```
 1 # 1- Valid anagram:
 2
 3 from collections import Counter
 4
 5
 6 def are_anagrams(s1, s2):
 7     if len(s1) != len(s2):
 8         return False
 9     return Counter(s1) == Counter(s2)
10
11
12 def are_anagrams(s1, s2):
13     if len(s1) != len(s2):
14         return False
15     return sorted(s1) == sorted(s2)
16
17
18 # 2- First and last index:
19
20 def first_and_last(arr, target):
21     for i in range(len(arr)):
22         if arr[i] == target:
23             start = i
24             while i + 1 < len(arr) and arr[i
   + 1] == target:
25                 i += 1
26             return [start, i]
27     return [-1, -1]
28
29
30 def find_start(arr, target):
31     if arr[0] == target:
32         return 0
33     left, right = 0, len(arr) - 1
34     while left <= right:
35         mid = (left + right) // 2
```

```
36          if arr[mid] == target and arr[mid - 1
   ] < target:
37              return mid
38          elif arr[mid] < target:
39              left = mid + 1
40          else:
41              right = mid - 1
42      return -1
43
44
45  def find_end(arr, target):
46      if arr[-1] == target:
47          return len(arr) - 1
48      left, right = 0, len(arr) - 1
49      while left <= right:
50          mid = (left + right) // 2
51          if arr[mid] == target and arr[mid + 1
   ] > target:
52              return mid
53          elif arr[mid] > target:
54              right = mid - 1
55          else:
56              left = mid + 1
57      return -1
58
59
60  def first_and_last(arr, target):
61      if len(arr) == 0 or arr[0] > target or
   arr[-1] < target:
62          return [-1, -1]
63      start = find_start(arr, target)
64      end = find_end(arr, target)
65      return [start, end]
66
67
68  # 3- Kth largest element:
```

```
 69
 70 def kth_largest(arr, k):
 71     for i in range(k - 1):
 72         arr.remove(max(arr))
 73     return max(arr)
 74
 75
 76 def kth_largest(arr, k):
 77     n = len(arr)
 78     arr.sort()
 79     return arr[n - k]
 80
 81
 82 import heapq
 83
 84
 85 def kth_largest(arr, k):
 86     arr = [-elem for elem in arr]
 87     heapq.heapify(arr)
 88     for i in range(k - 1):
 89         heapq.heappop(arr)
 90     return -heapq.heappop(arr)
 91
 92
 93 # 4- Symmetric tree:
 94
 95 def are_symmetric(root1, root2):
 96     if root1 is None and root2 is None:
 97         return True
 98     elif ((root1 is None) != (root2 is None)
    ) or root1.val != root2.val:
 99         return False
100     else:
101         return are_symmetric(root1.left,
    root2.right) and are_symmetric(root1.right,
    root2.left)
```

```
102
103
104 def is_symmetric(root):
105     if root is None:
106         return True
107     return are_symmetric(root.left, root.
