**Remove Duplicates from Sorted Array**

**first solution is by creating another array**

1. arr = [1,2,2,5,6,7,7,8,9]
2. rem = []
3. # copy to the second array only when elements are not duplicate
4. **for** i **in** range(len(arr)):
5. **if** i == 0:
6. rem.append(arr[i])
7. **elif** i > 0 **and** arr[i] != arr[i-1]:
8. rem.append(arr[i])
9. count = 0
10. **print** rem
11. **print** arr

**# another approach  - not using another array - O(n\*n)**

1. arr = [1,2,2,5,6,7,7,8,9]
2. temp = 0
3. dupCount = 0
5. **for** i **in** range(len(arr)):
6. **if** i > 0 **and** arr[i] == arr[i-1]:
7. # we found a duplicate
8. dupCount += 1
9. # loop through rest of the array to change the index by -1
10. **for** j **in** range(i, len(arr)-1):
11. arr[j] = arr[j+1]
13. # trim the array original length minus duplicates
14. arr = arr[0:len(arr)-dupCount+1]
15. **print** arr

**# another approach  - not using another array - O(n)**

1. **Have 2 pointers – “i at 0, j at 1**
2. arr = [1,2,3,5,6,7,7,8,9,9]
3. i = 0
4. nondup = 0
5. **for** j **in** range(len(arr)):
6. **if** arr[i] != arr[j]:
7. # till the time arr[i] = arr[j] then dont increment i
8. # as soon as they are not equal that means there is a new element
9. # so copy the new element
10. i = i + 1
11. nondup = nondup + 1
12. arr[i] = arr[j]
14. arr = arr[0:nondup + 1]
15. **print** arr

**Remove Duplicates from unsorted Array**

1. **def** removeDupes(arr):
2. res = []
3. uniqueValues = {}
4. j = 0
5. **for** i **in** range(len(arr)):
6. **if** arr[i] **not** **in** uniqueValues:
7. # we are putting sequence 0,1,2,3.. in values
8. uniqueValues[arr[i]] = j
9. j = j + 1
11. **for** i **in** range(len(uniqueValues)):
12. # get key using value
13. res.append(int(uniqueValues.keys()[uniqueValues.values().index(i)]))

16. **return** res
18. myArr = [1,2,55,-9,5,10,2,10,9,8]
19. **print** removeDupes(myArr)

**Palindrome – approach 1 – 2 pointers**

**2 pointer – one at beginning and one at end, keep them moving towards each other**

1. **def** checkPalindrome(str):
2. i = 0
3. j = len(str) - 1
5. **while** i < j:
6. **if** str[i].lower() != str[j].lower():
7. **return** False
8. i = i + 1
9. j = j - 1
11. **return** True

**Palindrome – approach 2**

**Compare first part and second part(in reverse)**

**def** checkPalindrome(str):

1. **if** len(str)%2 == 0:
2. **if** str[:int(len(str)-len(str)/2)-1:-1] == str[0:int(len(str)-len(str)/2):]:
3. **return** "True"
4. **else**:
5. **if** str[:int(len(str))-int(len(str)/2)-1:-1] == str[0:int(len(str))-int(len(str)/2)-1]:
6. **return** "True"
7. **return** "False"

**Possible Palindrome – permutation**

Given a string, determine if a permutation of the string could form a palindrome. For example, "code" -> False, "aab" -> True, "carerac" -> True. '''

**class** Solution(object):

**def** canPermutePalindrome(self, s):

1. """ # a string is Palindrome of the each charater appears for even number of times and only character can appear for odd number of times so we count the characters using Hash map and then count
2. myLetters = {}
3. **for** i **in** range(len(s)):
4. **if** s[i] **in** myLetters:
5. #print s[i]
6. myLetters[s[i]] += 1
7. **else**:
8. myLetters[s[i]] = 1
9. #print myLetters
10. countOdd = 0
11. **for** k **in** myLetters:
12. #print myLetters[k]
13. **if** myLetters[k]%2 <> 0:
14. countOdd = countOdd + 1
15. #print "countOdd", countOdd
16. **if** countOdd > 1:
17. **return** False

