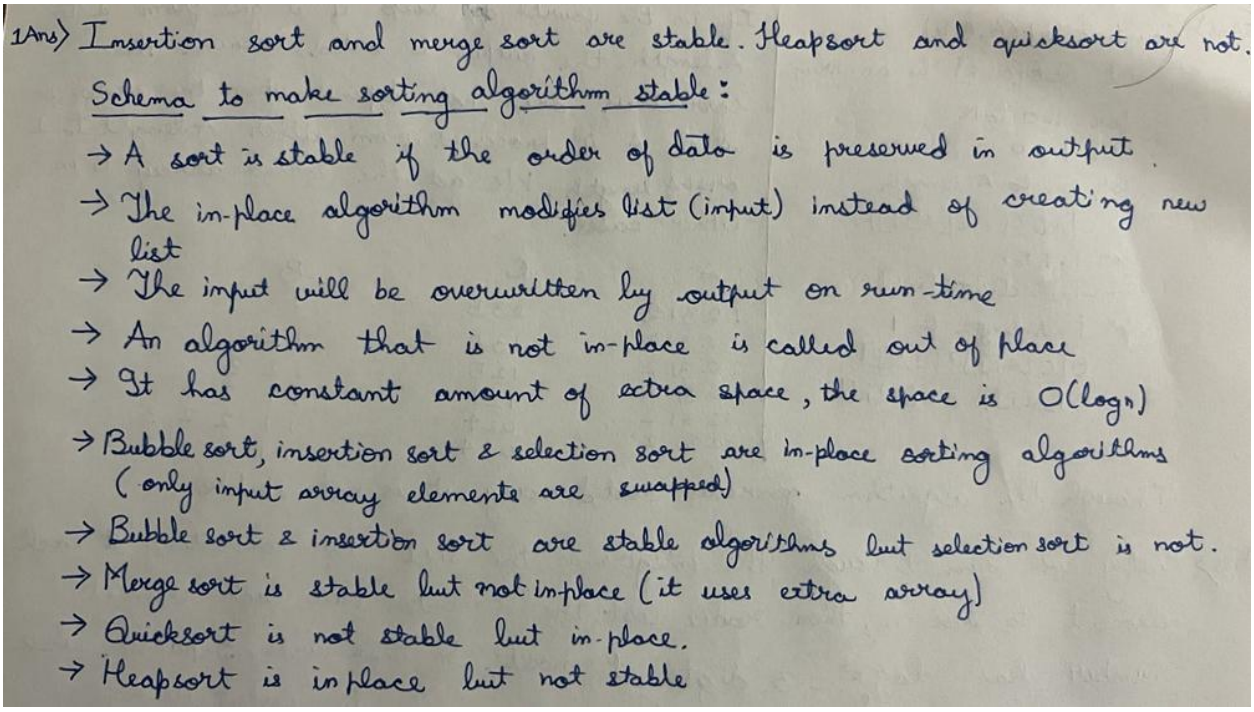
CSCE 5150 – Analysis of Computer Algorithms

Homework No. 4 – Sorting in Linear Time

Neha Goud Baddam
11519516

Q1. [5 points] A sorting algorithm is said to be stable if numbers with the same value appear in the output array in the same order as they do in the input array. Which of the following sorting algorithms are stable: insertion sort, merge sort, heapsort, and
quicksort? Give a simple scheme that makes any sorting algorithm stable. (Ex. 8.3-2)



1Ans) Insertion sort and merge sort are stable. Heapsort and quicksort are not.

Schema to make sorting algorithm stable:

