Data Structures Lab
Session 06

Course: Data Structures (CS2001) **Semester: Fall** 2022
Instructor: Mafaza Mohi **T.A:** N/A

Note:
- Lab manual cover following below elementary and advanced sorting algorithms
  {Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Radix Sort}
- Maintain discipline during the lab.
- Just raise your hand if you have any problems.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

Bubble Sort:

Bubble Sort, the two successive strings arr[i] and arr[i+1] are exchanged whenever arr[i]> arr[i+1]. The larger values sink to the bottom and are hence called sinking sort. At the end of each pass, smaller values gradually "bubble" their way upward to the top and hence called bubble sort.

Flag-based Bubble Sort:

An improvement of bubble sort is obtained by adding a flag to discontinue processing after a pass in which no swap was performed.

- **Examples:** If the array is [10, 12, 20, 25, 30, 31, 32, 33, 35, 60, 50]
- Sort the array using Bubble Sort & Modified Bubble Sort.
- Provide the number of Swaps for both sorting techniques.

Insertion Sort:

Insertion sort is an intelligent sorting algorithm, as it only swaps the data if it is not already sorted and works really well when we have partially sorted data. Add the functionality of insertion sort in your DSA class.

//Pseduocode:

```
void insertionSort (int *array, int size) {
    Choose the second element in the array and place it in order with respect to the first element.

    Choose the third element in the array and place it in order with respect to the first two elements.

    Continue this process until done.

    Insertion of an element among those previously considered consists of moving larger elements one position to the right and then inserting the element into the vacated position
}
```

Selection Sort:

Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

Working of Selection Sort:

1. Select the first element as a minimum
2. Compare the minimum with the second element. If the second element is smaller than the minimum, assign the second element as the minimum.
3. Compare the minimum with the third element. Again, if the third element is smaller, then assign a minimum to the third element otherwise do nothing. The process goes on until the last element.
4. Swap the first with a minimum.
5. For each iteration, indexing starts from the first unsorted element. Steps 1 to 3 are repeated until all the elements are placed in their correct positions.

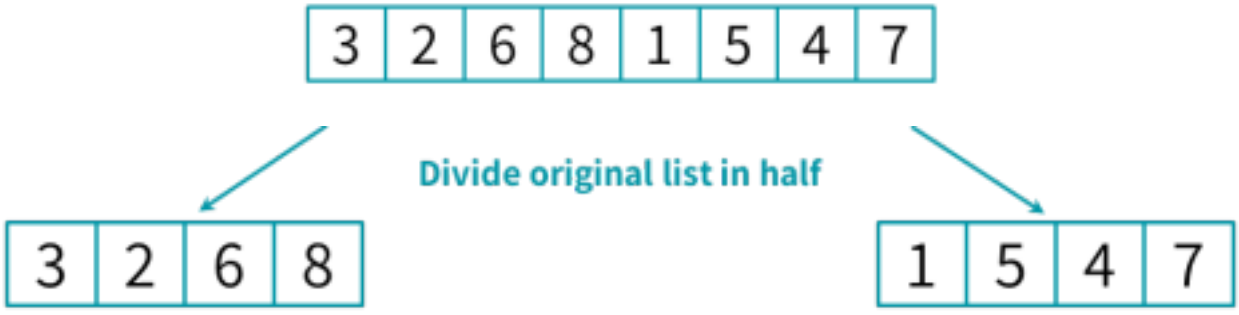
Task 01:

