**Sorting Algorithms**

1. **Time Stamp (**year, month, date, and time**):**

**Algorithm Description:**

The algorithm used would be **Bubble Sort,** for sorting time with respect to year/month/date/time. Bubble sort is an algorithm in which consecutive values/elements of the list are compared with each other, if the first element is greater than the second element they are swapped, otherwise, they remain as is. Then it moves to the next pair.

The bubble sort completes, once it passes the whole array without swapping elements.

According to the user’s requirement, the specific year/month/date/time, of the data on Twitter would be scraped, and sorted from oldest to latest. For instance, if the user chooses the year to be 2000, by applying bubble sort the data would be sorted from January to December.

**Pseudo-code of the algorithm:**

1. The timestamps of all data are stored in an array
2. Bubble sort will start with the first element (i=0 index) and check if it is sorted or not,(in case of the month it checks whether their order is correct or not/ for time/date/year it would check whether it is greater than (i+1)th element or not) and sort it.
3. If it is already sorted no swapping takes place
4. Now **i** gets incremented and move to the next element and sorts the next two elements as stated in 3.
5. Steps 3 and 4 are repeated till the second last element as the last element is already sorted.
6. Again i is set to 0 and the same steps are repeated to get the 2nd largest element at its respective place.

**Algorithm:**

bubble\_sort(array)

n=length of array

for i=0 to n-1

for j=0 to n-i-1

if array[j]>array[j+1]

swap array[j] with array[j+1]

**Code in Python:**

‘ def bubble\_Sort(a):

n = len(a)

for i in range(n-1):

for j in range(n-i-1):

if a[j]>a[j+1]:

a[j],a[j+1]=a[j+1],a[j]

**Time Complexity Analysis:**

* Comparing each element with the next one for an array of size n takes **n time.**
* The above step is repeated **n times.**
* The best time complexity: O(n^2)
* Average time complexity: O(n^2)
* Worst time complexity: O(n^2)

**Strengths:**

1. Easy to understand
2. Small code
3. Little memory overhead

**Weaknesses:**

1. Takes a lot of time

**Dry Run:**

****



1. **Categories:**

**Algorithm Description:**

The algorithm used would be **Select Sort,** for sorting numbers select sort selects the minimum value and from the un-sorted list and places it at the beginning. In the case of categories, whatever category name is stated by the user, would be selected from the list and brought to the beginning.

The user would enter the category they want, the text of the category would be used and tweets falling in that category would be selected and placed at the beginning

**Pseudo-code of the algorithm:**

1. Find the text same as category text and place it at the beginning

2. Repeat step 1 till all tweets are dealt with.

**Code in Python:**

‘ def selection\_sort(list,category):

n=len(list)

for i in range(n-1):

min\_i=i

for j in range(i+1,n):

if list[j]==Category:

min\_i=j

if min\_i!=i:

list[i],list[min\_i]=list[min\_i],list[i]

**Time Complexity Analysis:**

* There are 2 nested loops, and each is executed **n** times.
* The best time complexity: O(n^2)
* Average time complexity: O(n^2)
* Worst time complexity: O(n^2)

**Strengths:**

1. No overhead memory required
2. As we need specific categories, it will simply select the category name and place in the beginning
3. Performance is influenced by the initial ordering, so we’ll already use twitter’s categories to sort it w.r.t the text, further using sorting algorithm would further refine the search.

**Weaknesses:**

1. Less Efficient

**Dry Run:**

Let C be the category entered by the user

****



1. **Likes:**

**Algorithm Description:**

The algorithm used would be **Insertion Sort.** Virtually the list is split into two sub-arrays and values are taken from the unsorted array and placed in the right order in the sorted sub-array.

In twitter squeaker, the user is provided with an option to sort tweets from maximum to a minimum number of likes or from minimum to a maximum number of likes. So to address both conditions we would just change the code at one position, which would be addressed in the following pseudocode section.

