Data Structures 1st Assignment Report

Name:	ID:
Armia Joseph Hakim	21010229
Amin Mohamed Amin	21010310
Andrew Safwat Fawzy	21010314
Habiba Tarek Ramadan Ali	21010445
Rafy Hany Saeed	21010504

Below is a comparison table of three sorting algorithms, focusing on their respective complexities. This comparison aims to illustrate the differences in performance characteristics among the algorithms.

		Radix	**Merge**	**Insertion**
AvgTime:	ArrSize			
	900	1ms	4ms	1ms
	1800	0ms	6ms	3ms
	2700	0ms	5ms	4ms
	3600	1ms	13ms	7ms
	4500	1ms	13ms	12ms
	5400	1ms	27ms	26ms
	6300	1ms	25ms	28ms
	7200	1ms	33ms	33ms
	8100	1ms	35ms	31ms
	9000	1ms	43ms	42ms
	9900	2ms	56ms	53ms
	10800	1ms	82ms	69ms
	11700	1ms	97ms	91ms
	12600	2ms	88ms	73ms
	13500	2ms	69ms	86ms
	14400	2ms	121ms	120ms
	15300	2ms	244ms	121ms
	16200	2ms	298ms	214ms
	17100	2ms	263ms	274ms
	18000	3ms	280ms	278ms
	18900	3ms	334ms	321ms
	19800	2ms	393ms	361ms
	20700	2ms	380ms	411ms
	21600	2ms	420ms	479ms
	22500	3ms	374ms	400ms
	23400	3ms	450ms	458ms
	24300	3ms	522ms	488ms
	25200	3ms	570ms	576ms
	26100	3ms	721ms	754ms
	27000	3ms	726ms	720ms

	27900	2ms	822ms	747ms
	28800	3ms	1051ms	940ms
	29700	4ms	989ms	899ms
	30600	3ms	1040ms	983ms
	31500	3ms	1135ms	1044ms
Time Complexity		O(n)	O(n log n)	$O(n^2)$
Space Complexity		O(n)	O(n)	O(1)

Radix Sort Algorithm:

Radix sort is a **non-comparative sorting algorithm** that sorts numbers by processing individual digits. It operates by distributing numbers into a set of buckets based on each digit's value, then recursively applying the same process to the numbers within each bucket.

Here's a more detailed explanation of the algorithm along with analysis of its time and space complexities and best, worst, and average-case scenarios.

Radix Sort Algorithm :

1. Get the maximum digits count of the largest number: Find the maximum number of digits in the input array.
2. Loop through each digit from least significant to most significant (rightmost to leftmost):

   2.1. Create 10 buckets (0 to 9) for each digit and 20 buckets in case there is negative numbers (-9 to 9)

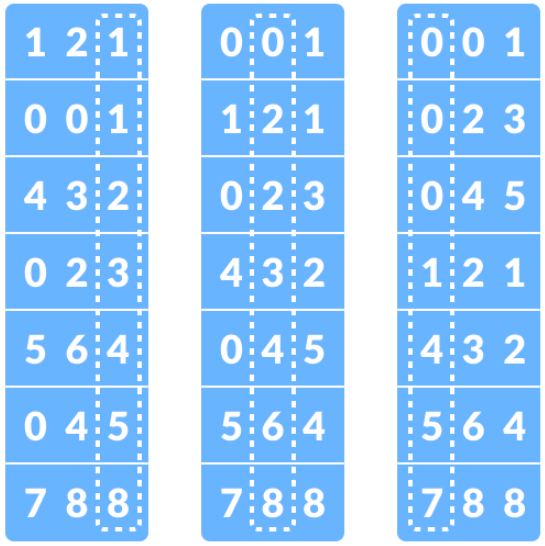
   2.2. Distribute numbers into buckets based on the current digit's value.

   2.3. Collapse the numbers in each bucket into a flat array.

   2.4. Update the original array with the new order.

3. Repeat the above step for each digit position.
4. Return the sorted array.

Data Structures 1st Assignment Report



sorting the integers according to units, tens and hundreds place digits

Radix Sort Helper Functions:

getMax(arr, n): This function iterates through the array arr of length n to find the maximum number present. It initialises maxNumber with the first element of the array and then compares it with subsequent elements to update maxNumber if a larger element is found. The function returns the maximum number found in the array.

getMaxNumberOfDigits(maxNumber): Given the maximum number maxNumber, this function calculates and returns the number of digits in that number. It repeatedly divides maxNumber by 10 until it becomes zero, incrementing a counter for each division operation. The final count represents the number of digits in maxNumber.

getDigit(number, digit): This function retrieves the digit at a specified position digit within a given number. It first divides the number by 10 raised to the power of digit - 1 to shift the target digit to the unit's place. Then, it takes the modulo 10 of the result to extract the desired digit.

