Ahmed Abd-Elsalam Muhammed Afify

Algorithms H.W 2.1

1) Recursion Binary Search:

In [99]:
```
'''
# -------------------- Binary search implementation with iterative algorithm ----
def search_iter(arr, l, r, element):
    while l <= r:
        mid = int((l+r)/2)
        if element == arr[mid]:
            return mid
        elif element>arr[mid]:
            l = mid+1
        else:
            r = mid-1
    return -1
'''
# -------------------- Binary search implementation with recursion algorithm ----
def search_recur(arr, l, r, element):
    if l > r:
        return -1
    else:
        mid = int((l+r)/2)
        if element == arr[mid]:
            return mid
        elif element>arr[mid]:
            return search_recur(arr, mid+1, r, element)
        else:
            return search_recur(arr, l, mid-1, element)

# ------------------ searching function -------------------
def binary_search():
    x = input("Enter numbers: ")
    y = input("Elements to search about: ")
    lst = list(map(float, x.strip().split(" ")))
    elem = list(map(float, y.strip().split(" ")))
    lst.sort()
    indeces_iter = []
    indeces_recur = []
    for i in elem:
        #i_index_iter = search_iter(lst, 0, len(lst)-1, i)
        #indeces_iter.append(i_index_iter)

        i_index_recur = search_recur(lst, 0, len(lst)-1, i)
        indeces_recur.append(i_index_recur)
    return indeces_recur #, indeces_iter
```

In [100]:
```
s_recur  = binary_search()
```

```
Enter numbers: 1 5 8 12 13
Elements to search about: 8 1 23 1 11
```

In [101]:
```
print(s_recur)
```

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

2) Merge Sort:

In [70]:
```
def mergeSort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]

        mergeSort(L)
        mergeSort(R)

        # HERE WE COMPARE ELEMENTS OF EACH SUB-LIST AND THEN WE SORT THEM
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        # Chck for left elements
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
```

In [75]:
```
x = [38, 27, 48, 3, 9, 82, 10]
mergeSort(x)
```

In [76]:
```
print(x)
```

```
[3, 9, 10, 27, 38, 48, 82]
```

3) Quick Sort:

In [104]:
```
def partition(arr, L, R):

    pivot = arr[R]
    counter = L-1
    for i in range(L, R-1):
        if arr[i] < pivot:
            counter += 1
            arr[counter], arr[i]  = arr[i], arr[counter]
    if arr[counter+1] > arr[R]:
        arr[counter+1], arr[R]  = arr[R], arr[counter+1]

    return (counter+1)

# -------------------- SORTING ALGIRUTHM -------------------------
def quick_sort(arr, L, R):
    if L < R :
        pi = partition(arr, L, R)
        quick_sort(arr, L, pi-1)
        quick_sort(arr, pi+1, R)
```

In [105]:
```
x = [38, 27, 43, 3, 9, 82, 10]
quick_sort(x, 0, len(x)-1)
```

In [106]:
```
x
```

Out[106]:
```
[3, 9, 10, 27, 38, 43, 82]
```

4) Fast Exponentiation:

In [1]:
```
def fast_expo(a,e):
    if e == 0:
        return 1
    elif e == 1:
        return a
    else:
        if e%2 == 0:
            x = fast_expo(a,e/2)
            return x**2
        else:
            x = fast_expo(a,(e-1)/2)
            return (x**2)*a

def fast_mod(a,e,n):
    x = fast_expo(a,e)
    return x%n
```

In [34]:
```
Fast_Mod = fast_mod(2, 20, 7)
```

In [36]:
```
Fast_Mod
```

Out[36]: 4

---

5) Segments and Pionts:

In [12]:
```
def point_segment():

    try:
        S_P = input('How many segmants and points to check: ')
        segment_point = tuple(map(int,S_P.rstrip().split(' ')))
        segments = []
        points = []
        for i in range(segment_point[0]):
            S = input(f'items of segment {i} = ')
            segments.append(tuple(map(int,S.rstrip().split(' '))))

        P = input(f'Points = ')
        points = list(map(int,P.rstrip().split(' ')))

        starts = []
        ends = []

        for segment in segments:
            starts.append(segment[0])
            ends.append(segment[1])

        # These must be sorted so we can cancel segments that wont by logic conta
        starts.sort()
        ends.sort()
        points.sort()

        SC = 0  # srt_counter
        EC = 0  # end_counter
        PSC = 0  # point_Segment_counter
        size = len(segments)
        result = dict()

        for point in points:
            while (SC < size) and (starts[SC] <= point):
                SC += 1
                PSC += 1  # This means that it is in fornt of segment that starts
            while (EC < size) and (ends[EC] < point):
                EC += 1
                PSC -= 1   # THis means that it's in front of segment ES
                # So for segmnt(SC, EC) it's not inside it and so not any other s
            # Add the No. of segments to that contain that point to the dictionar
            result[point] = PSC
        return result

    except:
        return 'INPUT ERROR'
```

In [9]:
```
x = point_segment()
```

```
How many segmants and points to check: 2 3
items of segment 0 = 0 5
items of segment 1 = 7 10
Points = 1 6 11
```

In [10]:
```
print(x)
```

```
{1: 1, 6: 0, 11: 0}
```

Majority Element:

In [37]:
```
def majority():
    n = input('How many elements: ')
    n = int(n)
    elements = input(f'Write elements: ')
    elements = list(map(int,elements.rstrip().split(' ')))

    if n != len(elements):
        return 'No. of elements is not  correct'

    result = dict()
    for i in range(n):
        if elements[i] in result:
            result[elements[i]] += 1
        else:
            result[elements[i]] = 1


    for key in result.keys():
        if result[key] > n/2:
            return f'(1) There is a majority number which is {key}, appeared {res
    return f'(0) There is no majority number'
```

In [40]:
```
major = majority()
```

```
How many elements: 5
Write elements: 2 3 9 2 2
```

In [41]:
```
print(major)
```

```
(1) There is a majority number which is 2, appeared 3 times
```

In [ ]:
```

```