In [5]:

```
def sort(array, size):
    for i in range(size):
        mi = i
        for j in range(i + 1, size):
            if array[j] < array[mi]:
                mi = j
        (array[i], array[mi]) = (array[mi], array[i])
arr = [-2, 45, 0, 11, -9,88,-97,-202,747]
size = len(arr)
sort(arr, size)
print('The array of selection sort is:')
print(arr)
```

```
The array of selection sort is:
[-202, -97, -9, -2, 0, 11, 45, 88, 747]
```

# 11. Sort Characters By Frequency

In [7]:

```
from collections import Counter
def sort_by_freq(s):
    return ''.join(char * count for char, count in Counter(s).most_common()
s = "tree"
sorted_s = sort_by_freq(s)
print(sorted_s)
```

```
eetr
```

# intersection of three

In [9]:

```
def inte_of_three(arr1, arr2, arr3):
    i, j, k = 0, 0, 0
    result = []
    while i < len(arr1) and j < len(arr2) and k < len(arr3):
        if arr1[i] == arr2[j] == arr3[k]:
            result.append(arr1[i])
            i += 1
            j += 1
            k += 1
        elif arr1[i] < arr2[j]:
            i += 1
        elif arr2[j] < arr3[k]:
            j += 1
        else:
            k += 1
    return result
arr1 = [1, 2, 4, 5, 6]
arr2 = [2, 4, 6, 8]
arr3 = [2, 4, 6, 8, 10]
print(inte_of_three(arr1, arr2, arr3))
```

```
[2, 4, 6]
```

# 14.diagonal sort

In [10]:

```
def diagonal_sort(mat):
    from collections import defaultdict
    import heapq
    n, m = len(mat), len(mat[0])
    diagonals = defaultdict(list)
    for i in range(n):
        for j in range(m):
            heapq.heappush(diagonals[i - j], mat[i][j])
    for i in range(n):
        for j in range(m):
            mat[i][j] = heapq.heappop(diagonals[i - j])
    return mat
mat = [
    [3, 3, 1, 1],
    [2, 2, 1, 2],
    [1, 1, 1, 2]
]
sorted_mat = diagonal_sort(mat)
for row in sorted_mat:
    print(row)
```

```
[1, 1, 1, 1]
[1, 2, 2, 2]
[1, 2, 3, 3]
```

# 10.insertion sort

In [11]:
```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def insertion_sort_list(head):
    dummy = ListNode(0)
    curr = head
    while curr:
        prev = dummy
        while prev.next and prev.next.val < curr.val:
            prev = prev.next
        next_temp = curr.next
        curr.next = prev.next
        prev.next = curr
        curr = next_temp
    return dummy.next
def print_list(head):
    while head:
        print(head.val, end=" -> ")
        head = head.next
    print("None")
nodes = [4, 2, 1, 3]
head = ListNode(nodes[0])
current = head
for value in nodes[1:]:
    current.next = ListNode(value)
    current = current.next
sorted_head = insertion_sort_list(head)
print_list(sorted_head)
```

```
1 -> 2 -> 3 -> 4 -> None
```

# Sort Colors

In [12]:
```python
def sort_colors(nums):
    low, mid, high = 0, 0, len(nums) - 1
    while mid <= high:
        if nums[mid] == 0:
            nums[low], nums[mid] = nums[mid], nums[low]
            low += 1
            mid += 1
        elif nums[mid] == 1:
            mid += 1
        else:
            nums[high], nums[mid] = nums[mid], nums[high]
            high -= 1
    return nums
nums = [2, 0, 2, 1, 1, 0]
sorted_colors = sort_colors(nums)
print(sorted_colors)
```

```
[0, 0, 1, 1, 2, 2]
```

# Merge k Sorted Lists

In [13]:

```
import heapq
def merge_k_sorted_lists(lists):
    heap = []
    for i in range(len(lists)):
        if lists[i]:
            heapq.heappush(heap, (lists[i][0], i, 0))
    result = []
    while heap:
        val, list_idx, elem_idx = heapq.heappop(heap)
        result.append(val)
        if elem_idx + 1 < len(lists[list_idx]):
            next_tuple = (lists[list_idx][elem_idx + 1], list_idx, elem_idx
            heapq.heappush(heap, next_tuple)
    return result
lists = [
    [1, 4, 5],
    [1, 3, 4],
    [2, 6]
]
merged_list = merge_k_sorted_lists(lists)
print(merged_list)
```

```
[1, 1, 2, 3, 4, 4, 5, 6]
```

# 3.duplicate from sorted array

In [14]:

```
def remove_duplicates(nums):
    if not nums:
        return 0
    unique_idx = 0
    for i in range(1, len(nums)):
        if nums[i] != nums[unique_idx]:
            unique_idx += 1
            nums[unique_idx] = nums[i]
    return unique_idx + 1
nums = [1, 1, 2, 2, 3]
length = remove_duplicates(nums)
print(nums[:length])
```

```
[1, 2, 3]
```

# 5. find first and last position

In [15]:

```
def find_first_and_last(nums, target):
    def binary_search_left(nums, target):
        left, right = 0, len(nums)
        while left < right:
            mid = (left + right) // 2
            if nums[mid] < target:
                left = mid + 1
            else:
                right = mid
        return left
    def binary_search_right(nums, target):
        left, right = 0, len(nums)
        while left < right:
            mid = (left + right) // 2
            if nums[mid] <= target:
                left = mid + 1
            else:
                right = mid
        return left
    left_idx = binary_search_left(nums, target)
    right_idx = binary_search_right(nums, target) - 1
    if left_idx <= right_idx and left_idx < len(nums) and nums[left_idx] ==
        return [left_idx, right_idx]
    else:
        return [-1, -1]
nums = [5, 7, 7, 8, 8, 10]
target = 8
positions = find_first_and_last(nums, target)
print(positions)
```

```
[3, 4]
```

