CSE 595: Advanced Topics in Computer Science
Presentation 1

Zeeshan Shaikh

Department of Computer Science, Stonybrook University

05/30/2021

Topics for today's presentation

- Count quadruplets with sum K from given array.
- Given an array of size n and a number k, find all elements that appear more than n/k times.

Problem 1: Count quadruplets with sum K from given array.

Understanding the problem:

- Given: an array and a target sum.
- Task: To find all possible quadruplets in the array, sum of which is equal to the given target sum.
- Example:
  exampleArray = [1, 2, 3, 4, 5]
  targetSum = 10
  Answer = 1 [subset:[1,2,3,4] satisfies!]

Possible Approaches:

- Naive Approach: Calculate the sum of all 4-element-combinations and check if it's equal to the required sum.
- Using a HashMap:
  - One Pass HashMap
  - Two Pass HashMap
- Double Pointer Method

Problem 1: Naive Approach

Algorithm 1 Naive Approach Algorithm

```
1: for i = 1 to N − 3 do
2:     for j = i + 1 to N − 2 do
3:         for k = j + 1 to N − 1 do
4:             for l = k + 1 to N do
5:                 sum = array[i] + array[j] + array[k] + array[l]
6:                 if sum = targetSum then
7:                     Increase counter
8:                 end if
9:             end for
10:        end for
11:    end for
12: end for
```

