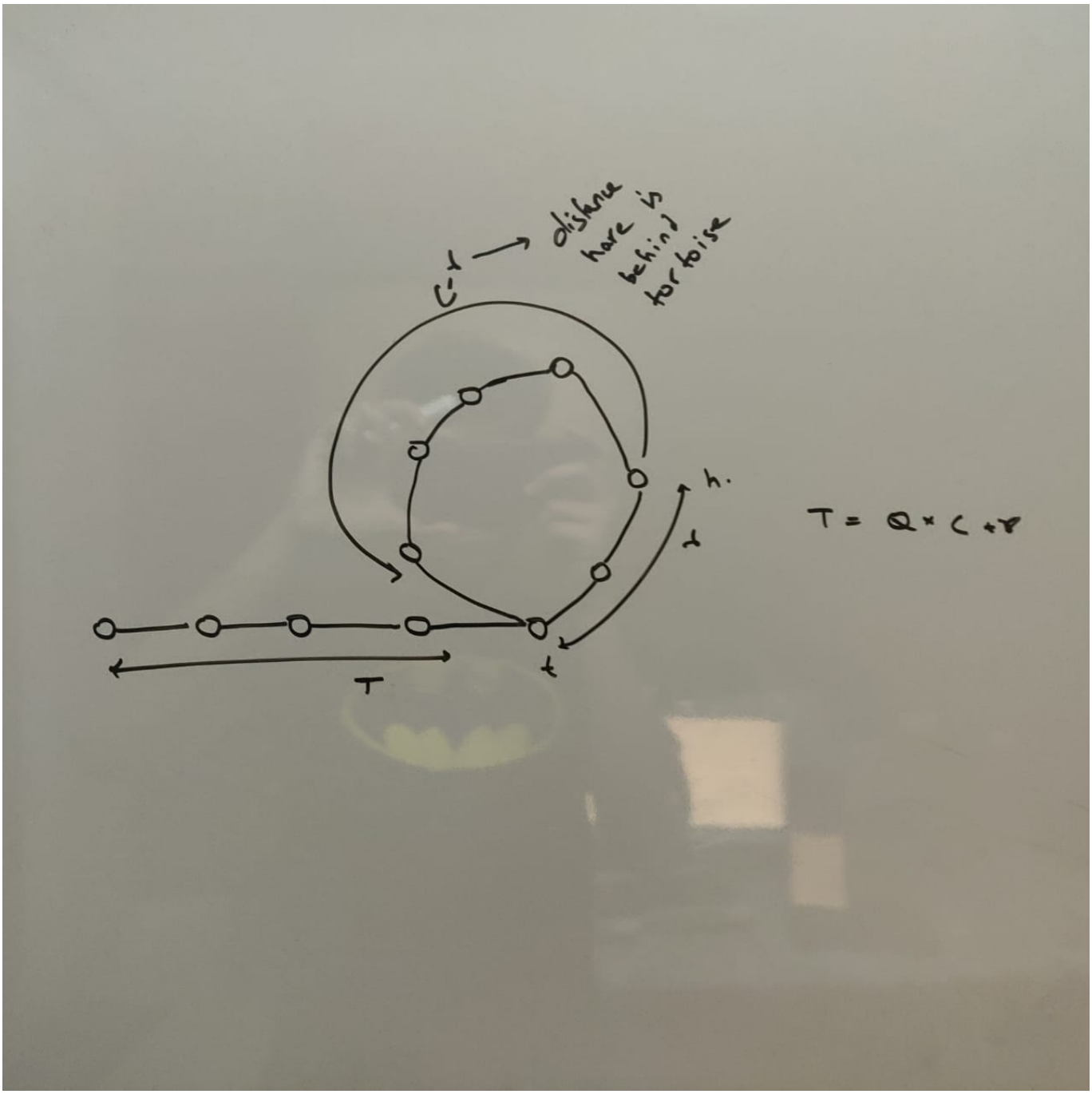
For showing that they eventually must meet, consider the first step at which the tortoise enters the loop. If the hare is on that node, that is a meeting and we are done. If the hare is not on that node, note that on each subsequent step the distance the hare is ahead of the tortoise increases by one, which means that since they are on a loop the distance that the hare is BEHIND the tortoise decreases by one. Hence, at some point the distance the hare is behind the tortoise becomes zero and the meet.