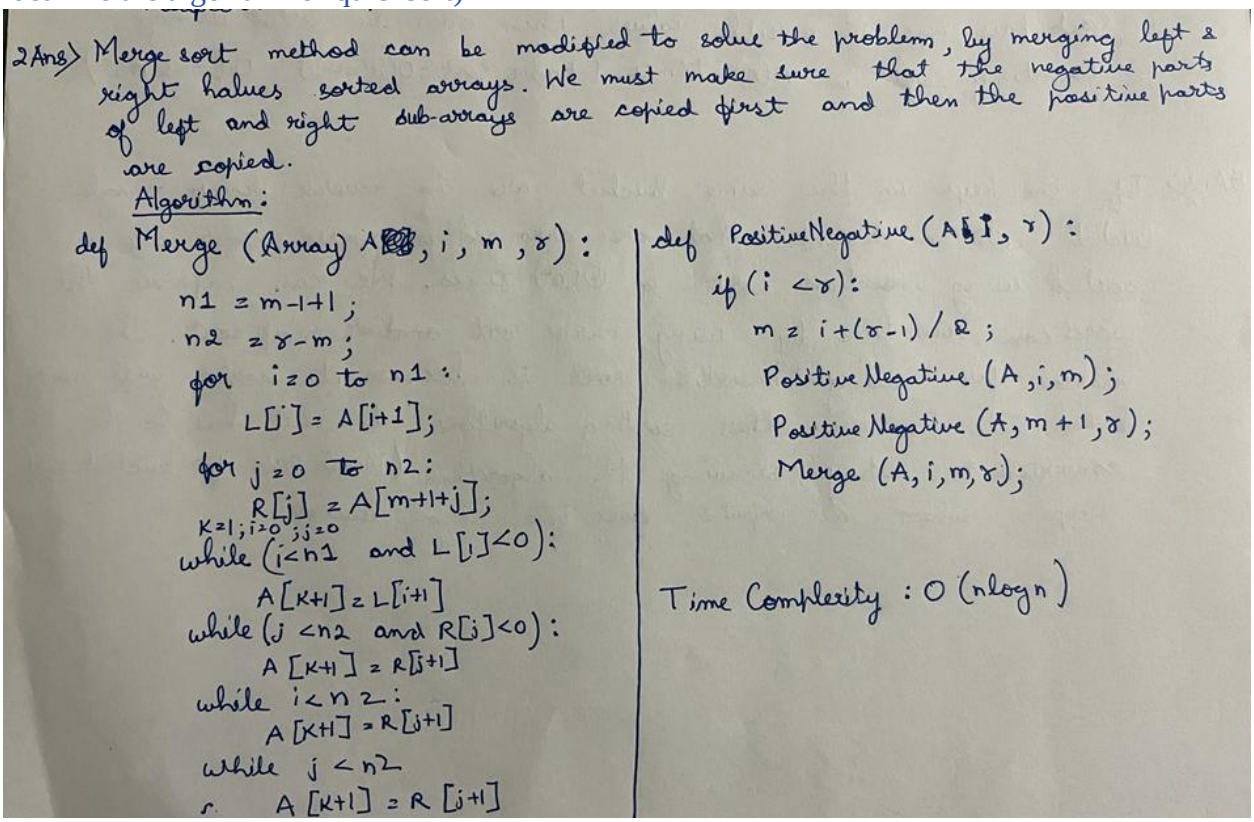
→ A sort is stable if the order of data is preserved in output.
→ The in-place algorithm modifies list (input) instead of creating new list
→ The input will be overwritten by output on run-time
→ An algorithm that is not in-place is called out of place
→ It has constant amount of extra space, the space is O(log n)
→ Bubble sort, insertion sort & selection sort are in-place sorting algorithms (only input array elements are swapped)
→ Bubble sort & insertion sort are stable algorithms but selection sort is not.
→ Merge sort is stable but not in-place (it uses extra array)
→ Quicksort is not stable but in-place.
→ Heapsort is in place but not stable

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 4 – Sorting in Linear Time

Neha Goud Baddam
11519516

Q2. [5 points] Design an algorithm to rearrange elements of a given array of n real numbers so that all its negative elements precede all its positive elements. (Hint: You can use the partition idea like the algorithm of quicksort)



2 Ans) Merge sort method can be modified to solve the problem, by merging left & right halves sorted arrays. We must make sure that the negative parts of left and right sub-arrays are copied first and then the positive parts are copied.