Problem 1: Naive Approach Visualization

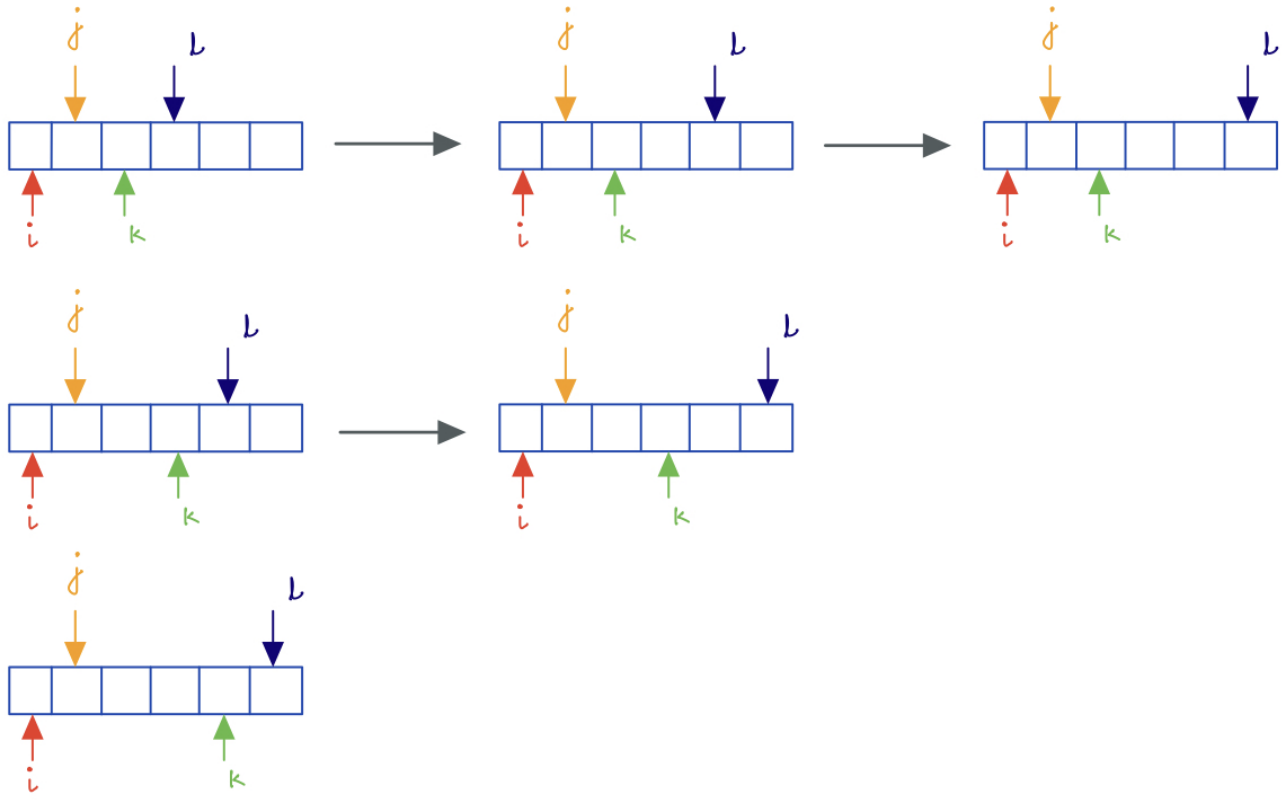

Figure 1: Movement of the iterators in the Naive Approach

Time Complexity: $O(N^4)$
Space Complexity: $O(N)$

Problem 1: HashMap Approach - Double Pass

Algorithm 2 Double Pass HashMap Algorithm

```
1: for i = 0 to N − 3 do
2:     for j = i + 1 to N − 2 do
3:         remainingSum = targetSum − (array[i] + array[j])
4:         for k = j + 1 to N do
5:             Create entries in the hashmap for values corresponding
               to array[k]
6:         end for
7:         for k = j + 1 to N − 1 do
8:             finalElement = remainingSum − array[k]
9:             if finalElement in hashmap then
10:                Increase counter
11:            end if
12:        end for
13:    end for
14: end for
```

Problem 1: Double Pass Hashmap Visualization

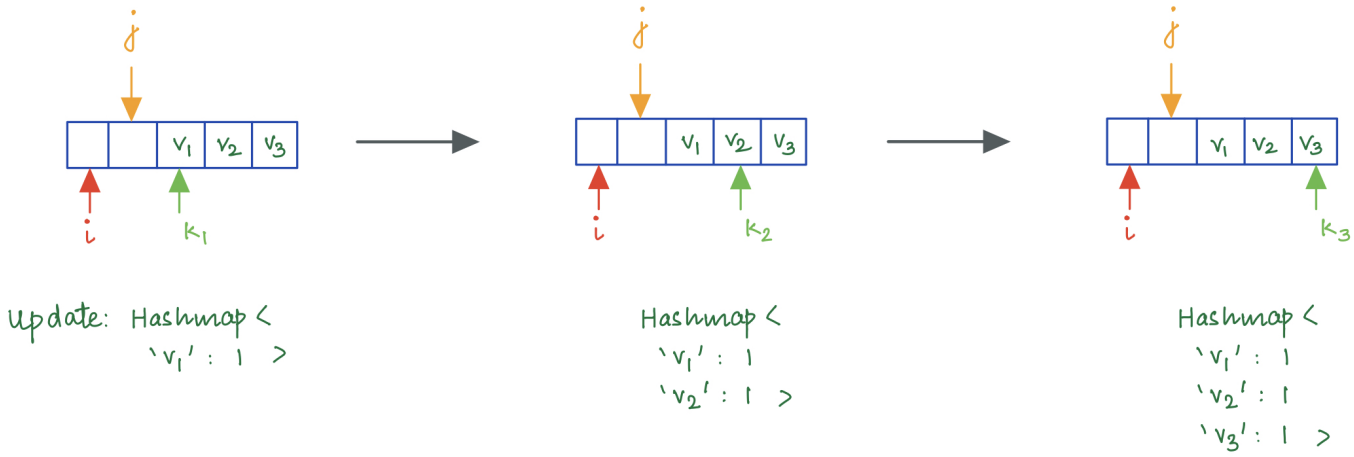

Figure 2: Movement of the iterators and update in the two-pass HashMap Approach

Problem 1: HashMap Approach - Single Pass

Algorithm 3 Single Pass HashMap Algorithm

```
1: for i = 0 to N − 1 do
2:     for j = i + 1 to N do
3:         tempSum = array[i] + array[j]
4:         remainingSum = targetSum − tempSum
5:         if tempSum < targetSum then
6:             counter += hashmap[remainingSum]
7:         end if
8:     end for
9:     for j = 0 to i do
10:        tempSum2 = array[i] + array[j]
11:        if tempSum2 < targetSum then
12:            Increase the counter of tempSum2 in hashmap
13:        end if
14:    end for
15: end for
```