**return** True

**Reverse Array – Swap - Iterative**

1. **def** reverseArray(arr):
3. **if** len(arr) == 0 **or** len(arr)  == 1:
4. **return** arr
5. i = 0
6. j = len(arr) - 1
8. **while** i < j:
9. (arr[i], arr[j]) = (arr[j], arr[i])
10. i = i + 1
11. j = j - 1
13. **return** arr
15. myArray = [1,9,7,10,55,16,67,-1]
16. **print** reverseArray(myArray)

**Reverse Array – Swap - Recursive**

1. **def** reverseArray(arr,arrStart, arrEnd):
2. **if** len(arr) == 0 **or** len(arr)  == 1:
3. **return** arr
4. # base condition
5. **if** arrStart == arrEnd **or** arrStart > arrEnd:
6. **return** arr
7. # swap the elements
8. (arr[arrStart], arr[arrEnd]) = (arr[arrEnd], arr[arrStart])
9. # call it recursively
10. **return** reverseArray(arr,arrStart + 1, arrEnd - 1)

**Reverse Array – Other approach**

1. # using the minus index  using another array
2. a = [1,2,3,4,5,6]
3. b = []
4. **for** i **in** a:
5. b.append(a[-i])
7. **print** b
9. # string array using the loop  using another array – copy the elements from last and so on
10. x = ['Ankit','Jasmeet','Sandip','Ganesan','Balaji']
11. y = []
13. **for** i **in** range(len(x)):
14. y.append(x[len(x) - 1- i])
16. **print** y
18. # string array using string inbuilt function
19. x = ['Ankit','Jasmeet','Sandip','Ganesan','Balaji']
20. y = x[::-1]

**Is a number Prime ?**

1. **def** isPrime(n):
2. # loop until square root of n
3. **for** i **in** range(2,int(n\*\*0.5)+1):
4. **if** n%i == 0:
5. **return** False
6. **return** True

**Target sum in Array – Hash table two pass**

1. **def** twoSum(arr, target):
2. # create a Hash table of the elements
3. hashArr = {}
4. **for** i **in** range(len(arr)):
5. hashArr[arr[i]]  = i
7. # now look for the complement in the Array vs Hashtable
8. **for** i **in** range(len(arr)):
9. **if** target - arr[i] **in** hashArr:
10. **return** i, hashArr[target - arr[i]]
11. **return** -1,-1

**Target sum in Array – Hash table one pass**

1. **def** twoSum(arr, target):
2. # create a Hash table of the elements
3. hashArr = {}
4. **for** i **in** range(len(arr)):
5. # check at the same time
6. **if** target - arr[i] **in** hashArr:
7. **return** i, hashArr[target - arr[i]]
8. hashArr[arr[i]]  = i
10. **return** -1,-1

**Fibonacci - Recursive**

1. **def** nFibonacci (n):
2. **if** n == 1:
3. **return** 1
4. **elif** n == 0:
5. **return** 0
6. **else**:
7. **return** nFibonacci(n-1) + nFibonacci(n-2)
9. **print** nFibonacci(8)

**Fibonacci - Iterative**

1. **def** nFibonacciLoop(n):
2. res = 0
3. **if** n == 1:
4. **return** 1
5. **elif** n == 0:
6. **return** 0
8. last = 1
9. secondLast = 1
10. **for** i **in** range(2,n):
11. res = last + secondLast
12. last = secondLast
13. secondLast = res
14. **return** res

17. **print** nFibonacciLoop(9)

**Fibonacci – Recursive with Dynamic Programming**

'''''

This function calculates Fibonocci number at nth place using Dynamic Programming

'''

1. fibHash = {}
2. **def** fibDP(n):
3. **if** n == 1:
4. **return** 1
5. **if** n == 0:
6. **return** 0
7. **if** n **in** fibHash:
8. **return** fibHash[n]
10. # store the computed value in Hash table
11. # Memoization
12. fibHash[n-1] = fibDP(n-1)
13. fibHash[n-2] = fibDP(n-2)
15. **print** fibHash
16. **return** fibDP(n-1) + fibDP(n-2)

**Reverse a Number**

1. # this function creates an array out of an integer
2. # this can also be used to reverse a number
4. **def** createArrayFromInt(n):
5. arr = []
6. **while** n > 10:
7. arr.append(int(n%10))
8. n = n/10
9. arr.append(int(n))
10. **return** arr
12. **print** createArrayFromInt(123)