    right)
108
109
110 # 5- Generate parentheses:
111
112 def generate(n):
113     def rec(n, diff, comb, combs):
114         if diff < 0 or diff > n:
115             return
116         elif n == 0:
117             if diff == 0:
118                 combs.append(''.join(comb))
119         else:
120             comb.append('(')
121             rec(n - 1, diff + 1, comb, combs
    )
122             comb.pop()
123             comb.append(')')
124             rec(n - 1, diff - 1, comb, combs
    )
125             comb.pop()
126
127     combs = []
128     rec(2 * n, 0, [], combs)
129     return combs
130
131
132 # 6- Gas station:
133
134 def can_traverse(gas, cost, start):
```

```
135     n = len(gas)
136     remaining = 0
137     i = start
138     started = False
139     while i != start or not started:
140         started = True
141         remaining += gas[i] - cost[i]
142         if remaining < 0:
143             return False
144         i = (i + 1) % n
145     return True
146 
147 
148 def gas_station(gas, cost):
149     for i in range(len(gas)):
150         if can_traverse(gas, cost, i):
151             return i
152     return -1
153 
154 
155 def gas_station(gas, cost):
156     remaining = 0
157     prev_remaining = 0
158     candidate = 0
159     for i in range(len(gas)):
160         remaining += gas[i] - cost[i]
161         if remaining < 0:
162             candidate = i + 1
163             prev_remaining += remaining
164             remaining = 0
165     if candidate == len(gas) or remaining +
    prev_remaining < 0:
166         return -1
167     else:
168         return candidate
169 
```

```
170
171 # 7- Course schedule:
172
173 def dfs(graph, vertex, path, order, visited)
    :
174     path.add(vertex)
175     for neighbor in graph[vertex]:
176         if neighbor in path:
177             return False
178         if neighbor not in visited:
179             visited.add(neighbor)
180             if not dfs(graph, neighbor, path
    , order, visited):
181                 return False
182     path.remove(vertex)
183     order.append(vertex)
184     return True
185
186
187 def course_schedule(n, prerequisites):
188     graph = [[] for i in range(n)]
189     for pre in prerequisites:
190         graph[pre[1]].append(pre[0])
191     visited = set()
192     path = set()
193     order = []
194     for course in range(n):
195         if course not in visited:
196             visited.add(course)
197             if not dfs(graph, course, path,
    order, visited):
198                 return False
199     return True
200
201
202 from collections import deque
```

```
203
204
205 def course_schedule(n, prerequisites):
206     graph = [[] for i in range(n)]
207     indegree = [0 for i in range(n)]
208     for pre in prerequisites:
209         graph[pre[1]].append(pre[0])
210         indegree[pre[0]] += 1
211     order = []
212     queue = deque([i for i in range(n) if
    indegree[i] == 0])
213     while queue:
214         vertex = queue.popleft()
215         order.append(vertex)
216         for neighbor in graph[vertex]:
217             indegree[neighbor] -= 1
218             if indegree[neighbor] == 0:
219                 queue.append(neighbor)
220     return len(order) == n
221
222
223 # 8- Kth permutation:
224
225 import itertools
226
227
228 def kth_permutation(n, k):
229     permutations = list(itertools.
    permutations(range(1, n + 1)))
230     return ''.join(map(str, permutations[k -
     1]))
231
232
233 def kth_permutation(n, k):
234     permutation = []
235     unused = list(range(1, n + 1))
```

```
236     fact = [1] * (n + 1)
237     for i in range(1, n + 1):
238         fact[i] = i * fact[i - 1]
239     k -= 1
240     while n > 0:
241         part_length = fact[n] // n
242         i = k // part_length
243         permutation.append(unused[i])
244         unused.pop(i)
245         n -= 1
246         k %= part_length
247     return ''.join(map(str, permutation))
248
249
250 # 9- Minimum window substring:
251
252 def contains_all(freq1, freq2):
253     for ch in freq2:
254         if freq1[ch] < freq2[ch]:
255             return False
256     return True
257
258
259 def min_window(s, t):
260     n, m = len(s), len(t)
261     if m > n or m == 0:
262         return ""
263     freqt = Counter(t)
264     shortest = " " * (n + 1)
265     for length in range(1, n + 1):
