Understanding Time Complexities and using them in constraints to estimate an approach

Time and Space complexities are really really important topics.

We will not discuss space complexity in much but rather about time complexity. From L1 pdf you already know what three notations are.

For layman language let's just say our time complexity is actually time proportionality
As we all have studied physics already.
Let's see some equations

1. $s = ut + \frac{1}{2}at^2$
   Now here t^2 is most important because we know that if we have to draw a graph for this, it will be parabola. That explains that $s \propto t^2$
2. $v = u + at$
   This explains that if a graph is drawn it will be straight line.
   So $v \propto t$

Similarly algorithms as explained in L1 have formula such as this $T(n) = nt_1 + nt_2 + n^2t_3 + ...$,
Note that here again the highest degree is responsible for graph plotting.
So we can write this as $T(n) \propto n^{highest-degree}$

Lets see some examples and there time complexities
1.

```
for(int i = 0; i < n; ++i){
    for(int j = 0; j < n; ++j){
        //SOME CODE
    }
}
```

Here if SOME CODE take t constant time to perform
then $T(n) = n \times n \times t = n^2t \implies T(n) \propto n^2$
Thus time complexity is $O(n^2)$
2.

```
for(int i = 0; i < n; ++i){
    for(int j = i; j < n; ++j){
        //SOME CODE
    }
}
```

Here if SOME CODE take t constant time to perform
then
for $i = 0$, j runs n times
for $i = 1$, j runs $n - 1$ times
for $i = n - 1$, j runs 1 times
Therefore $T(n) = n + n - 1 + n - 2 + ... + 1 = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$
Thus time complexity is $O(n^2)$
3.

```
for(int i = 0; i < n; ++i){
    for(int j = i; j > 0; j/=2){
        //SOME CODE
    }
}
```

Here if SOME CODE take t constant time to perform
then
for $i = 0$, j runs 0 times
for $i = 1$, j runs 1 times
for $i = 2$, j runs 2 times
for $i = n - 1$, j runs $log_2(n)$ times
Thus time complexity is $O(nlog(n))$

4.

```
for(int i = 0; i < n; ++i){
    for(int j = 0; j < m; ++j){
        //SOME CODE
    }
}
```

Here if SOME CODE take t constant time to perform
Thus time complexity is $O(n \times m)$
5.

```
for(int i = 0; i < n; ++i){

}
```

Here if SOME CODE take t constant time to perform
Thus time complexity is $O(n)$
6.

```
//BINARY SEARCH
int low = 0;
int high = n-1;
while(low < high){
    int mid = (low+high)/2;
    if(arr[mid] == val)return mid;
    if(arr[mid] > val)high = mid-1;
    else low = mid+1;
}
return -1;
```

Thus time complexity is $O(log(n))$

Using constraints with knowledge.

Whenever we come up with a solution, its is a classical algorithm, we will know its time complexity already, otherwise it won't be hard to complexity of code we are designing.
But better approach is to think to maximum suitable complexity and design answer according to that.
Let's see for example

1. Time Limit $= 1\ second$
   $N = 10^6$
   $TestCases = 10$
   So in $1\ second$ we are allowed to perform 10^8 operations
   If equally distributed to each test case, we remain with 10^7
   - $O(n^2)$ is out of question as that will take operation in order of 10^{12}
   - Let's check $O(nlog(n))$ takes order of 2×10^7 which is a little bit around the edge, you can go with it but chances of TLE are still there
   - $O(n)$ is order 10^6 so it the best choice so far.
2. Time Limit $= 1\ second$
   $N = 10^5$
   $TestCases = 10$
   So in $1\ second$ we are allowed to perform 10^8 operations
   If equally distributed to each test case, we remain with 10^7
   - $O(n^2)$ is out of question as that will take operation in order of 10^{10}
   - Let's check $O(nlog(n))$ takes order of 1.6×10^6 which is a pretty good choice
3. Time Limit $= 5\ second$
   $N = 10^5$
   $TestCases = 1$
   - $O(n^2)$ is out of question as that will take operation in order of 10^{10}

- Let's check $O(nlog(n))$ takes order of 1.6×10^6 which is a pretty good choice
- But some good algorithm like Mo's Algorithm will also work as it has complexity of $O(N\sqrt{N})$ which makes it less than 10^8 so, good so far.

Lets see a chart

N	**Acceptable Complexity**
≤ 10	$O(N!)$ or $O(2^N)$
$\leq 10^2$	$O(N^3)$ or $O(N^4)$
$\leq 10^3$	$O(N^2)$
$\leq 10^4$	$O(N^2)$ or $O(Nlog(N))$
$\leq 10^5$	$O(Nlog(N))$ or $O(N\sqrt{N})$
$\leq 10^6$	$O(Nlog(N))$
$\leq 10^8$	$O(N)$
$\geq 10^8$	$O(log(N))$