**Reverse a Number** **Recursive version -own**

1. **def** createArrayFromInt(n):
2. **if** n <= 10:
3. **return** str(int(n))
5. **return** str(int(n%10)) + createArrayFromInt(n/10)

**Maximum SubArray**

1. Kadane's Algorithm
2. '''
4. **def** maxSubArray(arr):
5. max\_so\_far = 0
6. max\_ending\_here = 0
8. **for** i **in** range(len(arr)):
9. max\_ending\_here = max\_ending\_here + arr[i]
11. **if** max\_ending\_here < 0:
12. max\_ending\_here = 0
13. **if** max\_so\_far < max\_ending\_here:
14. max\_so\_far = max\_ending\_here
16. **return** max\_so\_far

**Greatest Common Divisor – Iterative**

1. # Iterative solution
2. **def** findGCD(a,b):
4. **while** a != b:
5. **if** a > b:
6. a = a - b
7. **else**:
8. b = b - a
9. **return** a

**Greatest Common Divisor – Recursive**

1. **def** findGCDRecur(a,b):
2. **if** a == 0 **or** b == 0:
3. **return** 0
4. **if** a == b:
5. **return** a
6. **if** a > b:
7. **return** findGCDRecur(a-b,b)
8. **else**:
9. **return** findGCDRecur(a,b-a)

**Merge Sorted Arrays**

1. **class** Solution(object):
2. **def** merge(self, nums1, m, nums2, n):  ##m and n are lengths of arrays
3. """
4. :type nums1: List[int]
5. :type m: int
6. :type nums2: List[int]
7. :type n: int
8. :rtype: void Do not return anything, modify nums1 in-place instead.
9. """
10. **while** n > 0 **and** m > 0:
11. **if** nums1[m-1] <= nums2[n-1]:
12. nums1[m+n-1] = nums2[n-1]
13. n = n -1
14. **else**:
15. nums1[m+n-1] = nums1[m-1]
16. m = m - 1
17. **print** "n:",n
18. **print** "m:",m
19. **print** "nums1:",nums1
20. **print** "nums2:",nums2
22. **if** n > 0:
23. #nums1 = nums2[0:n] + nums1[m+1:]
24. nums1[:n] = nums2[:n]
25. **print** nums1

**Convert Decimal to Binary**

1. **def** convertToBinary(num):
2. binary\_table = []
3. b = 0
5. # this will be something like binary\_table = [1,2,4,8,16,32,64,128,256,512,1024...]
6. binary\_table = [2\*\*b **for** b **in** range(30)]
7. my\_keys = {}
8. n = num
9. # try to find the value in binary\_table
10. # if found add it in the hash table
11. **while** n > 0:
12. i = 0
13. **while** n >= binary\_table[i]:
14. i = i + 1
16. **if** binary\_table[i-1] **in** my\_keys:
17. my\_keys[binary\_table[i-1]] += 1
18. **else**:
19. my\_keys[binary\_table[i-1]] = 1
20. n = n - binary\_table[i-1]
22. # my\_keys will be something like {4: 1, 8: 1, 16: 1, 32: 1, 64: 1}
23. i = 0
24. arr = []
25. # then create the binary numbers using keys
26. **while** 2\*\*i <= num:
27. **if** 2\*\*i **in** my\_keys:
28. arr.append(1)
29. **else**:
30. arr.append(0)
31. i = i + 1
32. **return** arr[::-1]

**Convert Binary to Decimal**

1. **def** convertToDecimal(n):
2. arr = list(str(n))
3. **print** arr
4. i = len(arr) - 1
5. #print i
6. dec = 0
7. **while** i >= 0:
8. dec = dec + 2\*\*i\*int(arr[(len(arr)-i-1)])
9. #print i, 2\*\*i,dec
10. i = i - 1

**return** dec

**Bubble Sort**

Idea is to bubble up the largest (or smallest) item

# second largest, third largest and so on

# complexity O(n\*n)

1. **def** sortList(myList):
3. **for** i **in** range(len(myList)):
4. **for** j **in** range(len(myList) - 1 - i):
5. **if** myList[j] > myList[j + 1]:
6. # swap the tuple
7. (myList[j],myList[j + 1]) = (myList[j + 1], myList[j])
8. **return** myList