266         for i in range(n - length + 1):
267             sub = s[i:i + length]
268             freqs = Counter(sub)
269             if contains_all(freqs, freqt)
    and length < len(shortest):
270                 shortest = sub
```

```
271     return shortest if len(shortest) <= n
    else ""
272
273
274 def min_window(s, t):
275     n, m = len(s), len(t)
276     if m > n or t == "":
277         return ""
278     freqt = Counter(t)
279     start, end = 0, n + 1
280     for length in range(1, n + 1):
281         freqs = Counter()
282         satisfied = 0
283         for ch in s[:length]:
284             freqs[ch] += 1
285             if ch in freqt and freqs[ch] ==
    freqt[ch]:
286                 satisfied += 1
287         if satisfied == len(freqt) and
    length < end - start:
288             start, end = 0, length
289         for i in range(1, n - length + 1):
290             freqs[s[i + length - 1]] += 1
291             if s[i + length - 1] in freqt
    and freqs[s[i + length - 1]] == freqt[s[i +
    length - 1]]:
292                 satisfied += 1
293             if s[i - 1] in freqt and freqs[s
    [i - 1]] == freqt[s[i - 1]]:
294                 satisfied -= 1
295             freqs[s[i - 1]] -= 1
296             if satisfied == len(freqt) and
    length < end - start:
297                 start, end = i, i + length
298     return s[start:end] if end - start <= n
    else ""
```

```
299
300
301 def min_window(s, t):
302     n, m = len(s), len(t)
303     if m > n or t == "":
304         return ""
305     freqt = Counter(t)
306     start, end = 0, n
307     satisfied = 0
308     freqs = Counter()
309     left = 0
310     for right in range(n):
311         freqs[s[right]] += 1
312         if s[right] in freqt and freqs[s[
    right]] == freqt[s[right]]:
313             satisfied += 1
314         if satisfied == len(freqt):
315             while s[left] not in freqt or
    freqs[s[left]] > freqt[s[left]]:
316                 freqs[s[left]] -= 1
317                 left += 1
318             if right - left + 1 < end -
    start + 1:
319                 start, end = left, right
320     return s[start:end + 1] if end - start +
     1 <= n else ""
321
322
323 # 10- Largest rectangle in histogram:
324
325 def largest_rectangle(heights):
326     max_area = 0
327     for i in range(len(heights)):
328         left = i
329         while left - 1 >= 0 and heights[left
     - 1] >= heights[i]:
```

```
330             left -= 1
331         right = i
332         while right + 1 < len(heights) and
    heights[right + 1] >= heights[i]:
333             right += 1
334         max_area = max(max_area, heights[i]
    * (right - left + 1))
335     return max_area
336
337
338 def rec(heights, low, high):
339     if low > high:
340         return 0
341     elif low == high:
342         return heights[low]
343     else:
344         minh = min(heights[low:high + 1])
345         pos_min = heights.index(minh, low,
    high + 1)
346         from_left = rec(heights, low,
    pos_min - 1)
347         from_right = rec(heights, pos_min +
    1, high)
348         return max(from_left, from_right,
    minh * (high - low + 1))
349
350
351 def largest_rectangle(heights):
352     return rec(heights, 0, len(heights) - 1)
353
354
355 def largest_rectangle(heights):
356     heights = [-1] + heights + [-1]
357     from_left = [0] * len(heights)
358     stack = [0]
359     for i in range(1, len(heights) - 1):
```

```
360         while heights[stack[-1]] >= heights[
    i]:
361             stack.pop()
362         from_left[i] = stack[-1]
363         stack.append(i)
364     from_right = [0] * len(heights)
365     stack = [len(heights) - 1]
366     for i in range(1, len(heights) - 1)[::-1
    ]:
367         while heights[stack[-1]] >= heights[
    i]:
368             stack.pop()
369         from_right[i] = stack[-1]
370         stack.append(i)
371     max_area = 0
372     for i in range(1, len(heights) - 1):
373         max_area = max(max_area, heights[i]
    * (from_right[i] - from_left[i] - 1))
374     return max_area
375
376
377 def largest_rectangle(heights):
378     heights = [-1] + heights + [-1]
379     max_area = 0
380     stack = [(0, -1)]
381     for i in range(1, len(heights)):
382         start = i
383         while stack[-1][1] > heights[i]:
384             top_index, top_height = stack.
