Raheel Ali (023-22-0024) DAA Assignment

Pancake Sort

Repeatedly flip the largest unsorted element to the front, then to its correct position at the end.

```
def flip(arr, i):
    arr[:i+1] = arr[:i+1][::-1]

def find_max_index(arr, n):
    return max(range(n), key=lambda i: arr[i])

def pancake_sort(arr): n=
    len(arr)
    for size in range(n, 1, -1):
        max_index = find_max_index(arr, size) if
        max_index != size - 1:
            flip(arr, max_index)
            flip(arr, size - 1)
```

Bucket Sort

Distribute elements into buckets, sort each bucket, then concatenat

```
def bucket_sort(arr): if
    len(arr) == 0:
    return arr
    bucket_count = 10
    max_val, min_val = max(arr), min(arr)
    bucket_range = (max_val - min_val) / bucket_count + 1
    buckets=[[] for _ in range(bucket_count)]
    for num in arr:
        index = int((num - min_val) // bucket_range)
        buckets[index].append(num)
    for bucket in buckets:
        bucket.sort()
    return [num for bucket in buckets for num in bucket]
```

Comb Sort

Improves bubble sort by comparing elements with a shrinking gap until fully sorted.

```
def comb_sort(arr):
 gap = len(arr)
shrink = 1.3
sorted= False
while not sorted:
        gap = int(gap / shrink)
        if gap <= 1:
            gap = 1
            sorted= True
        for i in range(len(arr) - gap):
        if arr[i]> arr[i + gap]:
                arr[i], arr[i + gap] = arr[i + gap], arr[i]
                sorted = False
```

Radix Sort

Sort numbers digit by digit using counting sort as subroutine.

```
 Def counting_sort(arr,exp):
    n=len(arr)
    output = [0] * n
    count = [0] * 10
    for i in arr:
        count[(i // exp) % 10] += 1 for i in
    range(1, 10):
        count[i] += count[i - 1]
         for i in reversed(arr):
        index = (i // exp) % 10
        output[count[index] - 1] = i count[index]=1
    for i in range(n):
        arr[i]=output[i]

def radix_sort(arr):
    max_num = max(arr)
    exp = 1
    while max_num // exp > 0:
        counting_sort(arr,exp)
        exp *= 10
```

Comparison Table

Algorithm	Best	Average	Worst	Space	Stable	Use Case
Selection	O(n^2)	O(n^2)	O(n^2)	O(1)	No	Small datasets, simplicity
Bubble	O(n)	O(n^2)	O(n^2)	O(1)	Yes	Educational, nearly sorted data
Insertion	O(n)	O(n^2)	O(n^2)	O(1)	Yes	Nearly sorted or small datasets
Merge	O(n log n)	O(n log n)	O(n log n)	O(n)	Yes	Large datasets, stable sorting needed
Quick	O(n log n)	O(n log n)	O(n^2)	O(log n)	No	Fast general-purpose sort (in-place)
Heap	O(n log n)	O(n log n)	O(n log n)	O(1)	No	Priority queues, large datasets
Count	O(n + k)	O(n + k)	O(n + k)	O(k)	Yes	Integers in known range
Pancake	O(n^2)	O(n^2)	O(n^2)	O(1)	No	Educational, theoretical interest
Bucket	O(n + k)	O(n + k)	O(n^2)	O(n + k)	Yes	Uniform float distribution
Comb	O(n log n)	O(n^2 / 2^k)	O(n^2)	O(1)	No	Faster Bubble Sort alternative
Radix	O(nk)	O(nk)	O(nk)	O(n + k)	Yes	Integers, fixed-length strings