# CS206M: DATA STRUCTURE AND ALGORITHM

## CLASS NOTES: 6TH FEB

**Problem Statement** : How to stabilize Selection Sort.

**Difficulty:**

Case 1: If we swap the duplicates during comparison.

Counter Example: consider 5, $2_a$, $2_b$

Final Result: $2_b$, $2_a$, 5 → original order is not maintained(unstable).

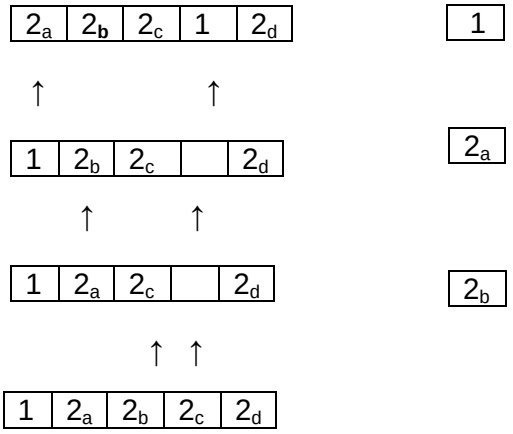
Case 2: If we don't swap the duplicates during comparison.

Counter Example: consider $2_a$, $2_b$, 1

Final Result: 1,$2_b$, $2_a$ → original order is not maintained(unstable).

**Solution:** Assume that our selection sort algorithm don't swap the duplicates. Whenever we are swapping two distinct numbers, replace left element with right and then move the duplicates of left element (including left element) to its next duplicate position present between the numbers and place the last duplicate in right element position.

Example: consider a series $2_a$, $2_b$, $2_c$, 1,$2_d$




Time complexity is T(n)= T(n-1) +O(n) →no change in asymptotic complexity

Home work: How to Stabilize any sorting algorithm

**Streaming:** Getting access to small part of output while the program is running.

Some of the Sorting algorithms with which streaming is possible are:

1) Selection sort- we will get the lower elements while the program is running.
2) Bubble sort – we will get the higher elements while the program is running
3) Heap sort- we will get least element while the program is running

With insertion sort ,merge sort & quick sort streaming is not possible**.**

**Parallelization:** Doing different tasks at same time. It reduces running time of an algorithm.

Parallelization is possible with merge sort.

**Merge sort:** we can stabilize merge sort by giving high preference (in case of duplicates) to the element in the left part.

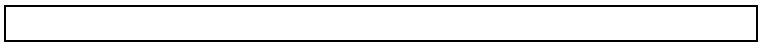
```
Mergesort(Arr,left,right)
{
        If(right-left >1)
        {
                Mergesort(Arr,left, floor( (left+right)/2))
                Mergesort(Arr, floor( (left+right)/2)+1,right)
                Merge(Arr,left,right)
        }
        else
                return;
}
```

Merge function using extra amount of space:

Program logic:


We have to merge these two sorted arrays so that the resulting is also a sorted array.

Consider an array whose size is equal to size (right-left+1) .



If left is lesser than right then insert the left element into new array else insert right. If left is equal to right then give preference to left side element .If the pointer on left side is array reaches its last element insert the remaining elements of right sided array in new array. Store these elements in old array (Arr) starting with index left and ending at index right.

Time complexity for merge function is O(n).

Total time complexity of merge sort T(n)=2T(n/2) + O(n)

Total amount of extra space using: $\log_2(n)+n$.

$\log_2(n)$ → to store recursion stack.

n→ for creating new array.

Home work: Can we solve it without using extra space.

**Merge sort using Linked list:**

Merge sort function: It is same as the merge sort function in arrays. But the time complexity to find the middle element is of O(n)

Merge function: It will take linear time complexity to sort two sorted linked lists. Here the time required to insert and delete an element in the linked list is constant .

Time complexity T(n) = 2T(n/2) + O(n) + O(n) →still O(nlogn)

Here also we are using extra amount of space for storing pointers.

Created by

k. siva teja

11010283