

① Intersection of Unsorted Arrays

Name

- Initialise $res = 0$
- Traverse through every element of $a[]$
 - Check if it has not appeared already
 - If a new element and also present in $b[]$, do $res++$
- Return res

Time: $O(m+n)$

Space: $O(1)$

```
def intersection(a[], m, a[], n):
    res = 0
    for i in range(m):
        flag = False
        for j in range(n):
            if a[i] == a[j]:
                flag = True
                break
        if flag == True:
            continue
        for j in range(n):
            if a[i] == a[j]:
                res += 1
                break
    return res
```

Efficient

- Insert all elements of $a[]$ in a set (s_a) $\Theta(m)$
- Insert all elements of $b[]$ in another set (s_b) $\Theta(n)$
- Now traverse through $a[]$ and increment count for elements that are present in s_b also $\Theta(m)$

Time: $\Theta(m+n)$ Space: $\Theta(m+n)$

1) Insert all elements of $a[]$ in set (s_a) $\Theta(m)$

2) Traverse through $b[]$. Search for every element $b[i]$ in s_a . If present:

- increment res
- remove $b[i]$ from s_a

Time: $\Theta(m+n)$ Space: $\Theta(m)$

```
def intersection(a[], m, a[], n):
    s_a = set()
    for i in range(m):
        s_a.add(a[i])
    res = 0
    for i in range(n):
        if a[i] in s_a:
            res += 1
            s_a.remove(a[i])
    return res
```

② Union of Two Unsorted Arrays

Name

```
def unionQ(a[], m, a[], n):
    c = [0] * (m+n)
    for i in range(m):
        c[i] = a[i]
    for i in range(n):
        c[i+m] = a[i]
    res = 0
    for k in range(m+n):
        flag = False
        for j in range(i):
            if c[i] == c[j]:
                flag = True
                break
        if flag == False:
            res += 1
    return res
```

Time: $O(m+n)$ Space: $O(m+n)$

Efficient

```
def unionQ(a[], m, a[], n):
    s_a = set()
    for i in a:
        s_a.add(i)
    for i in a2:
        s_a.add(i)
    return len(s_a)
```

Time: $O(m+n)$ Space: $O(m+n)$

③ All with given sum in Unsorted Array

Name

```
def posWithSum(a[], n):
    for i in range(len(a)):
        for j in range(i+1, len(a)):
            if a[i] + a[j] == n:
                return 1
    return 0
```

Time: $O(n^2)$ Space: $O(1)$

Efficient

```
def posWithSum(a[], n):
    s_a = set()
    for i in a:
        if n-i in s_a:
            return 1
        s_a.add(i)
    return 0
```

Time: $O(n)$ Space: $O(n)$

④ Subarray with 0 sum in Python

Name

```
def isZeroSum(a[]):
    n = len(a)
    for i in range(n):
        for j in range(i+1, n):
            if sum(a[i:j]) == 0:
                return True
    return False
```

Time: $O(n^2)$

Efficient

Idea: Using Prefix sum and Hashing

pre-com

go as $a_1, \dots, a_2, \dots, a_3, \dots, a_4, \dots, a_n$

→ If sum of $a[i:j]$ is 0, then pre-com must be same as pre-com

```
def isZeroSum(a[]):
    pre_com = 0
    n = len(a)
    for i in range(len(a)):
        pre_com += a[i]
        if pre_com == 0 or pre_com in h:
            return True
        h.add(pre_com)
    return False
```

Time: $O(n)$

⑤ Subarray with Given Sum

(There are no negative elements in array)

Name

```
def isSubSum(a[], com):
    for i in range(len(a)):
        curr = 0
        for j in range(i, len(a)):
            curr += a[j]
            if curr == com:
                return True
    return False
```

Efficient

We use window sliding technique with a window of variable size.

```
def isSubSum(a[], com):
    c = 0
    curr = 0
    for i in range(len(a)):
        curr += a[i]
        while curr > com:
            curr -= a[i]
        if curr == com:
            return True
    return False
```

Time: $O(n)$ Space: $O(1)$

⑥ Check for Palindrome Permutation

Answer is going to be true if there is at least one character with odd frequency.

