Algorithms&Complexity - In Class Assignments (Week 2)

Divide and Conquer Algorithms

Manuela Cleves

1. Implement following Pseudo Code in the programming language of your choice (keep it OOP)

Input array A contains only integers from 1 to M

algorithm CountSort(A[1,...,n])

```
B[1…M] ⇐ [0,…,0] // Declare array with the length M

For j = 1 to n

    B[A[j]] ⇐ B[A[j]] + 1

For i = 2 to M

    B[i] ⇐ B[i] + B[i−1]

For j = n to 1

    A’[B[A[j]]] ⇐ A[j]

    B[A[j]] ⇐ B[A[j]] − 1
```

Find out the complexity of this Algorithm.

Visualization: https://visualgo.net/en/sorting?slide=15

Explanation for future reference:

1) Initialize an array B of length M to store the count of occurrences of each element.

2) Iterate through the input array A to count the occurrences of each element. Increment the corresponding count in B for each element encountered in A.

3) Modify the count array B to store the position of each element in the sorted order. Each element in B now represents the cumulative count of elements less than or equal to its index.

4) Iterate through the input array A in reverse order. Place each element at its correct sorted position in the new array A' based on the information stored in array B. Decrement the count in B for each element placed in A'.

In [9]:
```
def counting_sort(A, M):

    # Create array B of length M to store the counting (note: it does so for ea
    B = [0] * (M + 1)

    #Iterate through the input array A to count the ocurrences of each element
    for j in range(len(A)):
        B[A[j]] += 1

    # Modify count array so that B can store cumulative counts
    for i in range(2, M + 1):
        B[i] += B[i - 1]

    # Build array A_final to store positions in the final order (to not have ex
    A_final = [0] * len(A)
    for j in range(len(A) - 1, -1, -1):
        A_final[B[A[j]] - 1] = A[j]
        B[A[j]] -= 1

    return A_final
```

In [11]:
```
# Example usage
A = [7, 0, 8, 7, 8, 8, 5, 8, 5, 6]
M = max(A)
sorted_A = counting_sort(A, M)
print("Original Array:", A)
print("Sorted Array:", sorted_A)
```

```
Original Array: [7, 0, 8, 7, 8, 8, 5, 8, 5, 6]
Sorted Array: [0, 5, 6, 7, 7, 8, 8, 8, 8, 0]
```

1.1 The complexity of this algorithm:

O(n + M)

2. QuickSort algorithm presented in this lecture has $O(n$^2) worst-case time. Try to implement following pseudo code:

Hoare-partition (A,p,r)

x=A[p]

i=p-1

j=r+1

while true

    repeat

        j=j−1

```
until A[j]<=x

repeat

    i=i+1

until A[i]>=x

if i<j

    exchange  A[i] with A[j]

else return j
```

Demonstrate the operation of Hoare-partition on the Array [13,19,9,5,12,8,7,4,11,2,6,21] showing the values of the array and auxiliary values

In [3]:
```python
def hoare_partition(A, p, r):
    x = A[p]
    i = p - 1
    j = r + 1

    while True:
        while True:
            j -= 1
            if A[j] <= x:
                break

        while True:
            i += 1
            if A[i] >= x:
                break

        if i < j:
            # Swap A[i] and A[j]
            A[i], A[j] = A[j], A[i]
        else:
            return j

# Example usage
arr = [13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21]
piv = hoare_partition(arr, 0, len(arr) - 1)

def quicksort(arr):
    if len(arr) == 1 or len(arr) == 0:
        return arr
    else:
        pivot = arr[0]
        i = 0
        for j in range(len(arr)-1):
            if arr[j+1] < pivot:
                arr[j+1], arr[i+1] = arr[i+1], arr[j+1]
                i += 1
                print(arr)
        arr[0], arr[i] = arr[i], arr[0]
```

```
        first_part = quicksort(arr[:i])
        second_part = quicksort(arr[i+1:])
        first_part.append(arr[i])
        return first_part + second_part

alist = [54,26,93,17,77,31,44,55,20]

print(quicksort(alist))
```

```
[54, 26, 93, 17, 77, 31, 44, 55, 20]
[54, 26, 17, 93, 77, 31, 44, 55, 20]
[54, 26, 17, 31, 77, 93, 44, 55, 20]
[54, 26, 17, 31, 44, 93, 77, 55, 20]
[54, 26, 17, 31, 44, 20, 77, 55, 93]
[20, 17, 26, 31, 44]
[77, 55, 93]
[17, 20, 26, 31, 44, 54, 55, 77, 93]
```