Algorithm:

```
def Merge (Array) A, i, m, r):
    n1 = m-i+1;
    n2 = r-m;
    for i=0 to n1:
        L[i] = A[i+1];
    for j=0 to n2:
        R[j] = A[m+1+j];
    K=1; i=0; j=0
    while (i<n1 and L[i]<0):
        A[K+1] = L[i+1]
    while (j<n2 and R[i]<0):
        A[K+1] = R[j+1]
    while i<n2:
        A[K+1] = R[j+1]
    while j<n2
        A[K+1] = R[j+1]
```

```
def PositiveNegative (A, i, r):
    if (i < r):
        m = i+(r-1)/2;
        PositiveNegative (A, i, m);
        PositiveNegative (A, m+1, r);
        Merge (A, i, m, r);
```

Time Complexity : O(nlogn)

Q3. [5 points] Suppose that we were to rewrite the last for loop header in the Counting sort algorithm as

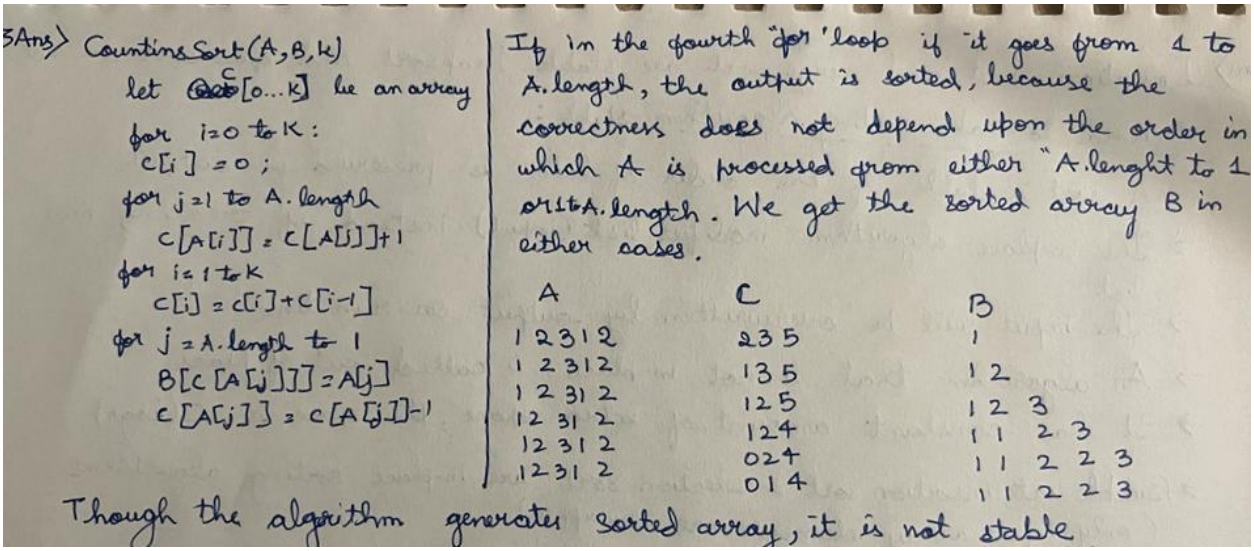
for j = 1 to n

Show that the algorithm still works properly. Is the modified algorithm still stable?

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 4 – Sorting in Linear Time

Neha Goud Baddam
11519516



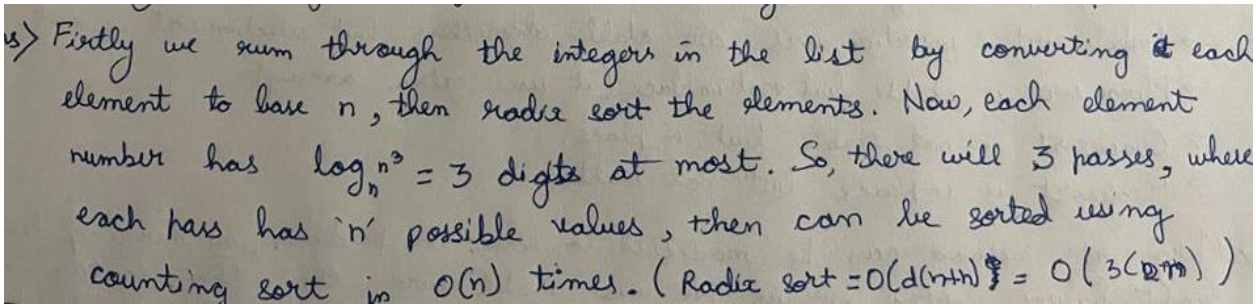
3Ans) CountingSort(A,B,k)
let C[0...k] be an array
for i=0 to K:
C[i] = 0;
for j=1 to A.length
C[A[j]] = C[A[j]]+1
for i=1 to K
C[i] = C[i]+C[i-1]
for j = A.length to 1
B[C[A[j]]] = A[j]
C[A[j]] = C[A[j]]-1