Time Complexity Analysis:

let n : the number of elements in the input array.

let k : the number of digits in the maximum number in the array.

let b : the base of the numbers, which is typically 10 in the case of decimal numbers.

1. Iteration through each digit position (from least significant to most significant): This step involves k iterations.
2. Distribution of elements into buckets: In each iteration, we process each element once, distributing them into buckets based on their digits. This operation takes **O(n)** time.
3. Total time complexity is **O(k (n + b))**

Space Complexity Analysis:

1. Space for the input array: **O(n)**
2. Space for the buckets: Since we create 10 buckets for each digit (0-9), the space required for buckets is proportional to the number of digits in the maximum number. Therefore, the space complexity for the buckets is **O(b)**.
3. Total space complexity is **O(n + b)**

Worst case time complexity : The worst case in radix sort occurs when all elements have the same number of digits except one element which has a significantly large number of digits. If the number of digits in the largest element is equal to n, then the runtime becomes **O(n²)**

Best case time complexity : The best case in radix sort occurs when all elements have the same number of digits. then the runtime becomes **O(k * n)**

Advantages of Radix Sort Algorithm:

1. Fast when the keys are short, i.e. when the array element range is small.
2. Radix Sort is a stable sort because it maintains the relative order of elements with equal values.

Disadvantages of Radix Sort Algorithm:

1. Radix Sort becomes less efficient when the range of the input data exceeds the number of data points.
2. Can’t Handle the case that we have floating point numbers.

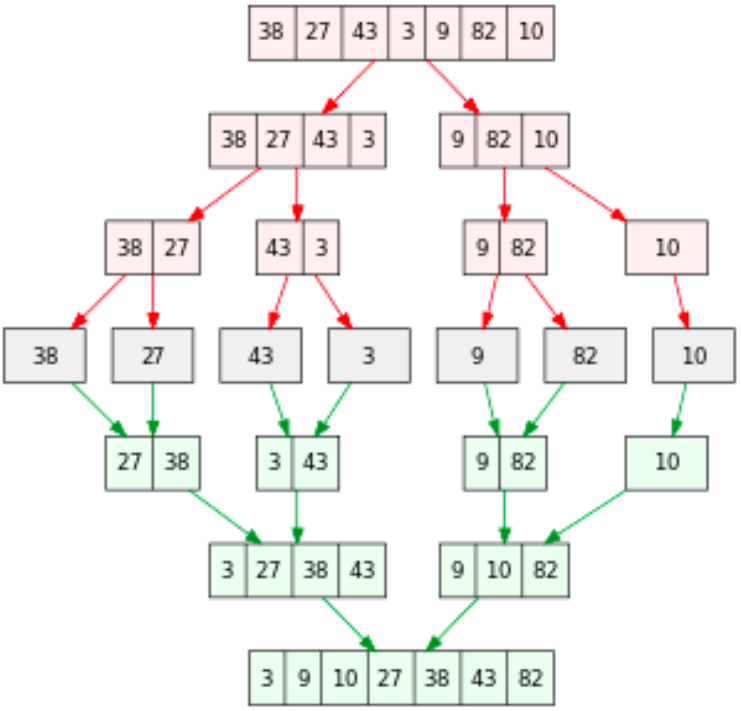
Data Structures 1st Assignment Report

Merge Sort Algorithm:

Merge sort is a recursive and comparative-based sorting algorithm, and classified as a divide-and-conquer algorithm.

Merge Sort Algorithm:

1. **Divide** the unsorted list into *n* sublists, each containing one element (a list of one element is considered sorted).
2. Repeatedly **merge** sublists to produce new sorted sublists until there is only one sublist remaining. This will be the sorted list.



Used Functions:

1. mergeSort function , performs the following two steps:

   Divide: The original array is divided into two halves.
   Conquer: Each half is recursively sorted using merge sort.

2. merge function , performs the merge step
   Combine: The sorted halves are merged to produce a single sorted array.

//Implementation did not involve any additional functions.

Data Structures 1st Assignment Report

Time Complexity Analysis:

let n : the number of elements in the input array.

