Two pointers

What is the pattern?

The idea here is to iterate two different parts of the array simultaneously to get the answer faster.

These kind of problems usually involve two pointers

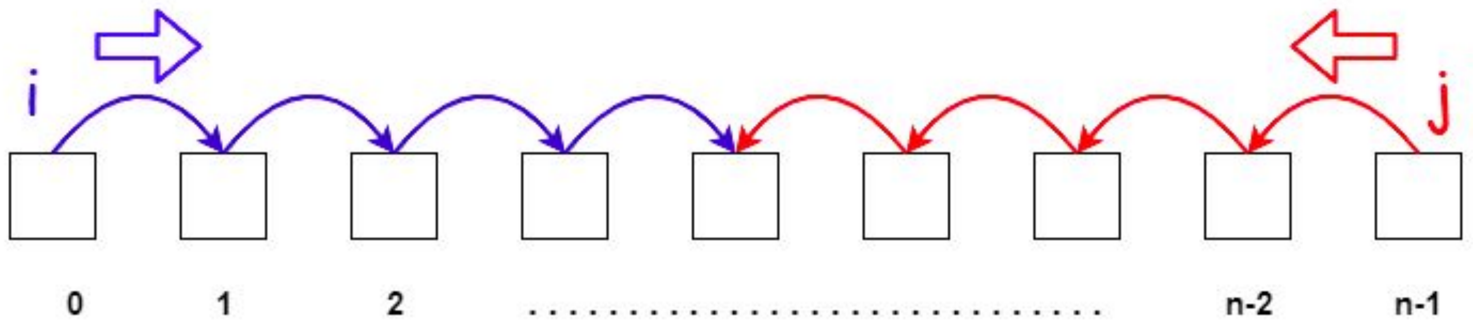
1. One pointer starts from the beginning while the other pointer starts from the end.

2. One slow-runner and the other fast-runner.

Implementation

1. One pointer at each end

One pointer starts from beginning and other from the end and they proceed towards each other