abba, abccbab, aabbbaabbb

The false
abcc, ab, aabbbaabbb
→ Let $(e_1, e_2, \dots, e_k, e_k)$ be pair of even frequency characters, we can always form a palindrome $e_1 \dots e_k \dots e_k e_{k-1} \dots e_1$. If there is one odd frequency character, we can still form a palindrome $e_1 e_2 \dots e_k e_{k+1} \dots e_k e_1$

⑦ Longest Subarray with Given Sum

Name

```
def longestSubarrayWithSum(a[], com):
    n = len(a)
    res = 0
    for i in range(n):
        curr_com = 0
        for j in range(i, n):
            curr_com += a[j]
            if curr_com == com:
                res = max(res, j-i+1)
    return res
```

Time: $O(n^2)$ Space: $O(1)$

Efficient

Time: $O(n)$

```
def longestSubarrayWithSum(a[], com):
    n = len(a)
    mydict = dict()
    pre_com = 0
    res = 0
    for i in range(n):
        pre_com += a[i]
        if pre_com == com:
            res = i+1
        if pre_com not in mydict:
            mydict[pre_com] = i
        if pre_com - com in mydict:
            res = max(res, i-mydict[pre_com])
    return res
```

⑧ Longest Subarray with equal number of zero and one

Name

```
def longestZeroSubarray(a[]):
    n = len(a)
    res = 0
    for i in range(n):
        c_0 = 0
        c_1 = 0
        for j in range(i, n):
            if a[j] == 0:
                c_0 += 1
            else:
                c_1 += 1
            if c_0 == c_1:
                res = max(res, j-i+1)
    return res
```

Time: $O(n^2)$ Space: $O(1)$

Efficient

Time: $O(n)$

```
def longestZeroSubarray(a[]):
    n = len(a)
    for i in range(n):
        if a[i] == 0:
            a[i] = -1
    mydict = dict()
    curr = 0
    maxlen = 0
    for i in range(n):
        curr += a[i]
        if curr == 0:
            maxlen = i+1
        if curr in mydict:
            maxlen = max(maxlen, i-mydict[curr])
        else:
            mydict[curr] = i
    return maxlen
```

Time: $O(n)$ Space: $O(n)$

⑨ Longest Common Span in Binary Array

→ We are given two binary subarrays of some sizes

Name

```
def longestCommonSpan(a[], a[]):
    n1 = len(a)
    n2 = len(a[])
    sum1 = 0
    sum2 = 0
    for i in range(n1):
        sum1 += a[i]
    for j in range(n2):
        sum2 += a[j]
    if sum1 == sum2:
        res = max(n1, n2)
    else:
        res = min(n1, n2)
```

Efficient

Problem is going to reduce into problem of longest subarray with 0 sum in array.

- Compute a difference array
 $\text{temp}[i] = a[i] - a[2i]$
- Return length of the longest subarray with 0 sum in temp.

Time: $O(n)$ Space: $O(n)$

```
def longestCommonSpan(a[], a[]):
    n1 = len(a)
    n2 = len(a[])
    sum1 = 0
    sum2 = 0
    for i in range(n1):
        sum1 += a[i]
    for j in range(n2):
        sum2 += a[j]
    if sum1 == sum2:
        res = max(n1, n2)
    else:
        temp = [0] * n1
        for i in range(n1):
            temp[i] = a[i] - a[2*i]
        mydict = dict()
        curr = 0
        maxlen = 0
        for i in range(n1):
            curr += temp[i]
            if curr == 0:
                maxlen = i+1
            if curr in mydict:
                maxlen = max(maxlen, i-mydict[curr])
            else:
                mydict[curr] = i
        return maxlen
```

longest subarray with zero sum

⑩ Longest Consecutive Subsequence

We need to find the longest subsequence in the form of $x, x+1, \dots, x+k$ with these elements appearing in any order.

Name

```
def findLargest(a[]):
    n = len(a)
    a1 = set()
    res = 1
    for i in range(n):
        if a[i] == a[i-1]+1:
            a1.add(a[i])
        else:
            res = max(res, len(a1))
            a1 = set()
    return res
```

Efficient

We first insert all elements in a hash table.

Then with 2 lookups, we find result.

def findLargest(a[]):

Time: $O(n)$ Space: $O(n)$

Time: $O(n \log n)$ Space: $O(n)$