If in the fourth "for" loop if it goes from 1 to A.length, the output is sorted, because the correctness does not depend upon the order in which A is processed from either "A.lenght to 1 or 1 to A.length. We get the sorted array B in either cases.

A	C	B
1 2 3 1 2	2 3 5	1
1 2 3 1 2	1 3 5	1 2
1 2 3 1 2	1 2 5	1 2 3
1 2 3 1 2	1 2 4	1 1 2 3
1 2 3 1 2	0 2 4	1 1 2 2 3
1 2 3 1 2	0 1 4	1 1 2 2 3

Though the algorithm generates sorted array, it is not stable

Q4. [5 points] Show how to sort n integers in the range 0 to n3-1 in O(n) time. (Hint. Try to put a method to use radix sort and then prove its time complexity). (Ex. 8.3-4)



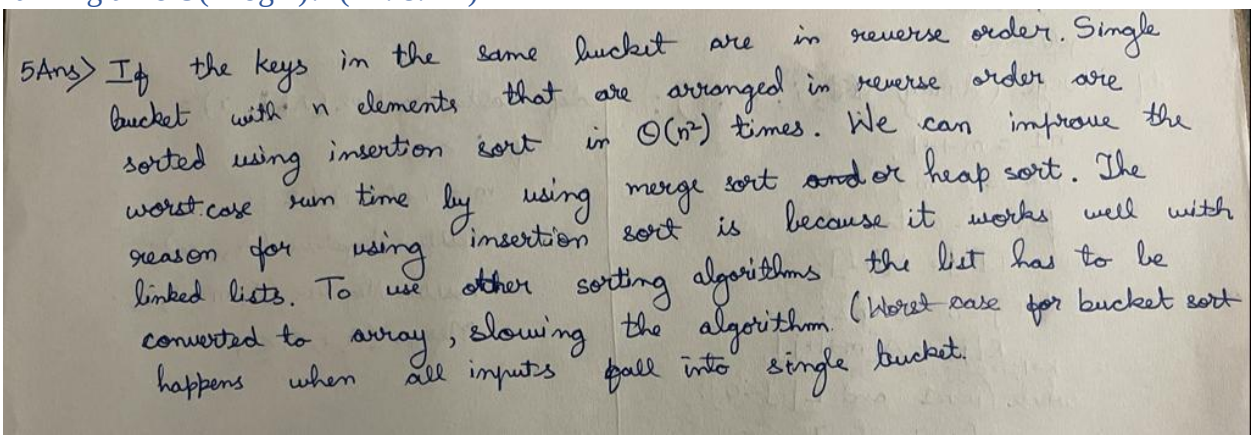
s) Firstly we sum through the integers in the list by converting it each element to base n, then radix sort the elements. Now, each element number has $\log_n n^3 = 3$ digits at most. So, there will 3 passes, where each pass has 'n' possible values, then can be sorted using counting sort in O(n) times. (Radix sort = $O(d(n+n)) = O(3(2n))$)

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 4 – Sorting in Linear Time

Neha Goud Baddam
11519516

Q5. [5 points] Explain why the worst-case running time for bucket sort is Θ(n2). What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time O(n log n)? (Ex. 8.4-2)



5Ans> If the keys in the same bucket are in reverse order. Single bucket with n elements that are arranged in reverse order are sorted using insertion sort in $\Theta(n^2)$ times. We can improve the worst-case run time by using merge sort ~~and~~ or heap sort. The reason for using insertion sort is because it works well with linked lists. To use other sorting algorithms the list has to be converted to array, slowing the algorithm. (Worst case for bucket sort happens when all inputs fall into single bucket.