**Insertion Sort**

1. # Complexity O(nlogn) , worst O(n\*n)
2. myArray = [3,1,4,5,10,2,7,100]
3. res = [myArray[0]]

6. **for** i **in** range(1,len(myArray)):
7. res.append(myArray[i])
8. j = i
9. **while** j > 0:
10. **if** res[j] < res[j-1]:
11. # "swapping"
12. (res[j],res[j-1]) = (res[j-1], res[j])
13. j = j -1
14. **else**:

**break**

**Merge Sort**

1. # Complexity always O(nlogn)
2. **def** merge(a,b):
3. c = []
4. **while** len(a) > 0 **and** len(b) > 0:
5. **if** a[0] < b[0]:
6. c.append(a[0])
7. a.remove(a[0])
8. **else**:
9. c.append(b[0])
10. b.remove(b[0])
11. **if** len(a) == 0:
12. c = c + b
13. **else**:
14. c = c + b
15. **return** c
17. **def** mergeSort(myList):
19. **if** len(myList) == 1:
20. **return** myList
21. **else**:
22. middle = int(len(myList)/2)
23. a = mergeSort(myList[:middle])
24. b = mergeSort(myList[middle:])

**return** merge(a,b)

**Best time to buy Stocks**

1. **class** Solution(object):
2. **def** maxProfit(self, prices):
3. """
4. :type prices: List[int]
5. :rtype: int
6. """
7. buy\_price = 999999999
8. profit = 0
9. **for** i **in** range(len(prices)):
10. **if** prices[i] < buy\_price:
11. buy\_price = prices[i]
12. **elif** prices[i] - buy\_price > profit:
13. profit = prices[i] - buy\_price
14. **return** profit

**Average of Levels in Binary Tree**

1. **class** Solution(object):
2. **def** averageOfLevels(self, root):
3. """
4. :type root: TreeNode
5. :rtype: List[float]
6. """
7. queue = []
8. res   = []

11. **if** **not** root:
12. **return** 0
14. queue = [root]
16. **while** queue:
17. temp\_queue = []
18. levelSum = 0
19. levelCount = 0
21. **while** queue:
22. node = queue.pop()
23. levelSum = levelSum + node.val
25. **if** node.left:
26. temp\_queue.append(node.left)
27. **if** node.right:
28. temp\_queue.append(node.right)
29. levelCount = levelCount + 1
31. res.append(levelSum\*1.0/levelCount)
32. queue = list(temp\_queue)
34. **return** res

**Count Unique words**

1. myhash = {}
2. **for** w **in** (str.split()):
3. **if** w.lower() **in** myhash:
4. myhash[w.lower()] += 1
5. **else**:
6. myhash[w.lower()] = 1
7. **for** word, times **in** myhash.items():
8. **print**((word,times))

**Depth of Binary Tree**

1. **class** Solution(object):
3. **def** maxDepth(self, root):
4. """
5. :type root: TreeNode
6. :rtype: int
7. """
9. right = 0
10. left = 0
11. **if** **not** root:
12. **return** 0


16. **if** root.left != 'None':
17. left = self.maxDepth(root.left)
19. **if** root.right != 'None':
20. right = self.maxDepth(root.right)
22. **return** 1 + max(left,right)

**Find Substr in String**

1. **class** Solution(object):
2. **def** strStr(self, haystack, needle):
3. """
4. :type haystack: str
5. :type needle: str
6. :rtype: int
7. """
8. needleLen = len(needle)
9. i = 0
10. **while** i < len(haystack)-needleLen+1:
11. **if** haystack[i:i+needleLen] == needle:
12. **return** i
13. i = i + 1
14. **return** -1

**Invert Binary Tree**

1. **class** Solution(object):
2. **def** invertTree(self, root):
3. """
4. :type root: TreeNode
5. :rtype: TreeNode
6. """
7. **if** **not** root:
8. **return** root
10. left = self.invertTree(root.left)
11. right = self.invertTree(root.right)
12. (root.left,root.right) = (root.right, root.left)
13. **return** root

