Table of Contents

[Array 1](#_Toc204842274)

[1-Two Sum (easy) 1](#_Toc204842275)

[3-Longest Substring without repeating characters 2](#_Toc204842276)

[9-Palindrome Number (easy) 3](#_Toc204842277)

[15-3sum(medium) 5](#_Toc204842278)

[31- Next Permutation 6](#_Toc204842279)

[560-Subarray Sum Equals K (medium) 6](#_Toc204842280)

[56-Merge Intervals 7](#_Toc204842281)

[Strings 9](#_Toc204842282)

[8-String to Integer(ATOI) 9](#_Toc204842283)

[Maze and map and Graph 10](#_Toc204842284)

[200-Number of Island (medium) 10](#_Toc204842285)

[133-Clone Graph (medium) 12](#_Toc204842286)

# Array

## 1-Two Sum (easy)

Text

AI-generated content may be incorrect.

1. The indices of the number list need to be tracked (using hashtable)
2. Then iterate through the number list again, find the complement number that will add up to the target.
3. Use the complement number to search the hashtable for it’s index stored at step1)
4. Needs to make sure it’s not the same number

Text

AI-generated content may be incorrect.

Time: O(n) : iterate through the list at most twice

Space: O(n): hashtable uses space to store the entire list.

## 3-Longest Substring without repeating characters

Text

AI-generated content may be incorrect.

1. Brutal force would result in O(n^2) so it’s not a good idea
2. Use sliding window technique. Start with the right pointer moving further right, if it hits a letter that is already in the window, it means there is a duplicate, and the duplicate will always be at the left pointer, then the letter at left point can be kicked out of the window.
3. A deque can be used to the store the characters in the window, and .append and .popleft fits perfectly for extent right pointer.
4. The length of the window (length of the deque) is checked at every window adjustment.
5. Text

   AI-generated content may be incorrect.
6. Time: O(n), one traversal of the string
7. Space: O(n), queue set up that will store the entire string at worst case

## 9-Palindrome Number (easy)

Text

AI-generated content may be incorrect.

1. Pointer Method:
   1. The number needs to be converted into string
   2. Have two pointers, 1 set in the beginning of the string one at the end
   3. Both pointers would move from the ends and meet in the middle, and the character will need to match each of the steps.
   4. If the count of the character is odd, there will be a middle number and it doesn’t matter what it is.
   5. A screenshot of a computer

      AI-generated content may be incorrect.
   6. Time: O(n): the number string is iterated through 1 time
   7. Space: O(n): a new string is created for the numbers
2. Mathmatical method (medium):
   1. No need for string conversion so the space is O(1)
   2. The negative numbers, numbers ends with 0 or = 0 can’t be palindrome
   3. Numbers modulus by 10 will get you the right most digit
   4. Numbers divided by 10^i (where i is the number of digits of the number) will get you the left most digit
      1. 10^i will be set as a divider value
      2. A screenshot of a video game

         AI-generated content may be incorrect.
      3. This way div will be a 10^i number with same number of digit as x
   5. Numbers modules by 10^i (where i is the number of digits of the number) will get rid of the left most digit (e.g. 1221%1000 = 221)
   6. Using the number from e), divide it by will remove the left most digit(e.g. 221//10 = 22)
   7. The divider value will always be decrease by 100 in the next iteration since two digits are removed
   8. Using the result from f) and repeat c) to g) until a non-matching pair is found
   9. Text

      AI-generated content may be incorrect.
   10. Time: O(logn): digit count is tied to the log of the number
   11. Space: O(1): no data structure usage, so no additional utilization

## 15-3sum(medium)

Text

AI-generated content may be incorrect.

1. Since there will be duplicate numbers, but we don’t want duplicate combinations as outputs, easiest way is to sort the input array.
2. have a pointer to iterate through the array, two other pointers to take care of the segment following the first pointer.
3. The first pointer of and the head segment pointer will take care of the duplicate issue (skipping if duplicated, sorted array is convenient in this case)
4. If First point + head and tail pointers of the following segment = 0, we find the answer, else adjust the head or tail pointers if <0 or >0
5. Text

   AI-generated content may be incorrect.
6. Time: O(nlogn):python sort; O(n^2): 2 nested loops, so nlogn + n^2
7. Space:O(1): no additional data structure used, maybe O(n) if sorting used some space

## 31- Next Permutation

Text

AI-generated content may be incorrect.

## 560-Subarray Sum Equals K (medium)

Text

AI-generated content may be incorrect.