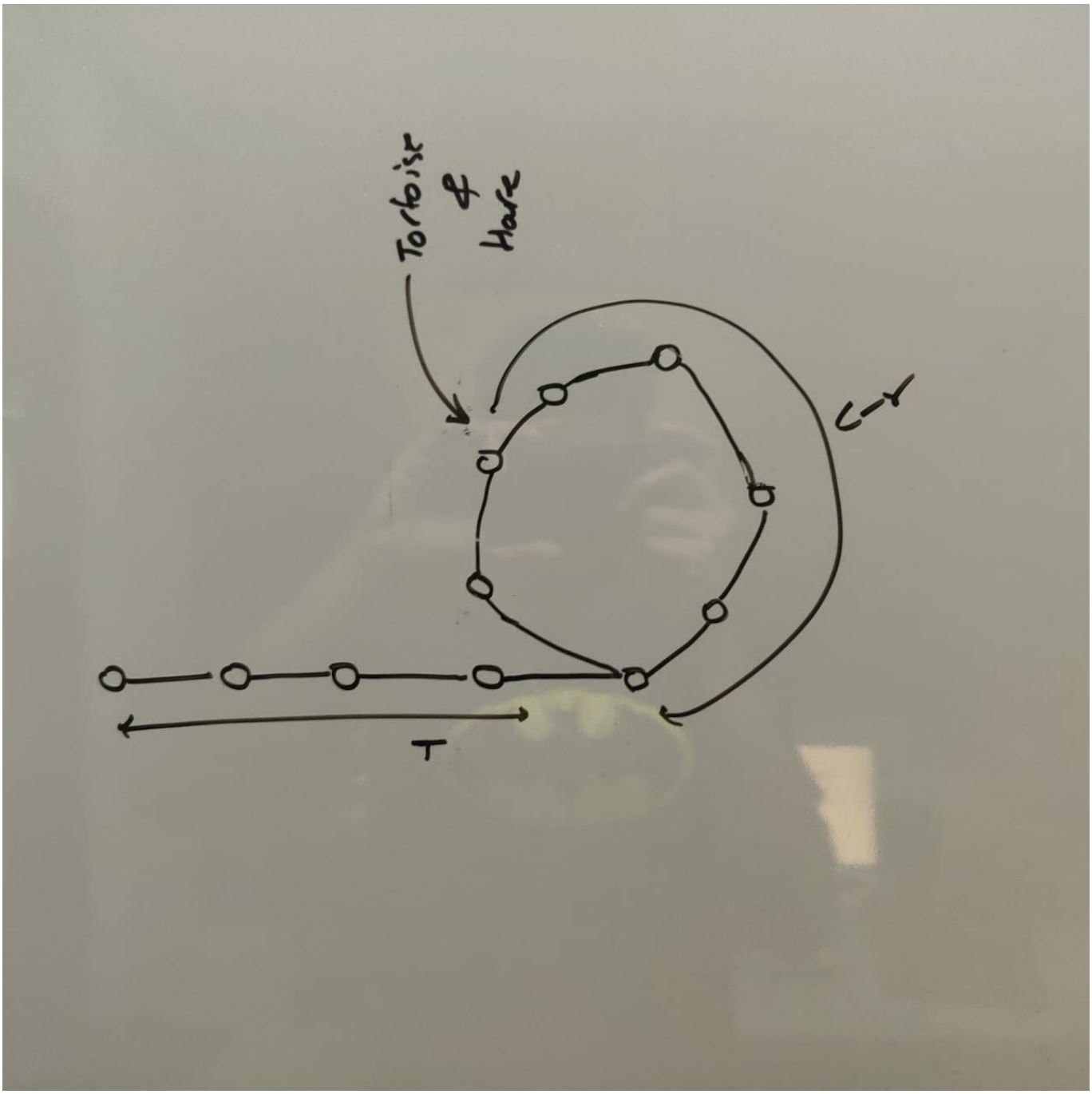
Mathematical Proof

Let's assume T is the length of the tail part of the linked list, and C is the length of cycle.
If we use division algorithm, T = QC + *r (Q->Quotient and r is remainder). After T time units, the tortoise is at the start of the cycle and hare is at r (Since T=QC* +r and the first QC didn't amount to anything and it made hare come back to start of loop, the remaining r is it's position).