**Replace characters recursively**

1. **def** replaceX (str):
2. **print** 'input:',str
3. **if** len(str) == 0:
4. **return** str
5. **if** len(str) == 1 **and** str == 'x':
6. **return** 'y'
7. **elif** len(str) == 1 **and** str != 'x':
8. **return** str
9. mid = int(len(str)/2)
10. **return** replaceX(str[0:mid]) + replaceX(str[mid:len(str)])

**Perfect Number – Brute Force**

1. **def** checkPerfectNumber(self, num):
2. """
3. :type num: int
4. :rtype: bool
5. """
6. sum = 0
7. i = 1
8. **if** num <= 0:
9. **return** False
10. #print num\*\*0.5
11. **while** i < num:
12. #print i, sum
13. **if** num%i == 0:
14. sum = sum + i
15. **if** sum > num:
16. **return** False
17. i = i + 1
18. **if** sum == num:
19. **return** True
20. **else**:
21. **return** False

**Perfect Number – Optimized**

**Algorithm**

In this method, instead of iterating over all the integers to find the factors of numnumnum, we only iterate upto the n\sqrt{n}√​n​​​. The reasoning behind this can be understood as follows.

Consider the given number numnumnum which can have mmm distinct factors, namely n1,n2,...,nmn\_1, n\_2,..., n\_mn​1​​,n​2​​,...,n​m​​. Now, since the number numnumnum is divisible by nin\_in​i​​, it is also divisible by nj=num/n1n\_j=num/n\_1n​j​​=num/n​1​​ i.e. ni∗nj=numn\_i\*n\_j=numn​i​​∗n​j​​=num. Also, the largest number in such a pair can only be up to num\sqrt{num}√​num​​​ (because num×num=num\sqrt{num} \times \sqrt{num}=num√​num​​​×√​num​​​=num). Thus, we can get a significant reduction in the run-time by iterating only upto num\sqrt{num}√​num​​​ and considering such nin\_in​i​​'s and njn\_jn​j​​'s in a single pass directly.Further, if num\sqrt{num}√​num​​​ is also a factor, we have to consider the factor only once while checking for the perfect number property.We sum up all such factors and check if the given number is a Perfect Number or not. Another point to be observed is that while considering 1 as such a factor, numnumnum will also be considered as the other factor. Thus, we need to subtract numnumnum from the sumsumsum.

**Java**

**public** **boolean** **checkPerfectNumber(int** num**)** **{**

**if** **(**num **<=** 0**)** **{**

**return** **false;**

**}**

**int** sum **=** 0**;**

**for** **(int** i **=** 1**;** i **\*** i **<=** num**;** i**++)** **{**

**if** **(**num **%** i **==** 0**)** **{**

sum **+=** i**;**

**if** **(**i **\*** i **!=** num**)** **{**

sum **+=** num **/** i**;**

**}**

**}} return** sum **-** num **==** num**; }}**

**Find pattern in a string recursively**

1. **def** findPatternRec(str, pattern):
3. **if** len(str) == 1 **and** str == pattern:
4. **return** 1
5. **elif** len(str) == 1 **and** str != pattern:
6. **return** 0
8. **if** str[0:len(pattern)] == pattern:
9. # we can skip few characters if there are no repeating characters in pattern
10. **return** 1 + findPatternRec(1:], pattern)
11. **elif** str[0:len(pattern)] != pattern:
12. **return** 0 + findPatternRec(str[1:], pattern)

15. **print** findPatternRec('adbcaddclecajhgf','ca')

**Find minimum absolute difference in Array**

1. # Returns minimum difference between any pair
2. **def** findMinDiff(arr, n):
3. # Sort array in non-decreasing order
4. arr = sorted(arr)
6. # Initialize difference as infinite
7. diff = 10\*\*20
9. # Find the min diff by comparing adjacent
10. # pairs in sorted array
11. **for** i **in** range(n-1):
12. **if** arr[i+1] - arr[i] < diff:
13. diff = arr[i+1] - arr[i]
15. # Return min diff
16. **return** diff
18. # Driver code
19. arr = [1, 5, 3, 19, 18, 25]
20. n = len(arr)
21. **print**("Minimum difference is " + str(findMinDiff(arr, n)))