Problem 1: Single Pass Hashmap Visualization

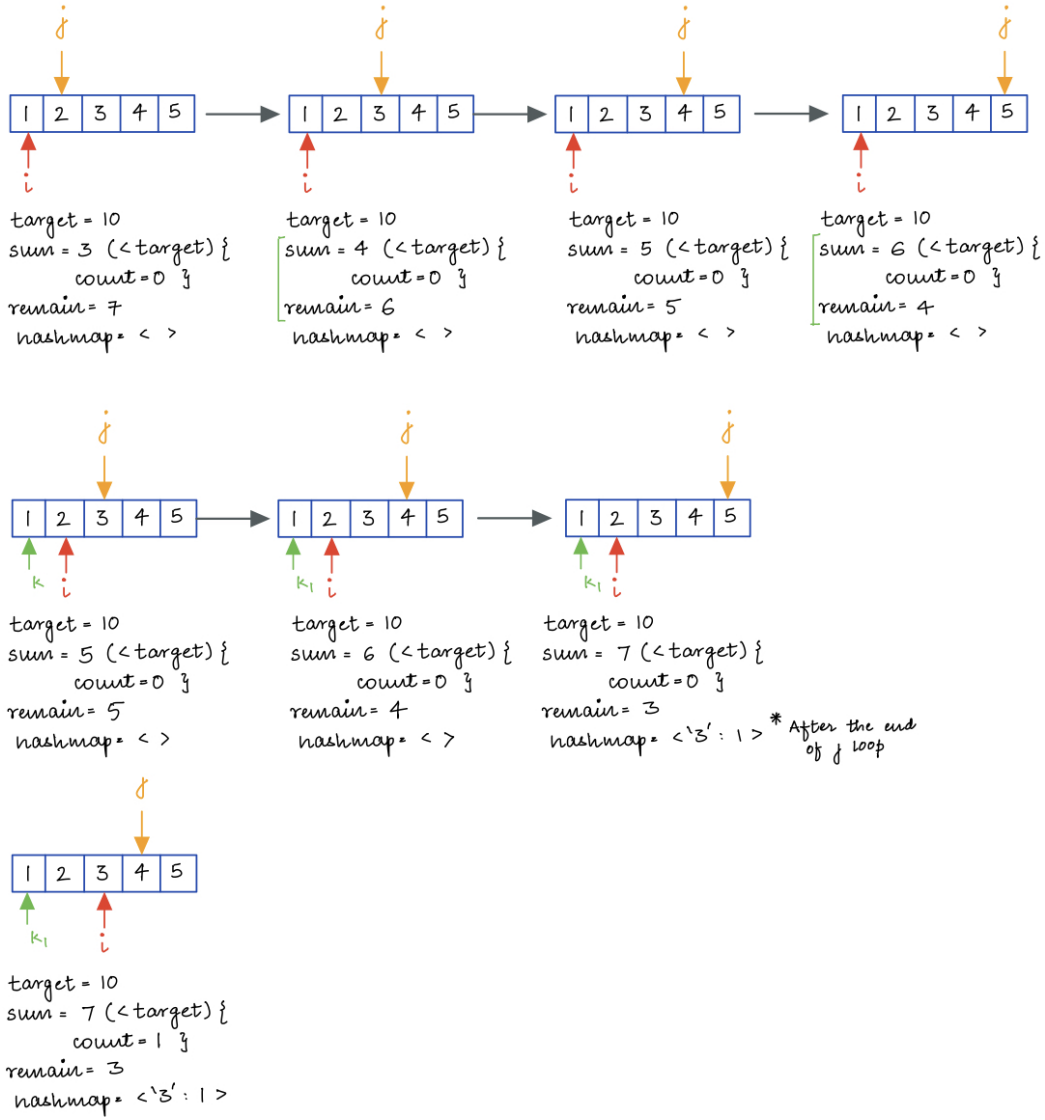

Figure 3: Movement of the iterators and update in the single-pass HashMap Approach

Complexity Analysis for Hashmaps

- Two Pass: Time Complexity: $O(N)$
  Space Complexity: $O(N)$
- One Pass: Time Complexity: (Worst case: linear, Average case: constant) $O(N)$
  Space Complexity: $O(N)$