Now at this position, you can infer hare is r ahead of the tortoise, but since it is a loop, tortoise is C-r nodes ahead of hare, Now since the distance between hare and tortoise decrease by 1 in each time step, they will eventually meet, after C-r time steps. In the mean while the tortoise would have travelled C-r steps. Thus they both meet at C-r'th node from the start of loop.

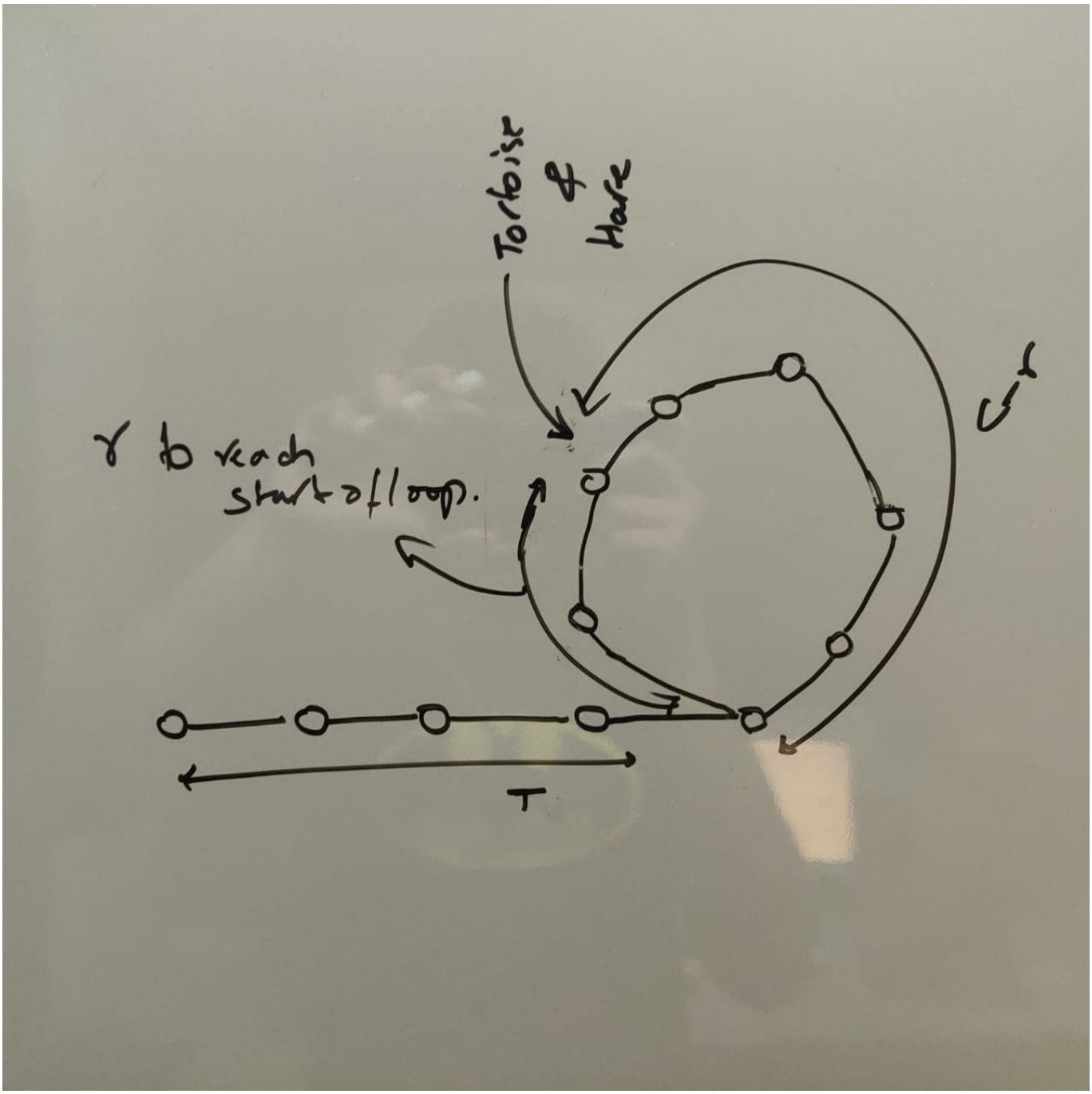

C-r
distance hare is behind tortoise
r
h.
T
t
T = Q × C + R



Theorem -> If from the point they meet, another tortoise starts from the first node, the old and new tortoise meet at the start of the loop.

