Advanced Time and Space Complexity Cheat Sheet (with Examples)

O(1) Constant Time

```
Accessing an element by index:
Example:
  arr = [10, 20, 30]
  print(arr[1])  # Output: 20
```

O(log n) Logarithmic Time

```
Binary search on sorted list:
Example:
  def binary_search(arr, target):
      low, high = 0, len(arr) - 1
      while low <= high:
          mid = (low + high) // 2
          if arr[mid] == target:
              return mid
          elif arr[mid] < target:
              low = mid + 1
          else:
              high = mid - 1
      return -1
```

O(n) Linear Time

```
Traversing a list:
Example:
  for item in [1, 2, 3, 4]:
      print(item)
```

O(n log n) Linearithmic Time

```
Merge sort (simplified structure):
Example:
  def merge_sort(arr):
      if len(arr) <= 1:
          return arr
      mid = len(arr) // 2
      left = merge_sort(arr[:mid])
      right = merge_sort(arr[mid:])
      return merge(left, right)
```

O(n^2) Quadratic Time

```
Nested loop through 2D array:
```

Advanced Time and Space Complexity Cheat Sheet (with Examples)

```
Example:
  for i in range(n):
      for j in range(n):
          print(i, j)
```

O(2^n) Exponential Time

```
Fibonacci without memoization:
Example:
  def fib(n):
      if n <= 1:
          return n
      return fib(n-1) + fib(n-2)
```

O(n!) Factorial Time

```
Permutations of list:
Example:
  def permute(nums):
      if len(nums) <= 1:
          return [nums]
      res = []
      for i in range(len(nums)):
          for p in permute(nums[:i] + nums[i+1:]):
              res.append([nums[i]] + p)
      return res
```

Space Complexity O(1), O(n), O(log n)

```
O(1): Swap values in-place
  a, b = 1, 2
  a, b = b, a

O(n): Create copy of list
  new_list = arr[:]

O(log n): Binary search recursive stack
  def bs(l, r):
      if l > r:
          return
      mid = (l + r) // 2
      bs(l, mid - 1)
      bs(mid + 1, r)
```