1. Brute force will have O(n^2) time because it will be a 2 nested loop, sliding window may sound like a good approach for O(n) but negative numbers will conflict the window expansion, shrink, and move, because increase and decrease of the window does not guaranteed increase and decrease of the sum.
2. To get O(n) there should only be 1 pass and info gain along the pass should be utilized
3. Understand presum and the idea of subSum + presum = k, so subSum = presum – k, see this as subSum is what’s need by presum to get to k
4. Since it’s one pass subSum should have been calculated it’s frequency counted and stored in hashtable.
5. If subSum not found in hashtable, it mean no such subarray existed yet, but still place it in hashtable, which maybe needed for next subarrays.
6. It’s like storing missing pieces during the 1 pass, and be sure to store the info for possible later usage

Text

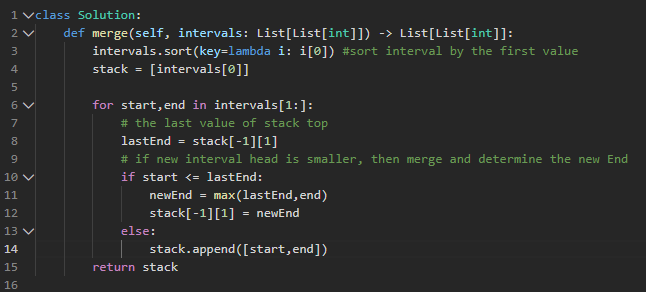
AI-generated content may be incorrect.

1. Time: O(n): 1 pass through the number list, and about 1 pass though the preSumList
2. Space: O(n): hashtable is created to store all the presums

## 56-Merge Intervals

Text

AI-generated content may be incorrect.

1. need to make sure the intervals are sorted by the first value.
2. Think about it inserting the intervals into a stack, if the first value of the incoming interval is smaller than the last value on the stack top, the interval is mergeable
3. After confirming it’s mergeable, the last value of the new interval needs to be determined and it will either be the last value of the incoming interval or the stack top. Whichever is larger
4. The stack needs to be initialized with a beginning interval
5. 
6. Time: O(n)-only 1 iteration through the list
7. Space:O(n)-a stack is used that will store the whole list at worst

# Strings

## 8-String to Integer(ATOI)

Text

AI-generated content may be incorrect.

Time:O(n)-one time traversal

Space: O(1), no additional data structure used

# Maze and map and Graph

## 200-Number of Island (medium)

Text

AI-generated content may be incorrect.

1. It’s a BFS problem, the pointer starts with the left top corner. if the pointer hits a land(“1”) then all it’s connecting lands will be saved in visited set, and the count adds by 1. Then the pointer starts to look for other lands that are isolated from the visited lands and start the whole process
2. Text

   AI-generated content may be incorrect.
3. Time: O(V+E), V is number of vertices and E is number of edges
4. Space: O(V), the number of vertices

## 133-Clone Graph (medium)

Graphical user interface, text

AI-generated content may be incorrect.

1. The edges are undirected meaning the pointers can go back and forth between the nodes
2. We basically want to clone each nodes including the neighbors that it points to
3. However, since edges are undirected all the nodes can be point to each other and and it could cause infinite number of cloning, or cycling
4. The key is to avoid cycling, so it’s important to keep track of the visited node (or the nodes that are already cloned), so you don’t go back to it again. We can use hashmap to store the visited notes and its information
5. If using DFS (recursive):
   1. Start at node 1, create a copy of 1, use hash map to map the old node to the new node
   2. Node 1 have 2 and 4 as neighbor, go to 2, create copies and store in hashmap, before going back to 4, due to recursion, the next node to visit is 2
   3. Node 2 have node 3 and 1 as neighbor, same thing (copy and store in hashmap), got to 3. Then same thing on 4
   4. Now, in 4, which has 1 and 3 as neighbors, they are all in the hashmap(visited), no more copies will be made but just linkage (copy.neighbors.append()). The pointer will go loop back to establish all the linkages, think about unwrapping.
   5. Text

      AI-generated content may be incorrect.
   6. Time: O(E+V) – all vertices and edges are visted
   7. Time:O(V) – hashmap to store all the vertices, O(H):a memory stack is created for every single vertices, H is the height of the graph, overall it will be O(V).
6. Using BFS:
   1. Visit the first node, and the all the neighbor connected to it before the next node. While visiting, make copy of the nodes and store them to hashmap to mark them as visited, also append the original nodes (not clones) to the que for later visit.
   2. Make clones of the neighbors and attached to the clone of the original node
   3. Then finish up the queue by repeating above
   4. Text

      AI-generated content may be incorrect.
   5. Time: O(E+V)
   6. Space: O(V) because of the hashmap usage