    pop()
385             max_area = max(max_area,
    top_height * (i - top_index))
386             start = top_index
387         stack.append((start, heights[i]))
388     return max_area
389
```

```
390
391 @TabotCharlesBessong
392
393
394 TabotCharlesBessong
395 commented
396 on
397 Jan
398 12, 2022
399 Nice
400 one, thank
401 you
402
403
404 @ismaeldevmw
405
406
407 ismaeldevmw
408 commented
409 on
410 Jan
411 14, 2022
412 Hello, In
413 the
414 "First and last index"
415 problem, why
416 doesn
417 't work the first solution with Python 3? I
    tried it in replit.com
418
419
420 @syphh
421
422
423 Author
424 syphh
```

```
commented
on
Jan
14, 2022 •
@ismaeldevmw


Hello, make
s# 1- Valid anagram:

from collections import Counter

def are_anagrams(s1, s2):
    if len(s1) != len(s2):
        return False
    return Counter(s1) == Counter(s2)


def are_anagrams(s1, s2):
    if len(s1) != len(s2):
        return False
    return sorted(s1) == sorted(s2)


# 2- First and last index:

def first_and_last(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            start = i
            while i+1 < len(arr) and arr[i+1
] == target:
                i += 1
            return [start, i]
    return [-1, -1]

```

```
460 def find_start(arr, target):
461     if arr[0] == target:
462         return 0
463     left, right = 0, len(arr)-1
464     while left <= right:
465         mid = (left+right)//2
466         if arr[mid] == target and arr[mid-1]
    < target:
467             return mid
468         elif arr[mid] < target:
469             left = mid+1
470         else:
471             right = mid-1
472     return -1
473
474
475 def find_end(arr, target):
476     if arr[-1] == target:
477         return len(arr)-1
478     left, right = 0, len(arr)-1
479     while left <= right:
480         mid = (left+right)//2
481         if arr[mid] == target and arr[mid+1]
    > target:
482             return mid
483         elif arr[mid] > target:
484             right = mid-1
485         else:
486             left = mid+1
487     return -1
488
489
490 def first_and_last(arr, target):
491     if len(arr) == 0 or arr[0] > target or
   arr[-1] < target:
492         return [-1, -1]
```

```
493     start = find_start(arr, target)
494     end = find_end(arr, target)
495     return [start, end]
496
497
498 # 3- Kth largest element:
499
500 def kth_largest(arr, k):
501     for i in range(k-1):
502         arr.remove(max(arr))
503     return max(arr)
504
505 def kth_largest(arr, k):
506     n = len(arr)
507     arr.sort()
508     return arr[n-k]
509
510
511 import heapq
512
513 def kth_largest(arr, k):
514     arr = [-elem for elem in arr]
515     heapq.heapify(arr)
516     for i in range(k - 1):
517         heapq.heappop(arr)
518     return -heapq.heappop(arr)
519
520
521 # 4- Symmetric tree:
522
523 def are_symmetric(root1, root2):
524     if root1 is None and root2 is None:
525         return True
526     elif ((root1 is None) != (root2 is None)
    ) or root1.val != root2.val:
527         return False
```

```
528     else:
529         return are_symmetric(root1.left,
    root2.right) and are_symmetric(root1.right,
    root2.left)
530
531 def is_symmetric(root):
532     if root is None:
533         return True
534     return are_symmetric(root.left, root.