1. **Divide Step**:
    a. During the divide step, the array is recursively divided into smaller subarrays until each subarray contains only one element. This process takes **O(log n)** time because the array is repeatedly divided in half.
2. **Merge Step**:
    a. During the merge step, the sorted subarrays are merged to produce a single sorted array. This process takes **O(n)** time because each element in the array is visited once during the merging process.

Since the merge step occurs after all the dividing is done, the overall time complexity of merge sort is **O(n log n)**.

Space Complexity Analysis:

Auxiliary Space:

- Merge sort requires additional space to merge two sorted subarrays into a single sorted array. This space is proportional to the size of the input array.

So the space complexity of the merge sort is **O(n).**

Worst case time complexity & best case time complexity :

- Both are O(n log n) as it always performs the same steps even if the array is sorted.

Advantages of Merge Sort Algorithm:

1. Consistent Performance: Merge sort has a consistent time complexity of O(n log n) for all cases (average, worst, and best). This makes it highly predictable and reliable, regardless of the input data distribution.
2. No Worst-Case Scenarios: Unlike other sorting algorithms like quicksort, merge sort does not have worst-case scenarios.

Disadvantages of Merge Sort Algorithm:

1. Not in place , required additional space
2. Recursive Overhead.

Data Structures 1st Assignment Report

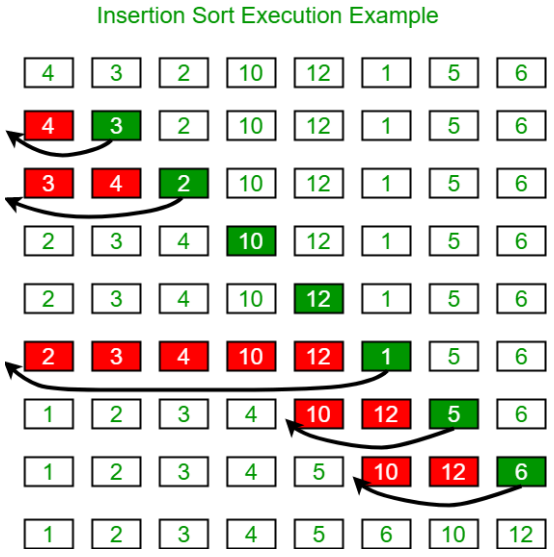
Insertion Sort Algorithm:

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time by comparisons.

Insertion sort algorithm:

1. **Initialization**: Start with the assumption that the first element in the array is already sorted. So, consider the first element as a sorted sublist of one element.

2. **Iterate through the unsorted portion**: Begin with the second element and iterate through the array, considering each element one by one.
3. **Insertion**: For each element in the unsorted portion, compare it with the elements in the sorted sublist (which starts with only the first element initially). Move elements in the sorted sublist that are greater than the current element to the right, creating space for the current element to be inserted.
4. **Insert the element**: Once the correct position for the current element in the sorted sublist is found (i.e., all elements to the left are smaller and all elements to the right are greater), insert the current element into that position.
5. **Repeat**: Repeat steps 2-4 until all elements are sorted. This involves iterating through the unsorted portion of the array and inserting each element into its correct position in the sorted sublist.
6. **End**: Once all elements are sorted, the entire array is now sorted.



Data Structures 1st Assignment Report

Time Complexity Analysis:

let n : the number of elements in the input array.

1. **Iterating through the array**:

   Starting from second element (assuming that first element is sorted), for each element i in the array, it gets compared with the previous element and if element i is greater, iteration continue, iterating through the arrays is **O(n)**.

2. **Sorting:**

   If element i is less than the previous element, the program **loops** backward till it finds the first element less than element i and **inserts** element i after it (before the first greater element), for each element unsorted, it takes **O(n)**.

The overall time complexity of insertion sort is **O(n*n) = O(n^2)**.

Space Complexity Analysis:

Auxiliary Space:

- Insertion sort doesn't require any additional spaces to perform its sorting as it works on the same array.

So the space complexity of the insertion sort is **O(1).**

Worst case time complexity:

- **O(n^2)** if the array is reverse sorted.

Best case time complexity:

- **O(n)** if the array is sorted.

Advantages of Insertion Sort Algorithm:

1. The main advantage of the insertion sort is its simplicity.
2. In-place sorting algorithm so the space requirement is minimal.

Disadvantages of Insertion Sort Algorithm:

1. With n-squared steps required for every n element to be sorted, the insertion sort does not deal well with a huge list.