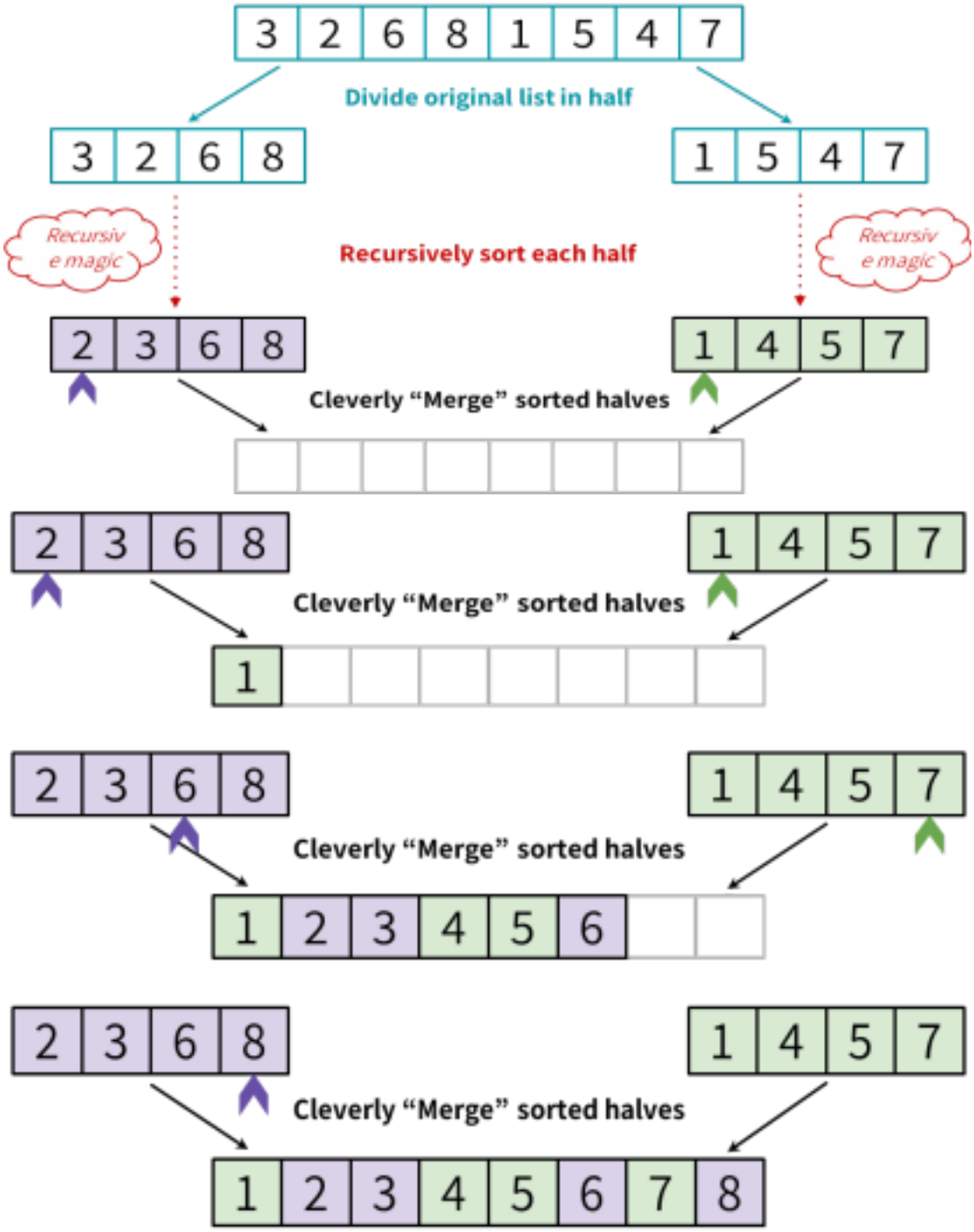
Augment your Dynamic Safe Array class to have a Merge Sort function which at any stage can be called to sort the DSA using merge sort.

- Count the number of comparisons performed by the sorting algorithm.
- Count the number of swaps performed by the sorting algorithm.
- Measure the time taken by the merge sort in seconds.
- Provide the values of sorting on DSA of length: 1000, 10000, and 100000. (All values will be randomly generated).

Key Points:

Using the *Divide and Conquer* technique, we divide a problem into subproblems. When the solution to each subproblem is ready, we 'combine' the results from the subproblems to solve the main problem.





Pseudocode for Merge Sort

```
MergeSort (data [ ], first, second )
        if first < last
                mid = ( first + last ) / 2 ;
                MergeSort ( data, first, mid);
```

MergeSort (data, mid+1, last);
Merge (data, first, last);

Pseudocode for Merge Operation

```
Merge (array1 [ ], first, second )
        mid = ( first + second ) / 2;
        i1 = 0; i 2 = first, i3 = mid + 1;
        while both left and right subarrays of array1 contain elements
                    if array1 [ i2 ] < array1 [ i3 ]
                            temp [ i1++ ] = array1 [ i2++ ] ;
                    else temp [ i1++ ] = array1 [ i3++ ] ;
              load into temp the remaining elements of array1;
              load to array1 the contents of temp;
```

Task 02:

Quicksort is another sorting algorithm, which implies divide and conquer strategy to solve the problem. Integrate Quick Sort in your DSA class as well with the additional following features.

- Count the number of comparisons performed by the sorting algorithm.
- Count the number of swaps performed by the sorting algorithm.
- Measure the time taken by the merge sort in seconds.
- Provide the values of sorting on DSA of length: 1000, 10000, and 100000. (All values will be randomly generated).

Pseudocode for Quick Sort

```
QuickSort(array [ ] )
        if length ( array ) > 1
                choose bound;
                while there are elements left in the array
  include element either in subarray1 if el ≤ bound
                        or in subarray2 if el ≥ bound;
                QuickSort(subarray1);
                QuickSort(subarray2);
```

Task 03:

Radix sort is a non-comparison based sorting algorithm and provide the best computational time to sort the data in linear time. You are required to write the method for radix sort in your DSA class. ● Measure the time taken by the merge sort in seconds.

- Provide the values of sorting on DSA of length: 1000, 10000, and 100000. (All values will be randomly generated).
- In your .cpp file, please mention the limitation of radix sort (as a multiline comment) which you observed during the implementation of the algorithm.

Pseudocode for Radix Sort

```
RadixSort ( )
        for d = 1 to the position of the leftmost digit of longest number
                distribute all numbers among piles 0 through 9 according to the dth
                digit put all integers on one list.
```

Task 04:

From labs 06 and 07 extract all of your sorting techniques in one .cpp file, use this file along with DSA to provide the same functionality. In addition,

- o Augment the insertion and deletion methods of your DSA class in such a way that your data is always in sorted order.
- o Please utilize the calculations you have performed for each sorting algorithm in identifying the best sorting algorithm for the following data:
  - ▪ Sorting an Array of Integers with the length of 10000.
  - ▪ Sorting an array of objects with a length of 1000

Task 05:

Integrate your sorting class with the DSA-2D class to have the following functionality ● Soring the names of the Users.

- ● Measure the time for 2D sorting as well.