**Pseudo-code of the algorithm:**

1. Begin from the 2nd element and iterate till the last element
2. Compare the present number with the 1st one(or the previous one)
3. If the element in consideration i.e key element is smaller(min-max)/larger(max-min), it is compared with all the previous elements and placed at the correct position.
4. The other elements are pushed ahead to make space for the key element

**Code in Python:**

‘ def insertion\_sort(arr):

n=len(arr)

for i in range(1,n):

key = arr[i]

j=i-1 if (max-min)

while j>=0 and arr[j]>key: while j>=0 and arra[j]<key

arr[j+1]=arr[j]

j=j-1

arr[j+1]=key

**Time Complexity Analysis:**

* If the list is already sorted in the desired order, only the first loop runs n-1 times
* The best time complexity: O(n)
* There are 2 nested loops, and each is executed **n** times.
* Average time complexity: O(n^2)
* Worst time complexity: O(n^2)

**Strengths:**

1. Simple Code
2. Minimum Space required

**Weaknesses:**

1. Does not perform as good as other algorithms
2. Does not work well with huge amount of lists

**Dry Run:**

For min-max number of likes

****

1. **Re-tweets:**

**Algorithm Description:**

The algorithm used would be **Quick Sort.** It is a divide and conquer algorithm. It takes an element as a pivot, in our algorithm we’ll use the last element as a pivot. We use a function partition(), and let’s say our pivot is x, we’ll place x at its correct position and with smaller elements behind and greater elements ahead of x.

In accordance with the user’s requirement, we’ll sort tweets in either order of max-min or min-max re-tweets.

**Pseudo-code of the algorithm:**

1. Using partition function we split the array into two halves with our pivot which is the last element of the list at its correct position
2. Recursively we’ll apply the quick sort function on the split arrays the same way as done before.

**Code in Python:**

‘ def quickSort(arr,low,high):

if low<high:

pi= partition(arr,low,high)

quickSort(arr,low,pi-1)

quickSort(arr,pi+1,high)

def partition(arr,low,high):

pivot=arr[high]

i=low-1

for j in range(low,high):

if arr[j]<=pivot:

i=i+1

arr[i],arr[j]=arr[j],arr[i]

arr[i+1],arr[high]=arr[high],arr[i+1]

return i+1

**Time Complexity Analysis:**

* Partitioning elements takes n time
* The problem is divided by factor 2
* The best time complexity: O(nlogn)
* Average time complexity: O(nlogn)
* When an array is sorted it will give us the worst time complexity
* Worst time complexity: O(n^2)

**Strengths:**

1. Can easily deal with a huge list
2. No overhead storage is required

**Weaknesses:**

1. If the list is already sorted it gives the worst case

**Dry Run:**

For min-max number of retweets. ****

1. **Multi-level Sorting:**

**Algorithm Description:**

The algorithm used would be **Merge Sort.** It is also a divide and conquer algorithm. It divides the array into sub-arrays in every iteration into two equal sub-arrays and sorts them by calling itself recursively. In the end, it merges the sorted halves.

**Pseudo-code of the algorithm:**

1. **Divide:** Finds the midpoint position and divides the aray
2. **Conquer:** Recursively it sorts the sub-arrays, if q is the midpoint then it recursively calls itself with array(....q) and (q+1…)
3. **Combine:** The sorted halves are combined

**Code in Python:**

‘ def merge\_sort(list):

if len(list)>1:

mid=len(list)//2

left = list[:mid]

right=list[mid:]

merge\_sort(left)

merge\_sort(right)

i=0

j=0

k=0

while i<len(left) and j<len(right):

if left[i]<=right[j]:

list[k]=left[i]

i+=1

else:

list[k]=right[j]

j+=1

k+=1

while i<len(left):

list[k]=left[i]

i+=1

k+=1

while j<len(right):

list[k]=right[j]

j+=1

k+=1

**Time Complexity Analysis:**

* The best time complexity: O(nlogn)
* Average time complexity: O(nlogn)
* Worst time complexity: O(nlogn)

**Strengths:**

1. Can sort of files of any size

**Weaknesses:**

1. Requires extra space

**Dry Run:**

****