Mathematical Proof.

Now, let's say we are at the time hare and tortoise meet, from that point the remaining distance to get the start of the loop would be r for the old tortoise.(It is at point C-r and advancing r steps brings it to the start of the loop). **So r is the distance required by the old tortoise to reach the start of the loop.**



Now let's say the new tortoise travelled T distance from start of linked list to start of loop. Now since both are tortoise, even the old tortoise must have travelled T distance. **T can also be represented as Q*C* +*r. But the Q*C part doesn't amount to anything as the old tortoise just reaches the same point it already is. So in essense the old tortoise travelled r distance along the cycle**, while the new tortoise travelled T distance to come to start of loop. But r is exactly the same amount of distance the old tortoise needed to reach the start of the loop. **So when they eventually reach at the same point it is at the start of the loop**.

References

- https://math.stackexchange.com/questions/913499/proof-of-floyd-cycle-chasing-tortoise-and-hare

Brian Kernighan's Counting Algorithm

- It is used to detect the number of set bits in an integer.

- If we had an operator which can unset the right most bit on each iteration. We can count the number of set bit as follows

```
int ans =0;
while(n){
  n=operator(n);
  ans++;
}
return ans;
```

- If we think about it, subtracting one from a number, unsets the rightmost bit and toggles/set every other bit to right of the this bit. If we can modify this operation such that the toggling effect is gone, we have our desired operator. If we AND the subtraction with the original number itself, we have that desired effect.

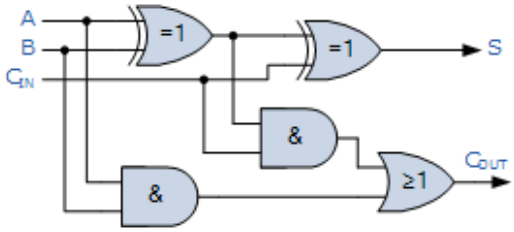
```
int ans =0;
while(n){
  n&=n-1;
  ans++;
}
return ans;
```

- If one is using c++ they can also use __builtin_popcount(n) to return the number of set bits.

For Bit questions it can be useful to know how to convert the truth table to logical circuit.

Say for instance we have to add 2 bits, with carry.
Truth Table Looks like.

Full Adder Truth Table with Carry

Symbol	Truth Table				
	C-in	B	A	Sum	C-out
	0	0	0	0	0
	0	0	1	1	0
	0	1	0	1	0
	0	1	1	0	1
	1	0	0	1	0
	1	0	1	0	1
	1	1	0	0	1
	1	1	1	1	1

Let us find the logical circuit for Carry Out. You need to just consider the cases where C-Out is 1, i.e A*B*(!C) + A*(!B)*C* + *ABC* + *(!A)BC* = *A*B + (A^B)*C

Similarly you can do for any truth table.

Important Questions

- LRU Cache
    - Use a hash<int,pair<int,list>> and a queue(i.e a list).
    - Use touch function to rearrange cache when necessary.
- https://leetcode.com/problems/jump-game/solution/
    - Read this article after solving the jump game question.
- https://leetcode.com/problems/binary-tree-maximum-path-sum/
    - Solution is elegant few lines.
- https://leetcode.com/problems/unique-email-addresses/submissions/
    - Solve this to get clarity of strings erase-remove idiom.
- https://leetcode.com/problems/majority-element/