Example

In a sorted array, find if a pair exists with a given sum S

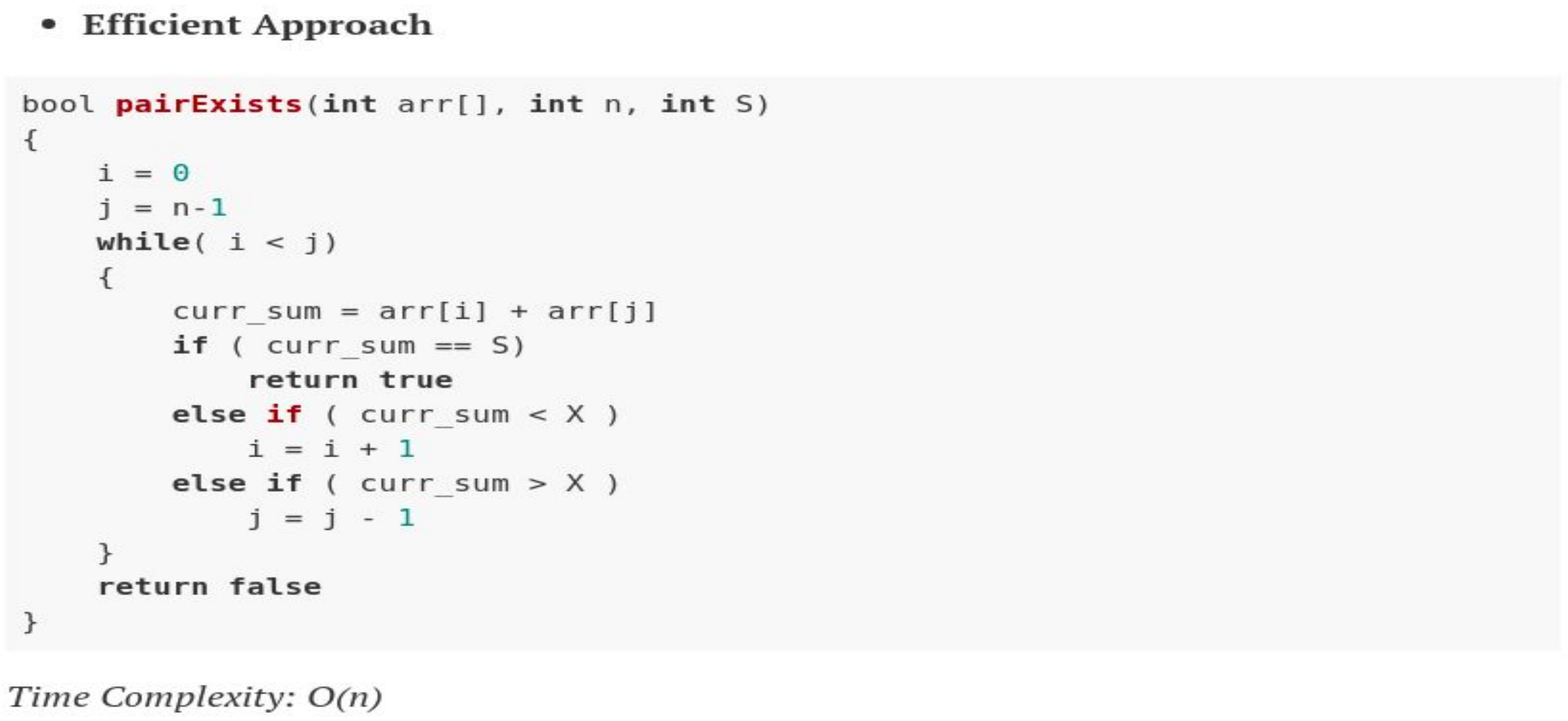
Brute Force Approach

- **Brute Force Approach:** We could implement a nested loop finding all possible pairs of elements and adding them.

```
bool pairExists(int arr[], int n, int S)
{
    for(i = 0 to n-2)
        for(j = i+1 to n-1)
            if(arr[i] + arr[j] == S)
                return true
    return false
}
```

Time complexity: $O(n^2)$

Efficient Approach



- **Efficient Approach**

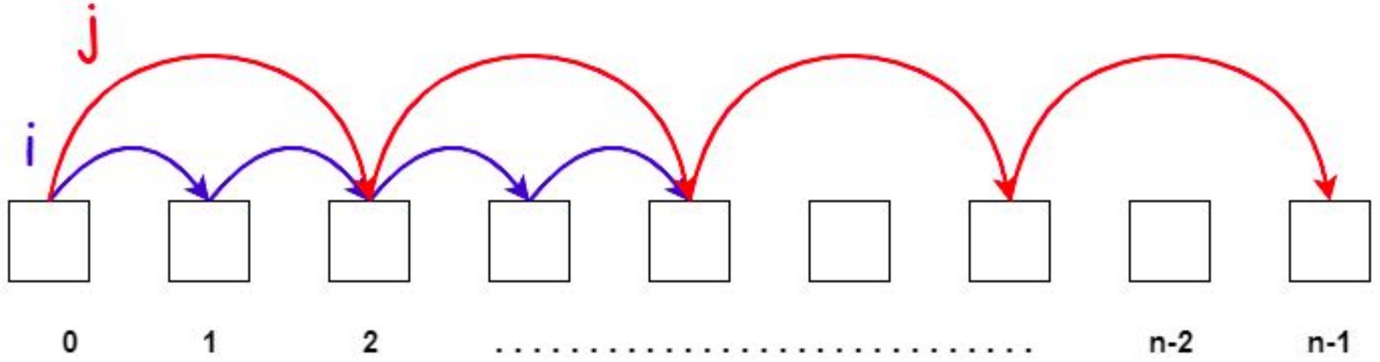
```
bool pairExists(int arr[], int n, int S)
{
    i = 0
    j = n-1
    while( i < j)
    {
        curr_sum = arr[i] + arr[j]
        if ( curr_sum == S)
            return true
        else if ( curr_sum < X )
            i = i + 1
        else if ( curr_sum > X )
            j = j - 1
    }
    return false
}
```

Time Complexity: O(n)

Implementation

2. Different Paces

Both pointers start from the beginning but one pointer moves at a faster pace than the other one.