# 4.target search

In [17]:

```
def search(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        if nums[left] <= nums[mid]:
            if nums[left] <= target < nums[mid]:
                right = mid - 1
            else:
                left = mid + 1
        else:
            if nums[mid] < target <= nums[right]:
                left = mid + 1
            else:
                right = mid - 1
    return -1
nums =  [5,7,7,8,8,10]
target=8
index = search(nums, target)
print(index)
```

```
4
```

# 7.duplicate from list

In [18]:

```
def remove_dupli(nums):
    if not nums:
        return nums
    unique_idx = 0
    for i in range(1, len(nums)):
        if nums[i] != nums[unique_idx]:
            unique_idx += 1
            nums[unique_idx] = nums[i]
    return nums[:unique_idx + 1]
nums = [1, 1, 2, 2, 3, 3]
unique_nums = remove_dupli(nums)
print(unique_nums)
```

```
[1, 2, 3]
```

# 8.Merge Sorted Array

In [19]:

```
def merge_sorted_array(nums1, m, nums2, n):
    p1, p2 = m - 1, n - 1
    p = m + n - 1
    while p1 >= 0 and p2 >= 0:
        if nums1[p1] > nums2[p2]:
            nums1[p] = nums1[p1]
            p1 -= 1
        else:
            nums1[p] = nums2[p2]
            p2 -= 1
        p -= 1
    nums1[:p2 + 1] = nums2[:p2 + 1]
nums1 = [1, 2, 3, 0, 0, 0]
m = 3
nums2 = [2, 5, 6]
n = 3
merge_sorted_array(nums1, m, nums2, n)
print(nums1)
```

```
[1, 2, 2, 3, 5, 6]
```

# merge two sort

In [20]:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def merge_two_sorted_lists(l1, l2):
    dummy = ListNode()
    current = dummy

    while l1 and l2:
        if l1.val < l2.val:
            current.next = l1
            l1 = l1.next
        else:
            current.next = l2
            l2 = l2.next
        current = current.next
    if l1:
        current.next = l1
    else:
        current.next = l2
    return dummy.next
l1 = ListNode(1, ListNode(2, ListNode(4)))
l2 = ListNode(1, ListNode(3, ListNode(4)))
merged_list = merge_two_sorted_lists(l1, l2)
while merged_list:
    print(merged_list.val, end=" ")
    merged_list = merged_list.next
```

```
1 1 2 3 4 4
```

# 12. max chunks

In [21]:

```
def max_chunks_to_sorted(arr):
    max_chunks = 0
    max_val = 0
    for i, val in enumerate(arr):
        max_val = max(max_val, val)
        if max_val == i:
            max_chunks += 1
    return max_chunks
arr = [4, 3, 2, 1, 0]
print(max_chunks_to_sorted(arr))
```

```
1
```

In [22]:

```
def add(a,b):
    c=[[0,0],[0,0]]
    for i in range(2):
        for j in range(2):
            c[i][j]=a[i][j]+b[i][j]
    return c
def mul(a,b):
    c=[[0,0],[0,0]]
    for i in range(2):
        for j in range(2):
            for k in range(2):
             c[i][j]+=a[i][k]*b[k][j]
    return c
def sub(a,b):
    c=[[0,0],[0,0]]
    for i in range(2):
        for j in range(2):
            c[i][j]=a[i][j]-b[i][j]
    return c
a=[[1,2,3,4],
   [5,6,7,8],
   [1,2,3,4],
   [5,6,7,8]]
b=[[1,2,1,3],
   [1,4,1,5],
   [1,6,1,7],
   [1,8,1,9]]
a11=[[a[0][0],a[0][1]],
     [a[1][0],a[1][1]]]
a12=[[a[0][2],a[0][3]],
     [a[1][2],a[1][3]]]
a21=[[a[2][0],a[2][1]],
     [a[3][0],a[3][1]]]
a22=[[a[2][2],a[2][3]],
     [a[3][2],a[3][3]]]
b11=[[b[0][0],b[0][1]],[b[1][0],b[1][1]]]
b12=[[b[0][2],b[0][3]],[b[1][2],b[1][3]]]
b21=[[b[2][0],b[2][1]],[b[3][0],b[3][1]]]
b22=[[b[2][2],b[2][3]],[b[3][2],b[3][3]]]
m1=mul(add(a11,a22),add(b11,b22))
m2=mul(add(a21,a22),b11)
m3=mul(a11,sub(b12,b22))
m4=mul(a22,sub(b21,b11))
m5=mul(add(a11,a12),b22)
m6=mul(sub(a21,a11),add(b11,b12))
m7=mul(sub(a12,a22),add(b21,b22))
c1=add(sub(add(m1,m4),m5),m7)
c2=add(m3,m5)
c3=add(m2,m4)
c4=add(add(sub(m1,m2),m3),m6)
print(c1[0],c2[0])
print(c1[1],c2[1])
print(c3[0],c4[0])
print(c3[1],c4[1])
```

```
[10, 60] [10, 70]
[26, 140] [26, 166]
[10, 60] [10, 70]
[26, 140] [26, 166]
```

In [ ]: