4 Sum

↳ Prerequisite => 2 Sum => 3 Sum

The same concept like 3 sum is used here also.

- **Brute Force** :— (Same as 3 sum)

```
for(int i=0 ⟶ n-1) {
    for(int j= i+1 ⟶ n-1) {
        for(int k= j+1 ⟶ n-1){
            for(int l= k+1 ⟶ n-1) {
```

}
}
}
}

T = ~~O(n^3)~~ O(1) Adding from set to List

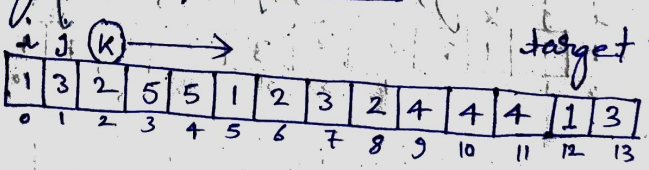
$T = O(n^4) * O(\text{hashSet\_search}) + O(n)$

$= O(n^4)$

$S = O(n)$

- **Better Approach (1st Best Sol.)** (Same as 3 sum)

↳ Using hashMap & hashSet



(Now, search for target - (arr[i] + arr[j] + arr[k]) in the hashMap. And then put (arr[k], k) into the hashMap.

Then, k++.

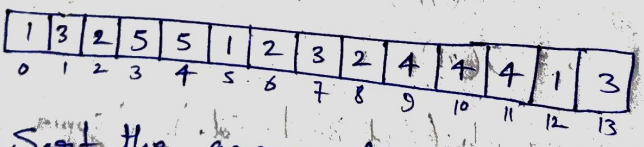
```
for(int i=0 ⟶ n-1) {
    for(int j=i+1 ⟶ n-1) {
        for(int k=j+1 ⟶ n-1) {
            search for (target-(arr[i]+arr[j]+arr[k]))
            in the hashMap
            ≡≡≡
        }
    }
}
```

$T = O(n^3) * O(\text{hashMap\_search} + \text{hashSet\_search} + \text{hashMap\_put}) + O(n)$

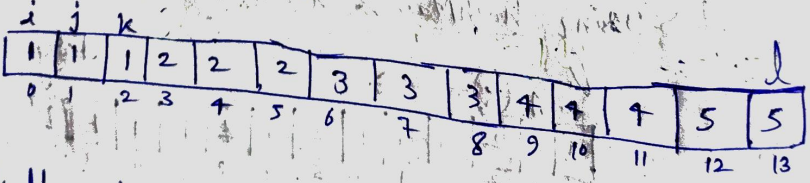
$\Rightarrow T = O(n^3)$

$S = O(n) + O(n) = O(n)$

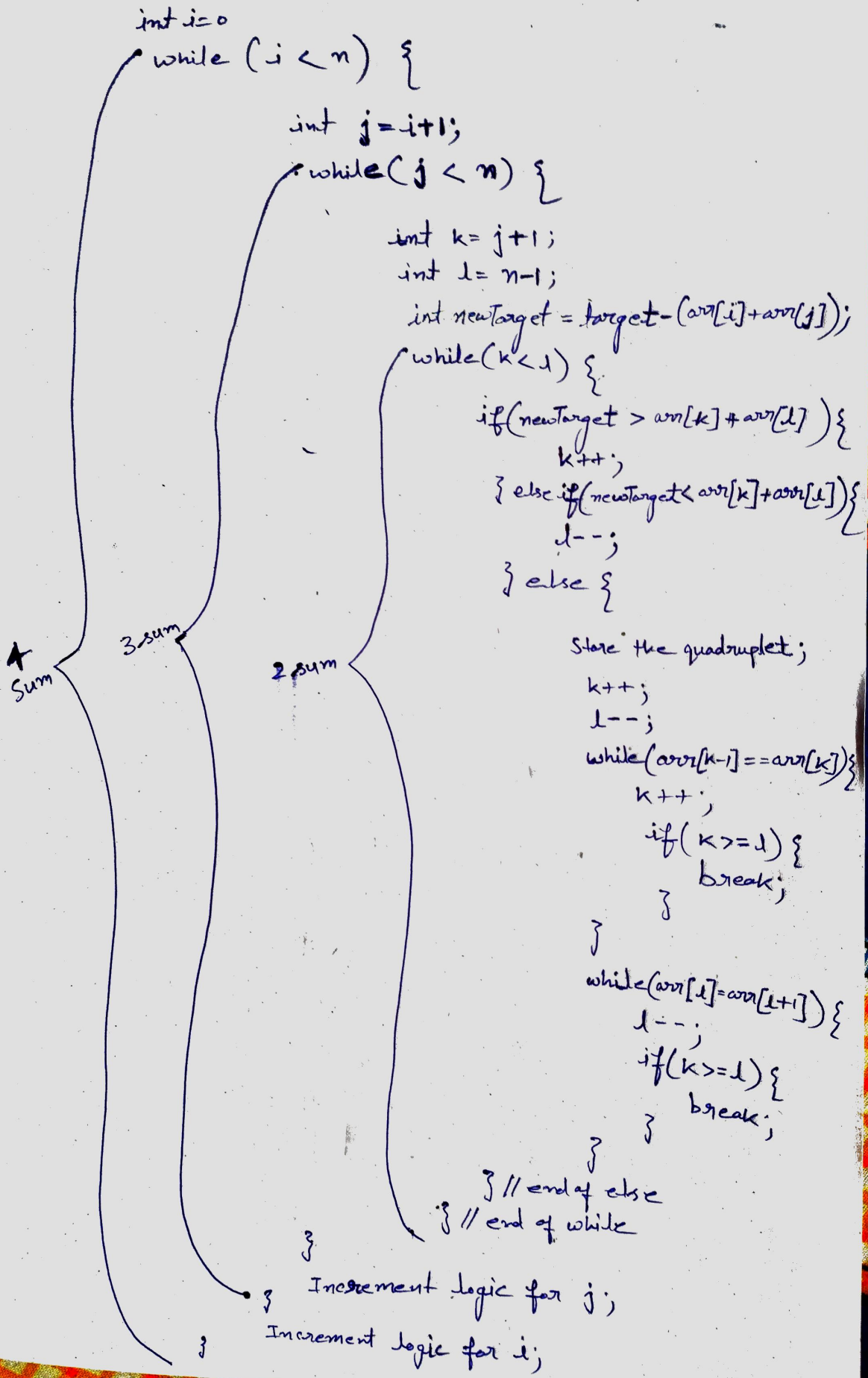
- 2nd Best Solution (Two pointer Solution) $\Rightarrow$ Same as 3 sum
  - $\Rightarrow$ Best and Optimal Approach



$\Rightarrow$ Sort the array first



Initially, $i=0$, $j=i+1$, $K=j+1$ and $l=n-1$


int i=0
while (i < n) {
int j=i+1;
while(j < n) {
int k= j+1;
int l= n-1;
int newTarget = target-(arr[i]+arr[j]);
while(k<l) {
if(newTarget > arr[k]+arr[l]){
k++;
} else if(newTarget< arr[k]+arr[l]){
l--;
} else {
Store the quadruplet;
k++;
l--;
while(arr[k-1]==arr[k]){
k++;
if(k>=l) {
break;
}
}
while(arr[l]=arr[l+1]) {
l--;
if(k>=l) {
break;
}
}
} // end of else
} // end of while
}
Increment logic for j;
}
Increment logic for i;
}
4 Sum
3 sum
2 sum

$T = O(n \log n) + O(n^3) = O(n^3)$

$S = O(1)$