    - Teaches the Boyer-Moore Voting Algorithm
    - Boyer-Moore Voting Algorithm
        - Explaination https://leetcode.com/problems/majority-element/solution/
        - For Diagram
- https://leetcode.com/problems/find-the-town-judge/
    - Teaches importance of
- https://leetcode.com/problems/cinema-seat-allocation/
    - Importance of looking at constraints and also bit masking
- https://leetcode.com/problems/cousins-in-binary-tree/
    - Important in knowing the tips for trees
    - https://leetcode.com/problems/cousins-in-binary-tree/discuss/238624/C%2B%2B-level-order-traversal
    - Tip - 1 -> Using nullptr to separate siblings from cousins.
    - Using 2 queues to keep track of levels.
- Boolean Parenthesis DP.
    - Different way of thinking bottom up.(Diagonal)
- https://leetcode.com/problems/flood-fill/
    - You don't need visited set.
- **IMPORTANT https://leetcode.com/problems/maximum-sum-circular-subarray/**
    - Tricky Question (Simple if you know solution).
- https://leetcode.com/problems/valid-parenthesis-string/
    - 3 Solution approaches
        - **2 Stack (best to think and implement). Link**
        - DP (Bad timecomplexity but good to know approach similar to boolean parenthesis). Link
        - **Simple solution (most efficient, hard to think like this) Link**
- https://leetcode.com/problems/remove-k-digits/
    - Teaches efficient processing of strings.
    - trimming zeros and all.
    - Use vector instead of stack.
- https://leetcode.com/problems/bulb-switcher-iii/
    - Beautiful solution. Simple straight.
- https://leetcode.com/problems/online-stock-span/

    - Couldn't solve
    - Simple solution

- https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

    - Beautiful Simple solution

- https://leetcode.com/problems/contiguous-array/submissions/

    - Couldn't Solve.

- https://leetcode.com/problems/counting-bits/

    - Very beautiful solution.
    - https://leetcode.com/problems/counting-bits/discuss/656474/C%2B%2B-simple-in-3-liner.... **There is a meaning behind the solution, not just pattern matching.**

- https://leetcode.com/problems/k-closest-points-to-origin/

    - Teaches the power of stl.
    - Important concept in sorting.
    - https://leetcode.com/problems/k-closest-points-to-origin/discuss/220235/Java-Three-solutions-to-this-classical-K-th-problem. **IMPORTANT CONCEPT WHAT IF ONLINE DATA**

- https://leetcode.com/problems/two-city-scheduling/

    - Took too much time
    - easy solution.

- If we have a stream of strings and we need to check if it is a subsequence of a particular string.

    - Followup of https://leetcode.com/problems/is-subsequence/

- https://leetcode.com/problems/find-the-duplicate-number/

    - Could not solve
    - Good concepts.
    - https://leetcode.com/problems/find-the-duplicate-number/discuss/72844/Two-Solutions-(with-explanation)%3A-O(nlog(n))-and-O(n)-time-O(1)-space-without-changing-the-input-array
    - Look at his comment -> StefanPochmann

- https://leetcode.com/problems/word-search-ii/

    - Time Limit Exceeded
    - Great solution
    - https://leetcode.com/explore/challenge/card/june-leetcoding-challenge/543/week-5-june-29th-june-30th/3376/discuss/59841/My-AC-very-clean-C++-code

- https://leetcode.com/problems/prison-cells-after-n-days/

    - Was able to solve
    - But the answer is bit tricky, I was able to solve with a very nice approach to problem.
    - See personal submission.

- https://leetcode.com/problems/subsets/
    - Was able to solve using backtracking.
    - Good concepts.
    - Elegant bit manipulation solution -> https://leetcode.com/problems/subsets/discuss/27278/C%2B%2B-RecursiveIterativeBit-Manipulation
- https://leetcode.com/problems/maximum-width-of-binary-tree/
    - One test case was failing due to overflow.
    - Could not solve using dfs, use bfs. https://www.youtube.com/watch?v=le-ZZSQRebw
    - To circumvent the overflow error, at each level you should know the leftmost index, which is not straightforward with dfs.
- https://leetcode.com/problems/3sum/
    - Important Question in interviews
    - Was able to solve with TLE.
    - 2 Approaches, convert to 2Sum(Works) or use hash map(TLE).
    - https://www.youtube.com/watch?v=Ca7k53qcTic&t=1043s
    - Everyone of the iterator should be incremented if same.
- https://leetcode.com/problems/reverse-words-in-a-string/
    - Couldn't solve the inplace solution
    - https://leetcode.com/problems/reverse-words-in-a-string/discuss/47840/C%2B%2B-solution-in-place%3A-runtime-O(n)-memory-O(1)
    - Teaches the good use of stringstream of non-inplace solution
- https://leetcode.com/problems/single-number-iii/
    - Couldn't solve, constant space
    - Remember bit's order precidence is less than equality.