Introducing the twoSum problem and solution using pointer approach

Algorithm 4 Basic two pointer algorithm

```
1: Initiate two pointers such that:
2: low = 0, high = lengthArray − 1
3: while high > low do
4:     sum = array[low] + array[high]
5:     if sum = targetSum then
6:         increment counter
7:         increment low
8:         high = lengthArray − 1
9:     else if sum > target then
10:        decrement high
11:    else
12:        increment low
13:    end if
14: end while
```

Pointer Solution Vizualisation

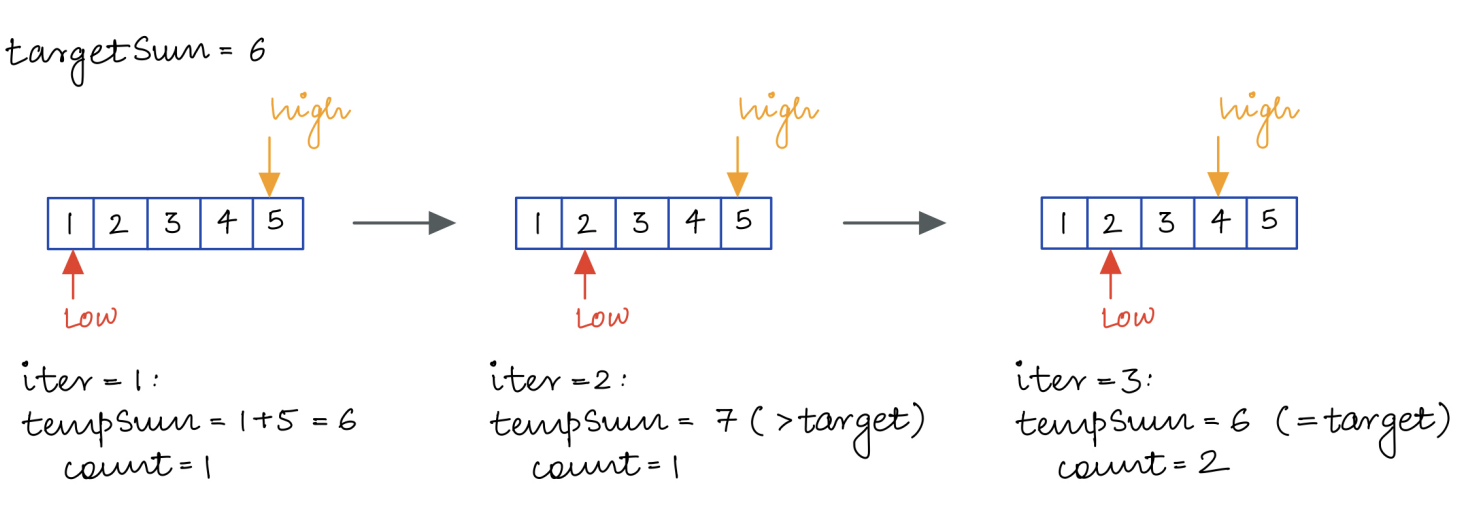

Figure 4: Pointer movement over the course of the algorithm

Time Complexity: $O(N^4)$
Space Complexity: $O(N)$

Can we generalize the two pointer algorithm?

- The two point algorithm is a very efficient algorithm to solve the two-sum problem.
- The question now is, can we generalize it for 3-sum, 4-sum ... k-sum algorithms?
- We can do so by converting any k-sum to a 2sum algorithm with k-2 loops.
- For example: If we take the 3-sum problem, we can fix the first element and then treat the remaining array as a 2-sum problem and apply the algorithm discussed in the previous slide.
- How would it look for 4-sum problem then?

Problem 1: Two pointer method

Algorithm 5 Two pointer algorithm for 4-sum

```
1:  for i = 0 to arrayLength − 3 do
2:      for j = i + 1 to arrayLength − 2 do
3:          remain = targetSum − (array[i] + array[j])
4:          low = i + 1, high = lengthArray − 1
5:          while high > low do
6:              sum = array[low] + array[high]
7:              if sum = remain then
8:                  increment counter, low
9:                  high = lengthArray − 1
10:             else if sum > remain then
11:                 decrement high
12:             else
13:                 increment low
14:             end if
15:         end while
16:     end for
17: end for
```

Problem 2: Given an array of size n and a number k, find all elements that appear more than n/k times

Understanding the problem:

