Todays Content:

1. longest substring with all distinct characters
2. Check if 2 strings are permutations of each other

1B length of longest substring with all distinct characters?

```
    0 1 2 3 4 5 6 7
S1 = a b c a b c d d   ans = 4

    0 1 2 3 4 5 6 7
S2 = s i p p i e r g   ans = 5

    0 1 2 3 4
S3 = a a a a a   ans = 1
```

Idea1: Generate all substrings:

for every substring [i..j], check if all char's are distinct

Insert all chars in a hashset hs.

```
if ( hs.size() == j-i+1) { # substring [i..j] is unique
    ans = max(ans, j-i+1);
}
```

TC: O(N^3) SC: O(N)

Idea2: Generate all substrings: { Better than 2B, use this explanation }

for every substring [i..j], Insert new char in hashset

```
if ( hs.size() == j-i+1) { # substring [i..j] is unique
    ans = max(ans, j-i+1);
}
```

TC: O(N^2) SC: O(N)

Ideas:

	0	1	2	3	4	5	6	7	8
s_1 =	a	b	c	a	b	c	d	e	e

Target: length of longest substring with distinct chars

Search Span: l : 0 r : N

Discard: T T T T .. T F F F .. F

if substring of m length contains all distinct characters ✓
Substring of m-1, m-2.. will also contain distinct chars

... m-2 m-1 m

ans = m; l = m+1

if substring of m length cannot contain all distinct characters
Substring of m+1, m+2 will also not contain distinct chars

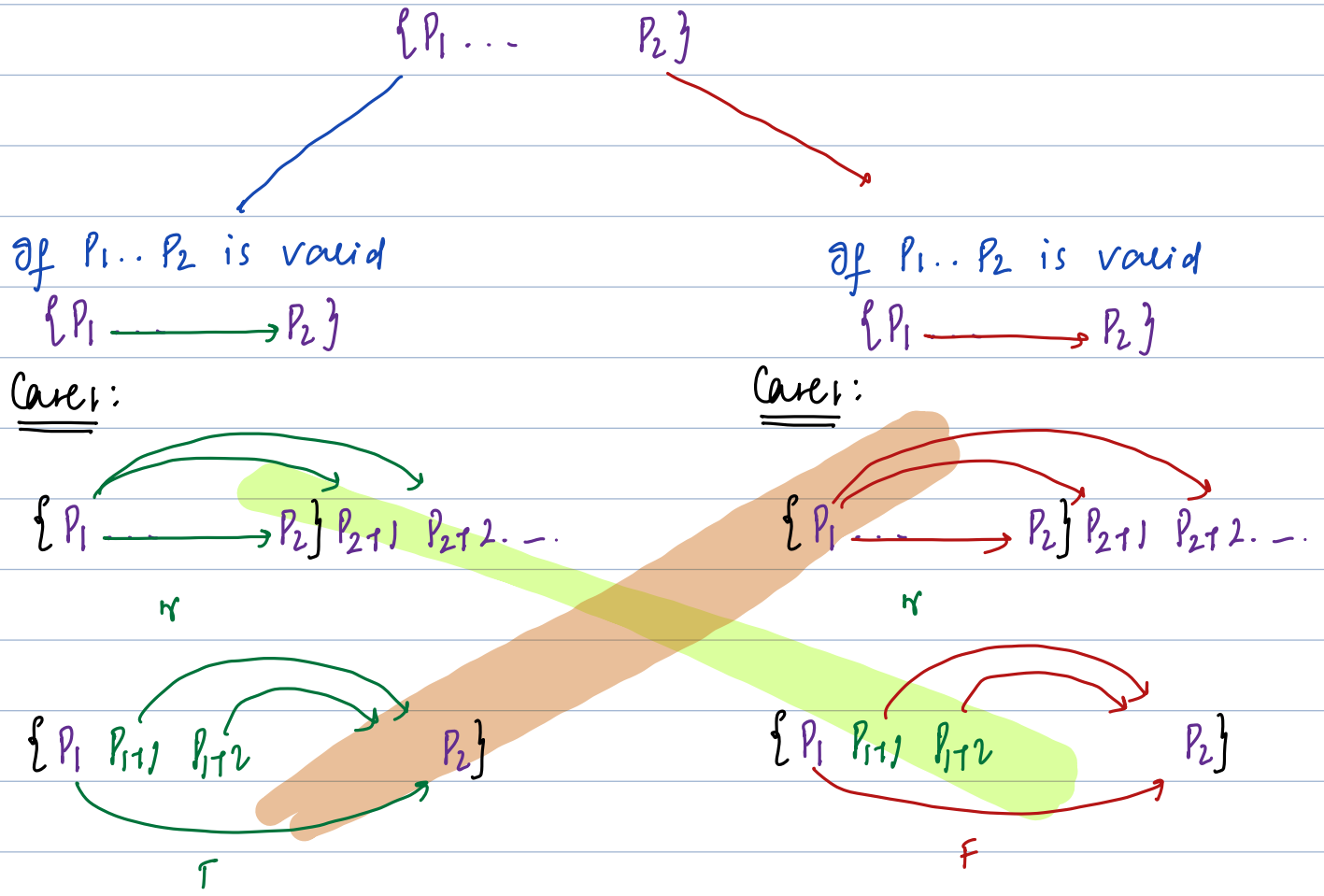
m m+1 m+2..