To-Do

Backlog

- Prefix Sum Array Concept - Rachit Jain.
- Buy and sell stocks part 2
- Kth Smallest Element in a BST followup.
- Largest Divisible Set
- Cheapest Flights within K stops followup
- Longest Duplicate Substring
- Count Square Submatrices with All Ones O(n*m) solution.
- Dungeon Game
- Single Number and similar questions discussion.
- Find the duplicate number using floyd cycle detection.

- Subset II
- To find the time complexity in backtracking problems like Subset.

Monotone Queue

- https://medium.com/@gregsh9.5/monotonic-queue-notes-980a019d5793
- Minimum Cost Tree From Leaf Values
- Sum of Subarray Minimums 907
- Online Stock Span 901
- Score of Parentheses 856
- Next Greater Element II 503
- Next Greater Element I 496
- Largest Rectangle in Histogram 84
- Trapping Rain Water 42
- Daily Temperatures 739

Graph Questions

Dynamic Programming Problems

Trees

- Construct Binary Search Tree from Preorder Traversal (Check O(N)) Solutions

Interview Questions

- Min(Subset) + Max(subset) == k or < k
    - https://leetcode.com/discuss/interview-experience/637356/amazon-apple-facebook-l5-ict4-e5-seattle-april-2020-may-2020-offer-offer-offer
    - https://leetcode.com/discuss/interview-question/268604/Google-interview-Number-of-subsets
    - https://leetcode.com/discuss/interview-question/275785/facebook-phone-screen-count-subsets
    - https://leetcode.com/discuss/interview-question/747879/de-shaw-online-assessment-questions-2020
- Is Split Possible in an array
    - https://leetcode.com/discuss/interview-experience/637356/amazon-apple-facebook-l5-ict4-e5-seattle-april-2020-may-2020-offer-offer-offer
    - no constraints
    - splitted array has to be equal in size.
    - geek for geek variations.
- Merge K Sorted Arrays

Lists

- https://docs.google.com/document/d/1wUCqhVHydWiDk6FJdFLSMpgigNrGcs4OFZg0Wa7JGEw/edit#
- https://leetcode.com/discuss/general-discussion/691825/binary-search-for-beginners-problems-patterns-sample-solutions

- https://leetcode.com/discuss/general-discussion/458695/Dynamic-Programming-Patterns
- https://leetcode.com/discuss/general-discussion/655708/graph-problems-for-beginners-practice-problems-and-sample-solutions
- https://leetcode.com/problems/subsets/discuss/27281/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partitioning)

Concepts

- https://stackoverflow.com/questions/28138188/why-pass-by-value-and-not-by-const-reference
- https://medium.com/@gregsh9.5/monotonic-queue-notes-980a019d5793
- https://en.wikipedia.org/wiki/
- Pigeonhole_principle
    - https://leetcode.com/problems/find-the-duplicate-number/discuss/72844/Two-Solutions-(with-explanation)%3A-O(nlog(n))-and-O(n)-time-O(1)-space-without-changing-the-input-array
    - Look at his comment -> StefanPochmann
- Boyer-Moore Voting Algorithm
- Brian Kerighan's Counting Algorithm
- Floyd's Tortoise and Hare (Cycle Detection)
    - https://www.quora.com/How-does-Floyds-cycle-finding-algorithm-work-How-does-moving-the-tortoise-to-the-beginning-of-the-linked-list-while-keeping-the-hare-at-the-meeting-place-followed-by-moving-both-one-step-at-a-time-make-them-meet-at-starting-point-of-the-cycle/answer/Brian-Quanz?ch=3&share=75229003&srid=bQkw