    right)
535
536
537 # 5- Generate parentheses:
538
539 def generate(n):
540     def rec(n, diff, comb, combs):
541         if diff < 0 or diff > n:
542             return
543         elif n == 0:
544             if diff == 0:
545                 combs.append(''.join(comb))
546         else:
547             comb.append('(')
548             rec(n-1, diff+1, comb, combs)
549             comb.pop()
550             comb.append(')')
551             rec(n-1, diff-1, comb, combs)
552             comb.pop()
553     combs = []
554     rec(2*n, 0, [], combs)
555     return combs
556
557
558 # 6- Gas station:
559
560 def can_traverse(gas, cost, start):
```

```
561     n = len(gas)
562     remaining = 0
563     i = start
564     started = False
565     while i != start or not started:
566         started = True
567         remaining += gas[i] - cost[i]
568         if remaining < 0:
569             return False
570         i = (i+1)%n
571     return True
572
573
574 def gas_station(gas, cost):
575     for i in range(len(gas)):
576         if can_traverse(gas, cost, i):
577             return i
578     return -1
579
580
581 def gas_station(gas, cost):
582     remaining = 0
583     prev_remaining = 0
584     candidate = 0
585     for i in range(len(gas)):
586         remaining += gas[i] - cost[i]
587         if remaining < 0:
588             candidate = i+1
589             prev_remaining += remaining
590             remaining = 0
591     if candidate == len(gas) or remaining+
    prev_remaining < 0:
592         return -1
593     else:
594         return candidate
595
```

```
596 # 7- Course schedule:
597
598 def dfs(graph, vertex, path, order, visited)
    :
599     path.add(vertex)
600     for neighbor in graph[vertex]:
601         if neighbor in path:
602             return False
603         if neighbor not in visited:
604             visited.add(neighbor)
605             if not dfs(graph, neighbor, path
    , order, visited):
606                 return False
607     path.remove(vertex)
608     order.append(vertex)
609     return True
610
611
612 def course_schedule(n, prerequisites):
613     graph = [[] for i in range(n)]
614     for pre in prerequisites:
615         graph[pre[1]].append(pre[0])
616     visited = set()
617     path = set()
618     order = []
619     for course in range(n):
620         if course not in visited:
621             visited.add(course)
622             if not dfs(graph, course, path,
    order, visited):
623                 return False
624     return True
625
626
627 from collections import deque
628
```

```
def course_schedule(n, prerequisites):
    graph = [[] for i in range(n)]
    indegree = [0 for i in range(n)]
    for pre in prerequisites:
        graph[pre[1]].append(pre[0])
        indegree[pre[0]] += 1
    order = []
    queue = deque([i for i in range(n) if
indegree[i] == 0])
    while queue:
        vertex = queue.popleft()
        order.append(vertex)
        for neighbor in graph[vertex]:
            indegree[neighbor] -= 1
            if indegree[neighbor] == 0:
                queue.append(neighbor)
    return len(order) == n


# 8- Kth permutation:

import itertools

def kth_permutation(n, k):
    permutations = list(itertools.
permutations(range(1, n+1)))
    return ''.join(map(str, permutations[k-1
]))


def kth_permutation(n, k):
    permutation = []
    unused = list(range(1, n+1))
    fact = [1]*(n+1)
    for i in range(1, n+1):
        fact[i] = i*fact[i-1]
```

```
662     k -= 1
663     while n > 0:
664         part_length = fact[n]//n
665         i = k//part_length
666         permutation.append(unused[i])
667         unused.pop(i)
668         n -= 1
669         k %= part_length
670     return ''.join(map(str, permutation))
671
672
673 # 9- Minimum window substring:
674
675 def contains_all(freq1, freq2):
676     for ch in freq2:
677         if freq1[ch] < freq2[ch]:
678             return False
679     return True
680
681
682 def min_window(s, t):
683     n, m = len(s), len(t)
684     if m > n or m == 0:
685         return ""
686     freqt = Counter(t)
687     shortest = " "*(n+1)
688     for length in range(1, n+1):
689         for i in range(n-length+1):
690             sub = s[i:i+length]
691             freqs = Counter(sub)
692             if contains_all(freqs, freqt)