r = m-1;

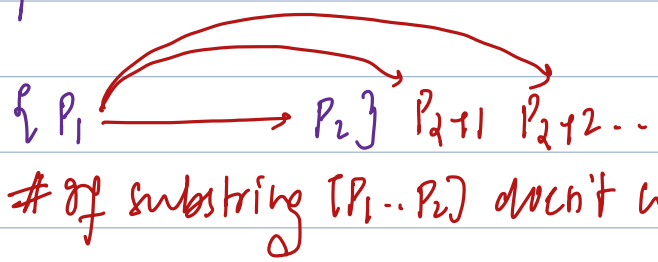
```
int longest(String s){ TC: O(NlogN) SC: O(N)
    int l=0, r=s.length();
    int ans=0;
    while(l<=r){
        int m=(l+r)/2;
        if(check(s,m)){ # In string check if there exists a m length substring with all distinct characters
            ans=m;
            l=m+1;
        }else{
            r=m-1;
        }
    }
    return ans;
}
```

TODO: Sliding window TC: O(N)

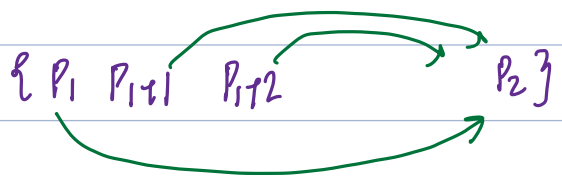
#If we want to apply 2 pointer in subarray, show me of the below



#For above problem.



# of substring [P1..P2] doesn't contain distinct elements



# of substring [P1..P2] contains distinct elements

Idea: 0 1 2 3 4 5 6 7 8 9 10 11 12 13

s = ~~a~~ b a g h c g k l m h a b k

a P_1 c P_2

↳ if substring size == this.size()

#Note: In hashset if we remove element, it will remove all it's occurences, so prefer hashmap

P_1	P_2	Valid	ans = 0	this: { ~~a~~ b c g h }
0	-1	✓	0	P_2++, Insert s[P_2] in this
0	0	✓	1	P_2++, Insert s[P_2] in this
0	1	✓	2	P_2++, Insert s[P_2] in this
0	2	✓	3	P_2++, Insert s[P_2] in this
0	3	✓	4	P_2++, Insert s[P_2] in this
0	4	✓	5	P_2++, Insert s[P_2] in this
0	5	*		Remove ar[P_1] & P_1++;

Idea: 0 1 2 3 4 5 6 7 8

s = ~~a~~ ~~b~~ a c h c g k l

P_1 P_2

Using HashMap

Note: If freq of ele == 0, remove it, so that it won't affect size.

↳ if substring size == hm.size()

P_1	P_2	Valid	ans = 0	hm: { c:2 a:1 b:1 }
0	-1	✓	0	P_2++, Insert s[P_2] in hm
0	0	✓	1	P_2++, Insert s[P_2] in hm
0	1	✓	2	P_2++, Insert s[P_2] in hm
0	2	✓	3	P_2++, Insert s[P_2] in hm
0	3	*		Delete s[P_1] in hm & P_1++
1	3	✓	3	P_2++, Insert s[P_2] in hm
1	4	✓	4	P_2++, Insert s[P_2] in hm
1	5	*		Delete s[P_1] in hm & P_1++
2	5	..	TODO	

```
int longestString s){ TC: O(N) SC: O(N)
    int p1=0, p2=-1, ans=0, N= s.length();
    unordered_map<char,int> um;

    while( p2<N){
        if( p2-p1+1 == um.size()){ # substring [p1 ... p2] contains distinct
            ans = max(ans, p2-p1+1);
            p2++;
            if p2==N){ break;}
            um[s[p2]]++;
        }
        else{ # substring [p1 ... p2] contains duplicates
            um[s[p1]]--;
            if( um[s[p1]]==0){
            } um.erase(s[p1]);
            p1++;
        }
    }

    return ans;
}
```