Example

Find the middle of a linked list

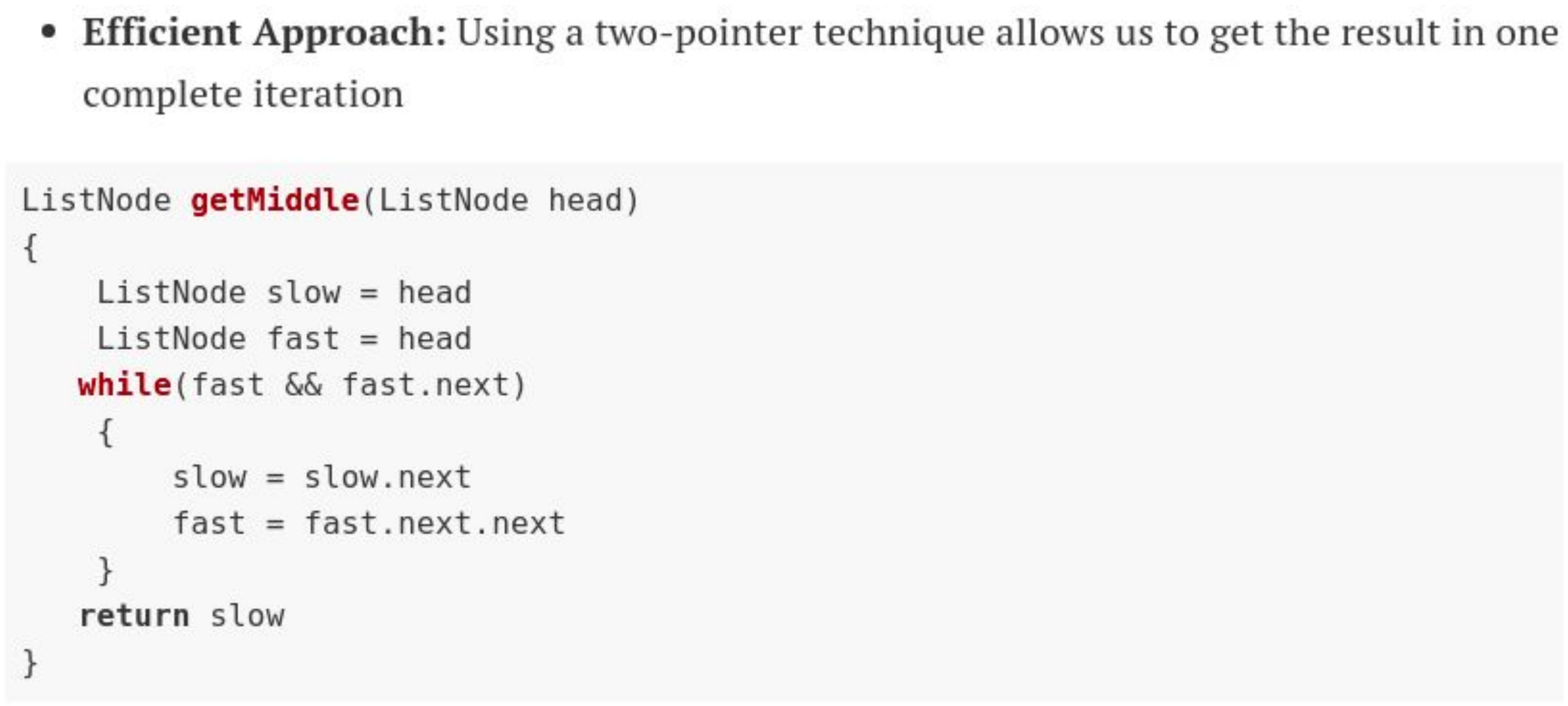
Brute Force Approach

- **Brute Force Approach:** We can find the length of the entire linked list in one complete iteration and then iterate till half-length again.

```
ListNode getMiddle(ListNode head)
{
    len = 0
    ListNode curr = head
    while ( curr != NULL )
    {
        curr = curr.next
        len = len + 1
    }

    curr = head
    i = 0
    while(i != len / 2)
    {
        curr = curr.next
        i = i + 1
    }
    return curr
}
```

Efficient Approach



- **Efficient Approach:** Using a two-pointer technique allows us to get the result in one complete iteration

```
ListNode getMiddle(ListNode head)
{
    ListNode slow = head
    ListNode fast = head
   while(fast && fast.next)
    {
        slow = slow.next
        fast = fast.next.next
    }
   return slow
}
```

How to identify the two pointer algorithms?

Does the problem request a search of two or more items? If this is not directly what the question is asking for, can it be reduced to such a task?

Does there exist an initialization of two pointers such that the rest of the algorithm can function properly for all cases?

Does there exist some sort of function to evaluate the goodness of the current locations of the pointers?

Based on the goodness, does there exist a universal rationale to move the pointers to maximize the goodness?

Could the moving of the pointers eliminate valid solutions?

Should the array be sorted before pointers are initialized?

Should pointers be moving at different speeds?

Should pointers be updated one at a time or simultaneously?

How can additional aspects of efficient computing, like creating lists to store previous pointer elements, assist pointers in further ruling out redundant solutions or to solve the problem more efficiently?

How should relationships between three or more pointers be managed (if applicable to the problem)?

If managing three or more pointers is not possible or very complex, can it be reduced to multiple two-pointer problems?

Time and space complexity

Time Complexity: $O(n^2)$ -> $O(n)$
Space Complexity: $O(n)$ -> $O(1)$

Resources

Articles:
https://medium.com/swlh/two-pointer-technique-solving-array-problems-at-light-speed-56a77ee83d16
https://afteracademy.com/blog/what-is-the-two-pointer-technique
https://leetcode.com/articles/two-pointer-technique/
https://algodaily.com/lessons/using-the-two-pointer-technique

Videos:
https://www.youtube.com/watch?v=sGdwSH8RK-o&ab_channel=takeUforward