    and length < len(shortest):
693                 shortest = sub
694     return shortest if len(shortest) <= n
    else ""
695
```

```
696
697 def min_window(s, t):
698     n, m = len(s), len(t)
699     if m > n or t == "":
700         return ""
701     freqt = Counter(t)
702     start, end = 0, n+1
703     for length in range(1, n+1):
704         freqs = Counter()
705         satisfied = 0
706         for ch in s[:length]:
707             freqs[ch] += 1
708             if ch in freqt and freqs[ch] ==
    freqt[ch]:
709                 satisfied += 1
710         if satisfied == len(freqt) and
    length < end-start:
711             start, end = 0, length
712         for i in range(1, n-length+1):
713             freqs[s[i+length-1]] += 1
714             if s[i+length-1] in freqt and
    freqs[s[i+length-1]] == freqt[s[i+length-1]]
    :
715                 satisfied += 1
716             if s[i-1] in freqt and freqs[s[i
    -1]] == freqt[s[i-1]]:
717                 satisfied -= 1
718             freqs[s[i-1]] -= 1
719             if satisfied == len(freqt) and
    length < end-start:
720                 start, end = i, i+length
721     return s[start:end] if end-start <= n
    else ""
722
723
724 def min_window(s, t):
```

```
725     n, m = len(s), len(t)
726     if m > n or t == "":
727         return ""
728     freqt = Counter(t)
729     start, end = 0, n
730     satisfied = 0
731     freqs = Counter()
732     left = 0
733     for right in range(n):
734         freqs[s[right]] += 1
735         if s[right] in freqt and freqs[s[
    right]] == freqt[s[right]]:
736             satisfied += 1
737         if satisfied == len(freqt):
738             while s[left] not in freqt or
    freqs[s[left]] > freqt[s[left]]:
739                 freqs[s[left]] -= 1
740                 left += 1
741             if right-left+1 < end-start+1:
742                 start, end = left, right
743     return s[start:end+1] if end-start+1 <=
    n else ""
744
745
746 # 10- Largest rectangle in histogram:
747
748 def largest_rectangle(heights):
749     max_area = 0
750     for i in range(len(heights)):
751         left = i
752         while left-1 >= 0 and heights[left-1
    ] >= heights[i]:
753             left -= 1
754         right = i
755         while right+1 < len(heights) and
    heights[right+1] >= heights[i]:
```

```
756                 right += 1
757             max_area = max(max_area, heights[i]*
    (right-left+1))
758         return max_area
759
760
761 def rec(heights, low, high):
762     if low > high:
763         return 0
764     elif low == high:
765         return heights[low]
766     else:
767         minh = min(heights[low:high+1])
768         pos_min = heights.index(minh, low,
    high+1)
769         from_left = rec(heights, low,
    pos_min-1)
770         from_right = rec(heights, pos_min+1,
     high)
771         return max(from_left, from_right,
    minh*(high-low+1))
772
773
774 def largest_rectangle(heights):
775     return rec(heights, 0, len(heights)-1)
776
777
778 def largest_rectangle(heights):
779     heights = [-1]+heights+[-1]
780     from_left = [0]*len(heights)
781     stack = [0]
782     for i in range(1, len(heights)-1):
783         while heights[stack[-1]] >= heights[
    i]:
784             stack.pop()
785         from_left[i] = stack[-1]
```

```
786         stack.append(i)
787     from_right = [0]*len(heights)
788     stack = [len(heights)-1]
789     for i in range(1, len(heights)-1)[::-1]:
790         while heights[stack[-1]] >= heights[
    i]:
791             stack.pop()
792         from_right[i] = stack[-1]
793         stack.append(i)
794     max_area = 0
795     for i in range(1, len(heights)-1):
796         max_area = max(max_area, heights[i]*
    (from_right[i]-from_left[i]-1))
797     return max_area
798
799
800 def largest_rectangle(heights):
801     heights = [-1]+heights+[-1]
802     max_area = 0
803     stack = [(0, -1)]
804     for i in range(1, len(heights)):
805         start = i
806         while stack[-1][1] > heights[i]:
807             top_index, top_height = stack.
    pop()
808             max_area = max(max_area,
    top_height*(i-top_index))
809             start = top_index
810         stack.append((start, heights[i]))
811     return max_area
812
```