- Given: an array and a random value "k".
- Task: To find all the elements such the frequency of the element is greater than the arrayLength/k.
- Example:
  exampleArray = [1, 2, 3, 4, 1, 1, 2, 2, 5]
  k = 4
  Answer = [1, 2]

Possible Approaches:

- Naive Approach: Calculate the frequency of all the elements by traversing the array
- Using a HashMap:
- Alternate Method

Problem 2: Hash method

Algorithm 6 Hash algorithm for n/k

```
1: Calculate the
2: for i from 0 to n − 1 do
3:     Add to the count of the element in the hashmap
4: end for
5: for Iterate over hashmap do
6:     if elementCount > ratio then
7:         save the corresponding element
8:     end if
9: end for
```

Time Complexity: $O(N)$
Space Complexity: $O(N)$

Problem 2: Alternate method

Algorithm 7 Alternate algorithm for n/k

```
1: define a structure of length k-1 to hold the element and count
2: if element of array is already present in the structure then
3:     Increase its count
4: else if element of array is not present in the structure then
5:     if there is space in structure then
6:         Add element and set count to 1
7:     else
8:         Reduce the count of all elements by 1
9:     end if
10: end if
```

Problem 2: Alternate algorithm Visualization

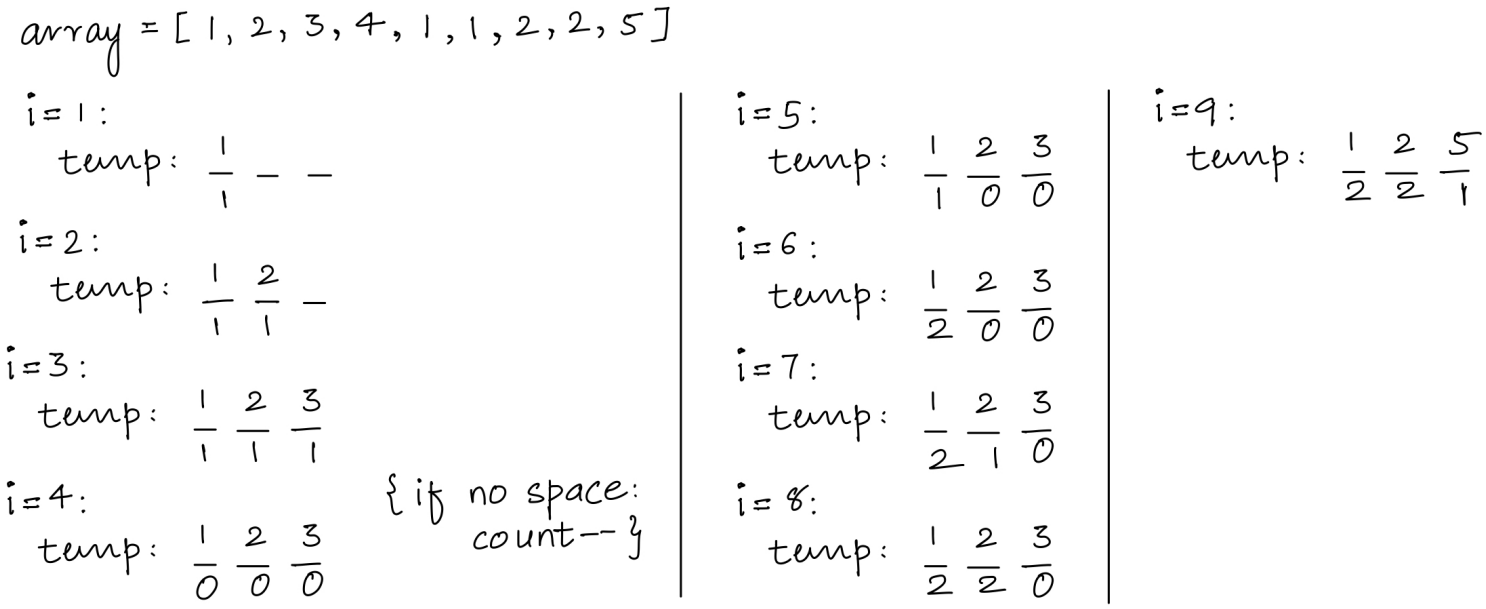

Figure 5: Structure element changes over the course of the algorithm

Time Complexity: $O(N * K)$
Space Complexity: $O(K)$

Thankyou!