OTHER CODE PROVIDED BY ILIA:

In [18]:

```
def karatsuba(x,y):
  print('Input', x, y)
  n = max(len(str(x)),len(str(y)))
  if n == 1:
    return x*y

  n2 = n // 2

  x1 = x // (10**n2)
  x2 = x % (10**n2)
  y1 = y // (10**n2)
  y2 = y % (10**n2)

  print('Halves', x1, x2, y1, y2)

  p1 = karatsuba(x1, y1)
  p2 = karatsuba(x2, y2)
  p3 = karatsuba(x1+x2, y1+y2) - p1 - p2

  print('Products', p1, p2, p3)

  return p1 * 10**(2*n2) + p3 * (10**n2) + p2

print(5126*128)
print(karatsuba(5126, 128))
```

```
656128
Input 5126 128
Halves 51 26 1 28
Input 51 1
Halves 5 1 0 1
Input 5 0
Input 1 1
Input 6 1
Products 0 1 5
Input 26 28
Halves 2 6 2 8
Input 2 2
Input 6 8
Input 8 10
Halves 0 8 1 0
Input 0 1
Input 8 0
Input 8 1
Products 0 0 8
Products 4 48 28
Input 77 29
Halves 7 7 2 9
Input 7 2
Input 7 9
Input 14 11
Halves 1 4 1 1
Input 1 1
Input 4 1
Input 5 2
Products 1 4 5
Products 14 63 77
Products 51 728 1454
656128
```

In [2]:
```python
def merge_lists(left_sublist,right_sublist):
  i,j = 0,0
  result = []

  while i<len(left_sublist) and j<len(right_sublist):
    if left_sublist[i] <= right_sublist[j]:
      result.append(left_sublist[i])
      i += 1
    else:
      result.append(right_sublist[j])
      j += 1
  result += left_sublist[i:]
  result += right_sublist[j:]
  return result
```

In [3]:
```python
def merge_sort(input_list):
  if len(input_list) <= 1:
    return input_list
  else:
    midpoint = int(len(input_list)/2)
    print('Midpoint', midpoint)
    print('Input left', input_list[:midpoint])
    left_sublist = merge_sort(input_list[:midpoint])
    print('Left output', left_sublist)
    print('Input right', input_list[midpoint:])
    right_sublist = merge_sort(input_list[midpoint:])
```

```
    print('Right output', right_sublist)
    return merge_lists(left_sublist,right_sublist)
```

In [4]:

```
# number_list = [3,1,5,3,2,5,8,2,9,6,12,53,75,22,83,123,12123]
number_list = [5, 2, 3, 9, 1]
merge_sort(number_list)

def quicksort(arr):
    if len(arr) == 1 or len(arr) == 0:
        return arr
    else:
        pivot = arr[0]
        i = 0
        for j in range(len(arr)-1):
            if arr[j+1] < pivot:
                arr[j+1], arr[i+1] = arr[i+1], arr[j+1]
                i += 1
                print(arr)
        arr[0], arr[i] = arr[i], arr[0]
        first_part = quicksort(arr[:i])
        second_part = quicksort(arr[i+1:])
        first_part.append(arr[i])
        return first_part + second_part

alist = [54,26,93,17,77,31,44,55,20]
print(quicksort(alist))
```

```
Midpoint 2
Input left [5, 2]
Midpoint 1
Input left [5]
Left output [5]
Input right [2]
Right output [2]
Left output [2, 5]
Input right [3, 9, 1]
Midpoint 1
Input left [3]
Left output [3]
Input right [9, 1]
Midpoint 1
Input left [9]
Left output [9]
Input right [1]
Right output [1]
Right output [1, 9]
Right output [1, 3, 9]
[54, 26, 93, 17, 77, 31, 44, 55, 20]
[54, 26, 17, 93, 77, 31, 44, 55, 20]
[54, 26, 17, 31, 77, 93, 44, 55, 20]
[54, 26, 17, 31, 44, 93, 77, 55, 20]
[54, 26, 17, 31, 44, 20, 77, 55, 93]
[20, 17, 26, 31, 44]
[77, 55, 93]
[17, 20, 26, 31, 44, 54, 55, 77, 93]
```