//reconstruct BST from preorder

```
int index;
template<class T>
void reconstructBST(shared_ptr<node<T>> & curr,T preorder[],int maxAllowed,int size){

	if(index==size)
		return;

	if(preorder[index]<curr->value){ //go left
		curr->left=make_shared<node<T>>(preorder[index]);
		index++;
		reconstructBST(curr->left,preorder,curr->value,size);
	}
	if(preorder[index]>curr->value && preorder[index]<maxAllowed){ //go right
		curr->right=make_shared<node<T>>(preorder[index]);
		index++;
		reconstructBST(curr->right,preorder,maxAllowed,size);
	}
}
```

Bst property

```
template<class T>
bool checkBSTproperty(shared_ptr<node<T>> & curr,int maxAllowed){

	//to break BST property

	bool ans=curr->value <maxAllowed;

	if(curr!=nullptr){
		if(curr->left!=nullptr){
			ans=ans && (curr->left->value < curr->value) &&
checkBSTproperty(curr->left,curr->value);
		}
		if(curr->right!=nullptr){
			ans=ans && (curr->right->value > curr->value) &&
checkBSTproperty(curr->right,maxAllowed);
		}

	}

	return ans;
```

```
}
```

binary tree LCA

```
//find LCA of two nodes a & b
//Algorithm: if both a&b are in the left subtree, then LCA is twoo (same for the right subtree)
template<class T>
shared_ptr<node<T>> find_LCA(shared_ptr<node<T>> & curr,shared_ptr<node<T>>
a,shared_ptr<node<T>> b){
	if(curr==nullptr)
		return nullptr;
	else{

		//check if current is one of the nodes a or b. Also works for the case that A is LCA (or b)
		if(curr->value==a->value)
			return a;
		if(curr->value==b->value)
			return b;


		shared_ptr<node<T>> leftT=find_LCA(curr->left,a,b);
		shared_ptr<node<T>> rightT=find_LCA(curr->right,a,b);

		//if not null from right and left: curr is LCA
		if(leftT!=nullptr && rightT!=nullptr){
			return curr;
		}

		return leftT==nullptr?rightT:leftT; //return the one that's not null

	}

}
```

Rabin-Karp string matching

```
int main(){

	string pattern="ababac";
	string text=" ababac hey bacman itsaba fer andababac fersarr ababac to ababac";

	int m=pattern.length();
	int n=text.length();

	int kBase=27; //26 letters in english alphabet +1 for space. Used to get numeric value of a string. "122" -> 122
	int kMod=997; //big prime number;

	int power=1;
	int p_hash=0;
	int t_hash=0;

	//compute the hash for the pattern and for the first window (substring of length m) in text string
	for(int i=0;i<m;i++){
		power=i>0?power*kBase:1; //to get 26^(M-1) -> used when deleting most significant character
		p_hash=(p_hash*kBase+pattern[i])%kMod; //kMod multiplies ALL TERMS (increasing their exponent) in previous hash each time
		t_hash=(t_hash*kBase+text[i])%kMod;
	}

	//try to find the pattern inside the text

	for(int i=m;i<n;i++){
		if(p_hash==t_hash ){ //only if they have same hash, compare strings
			if(pattern==text.substr(i-m,m)){
				cout<<"Match found at "<<i-m<<endl;
			}
		}


		//Rolling hash technique:
		//remove the Most Significant Char and add a new Least Significant.
		t_hash-=(power*text[i-m])%kMod; //remove MSC
		if(t_hash < 0)
			t_hash+=kMod;
		t_hash=(t_hash*kBase+text[i])%kMod; //add LSC and multiply hash by kBase to increment exponents by one after removing the MSB in previous step

	}

	//LAST CASE: check if pattern is last substring in text
	if(t_hash == p_hash && text.substr(n-m,m)==pattern)
		cout<<"Match at END "<<n-m<<endl;
```

Interview questions - print

Quicksort

```
//partition the array so that the left part is LESS than pivot and right part is MORE than pivot.
//this way, pivot is already in it's correct sorted position in the array
//a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi]

int partition(int array[],int lo,int hi){
	int  pivot=array[lo];

	while(lo<hi){

		//find item on low to swap (bigger than pivot)
		while(array[lo]<pivot)
			lo++;

		//find item on hi to swap (smaller than pivot)
		while(array[hi]>pivot)
			hi--;

		if(array[lo]==array[hi]) //for duplicate items, if we dont have this -> goes forever
			lo++;

		if(lo<hi)
			swap(array,lo,hi);
		else
			break;

	}

	return hi;
}


void quicksort(int array[],int lo,int hi){
	if(hi<=lo)
		return;
	int j=partition(array,lo,hi);

	quicksort(array,lo,j-1); //sort left partition
	quicksort(array,j+1,hi); //sort right partition

}
```

List : find cycle length (L) and start point(A)

```
template<class T>
shared_ptr<node<T>> findCycle(shared_ptr<node<T>> const & head){

	shared_ptr<node<T>> slow=head,fast=head;

	int cycle=1;
	bool hasCycle=false;

	while(slow->next!=nullptr && fast->next!=nullptr &&fast->next->next!=nullptr){
		slow=slow->next;
		fast=fast->next->next;
		if(slow==fast){

			hasCycle=true;

			slow=slow->next;
			while(slow!=fast){ //get cycle length
				cycle++;
				slow=slow->next;
			}

			//move fast from head cycle_length steps. cycle_start-fast=cycle_start-head
whatever we are shy of cycle start, we have put advanced before the cycle
			fast=head;
			slow=head;
			while(cycle--){
				fast=fast->next;
			}
			//advance both one at a time. Whereevery they meet, that's cycle_start
			while(slow!=fast){
				slow=slow->next;
				fast=fast->next;
			}

			int stepsToMeet=0;

			break;

		}
	}


	return hasCycle?slow:nullptr;
}
```