Rizwan Ali - 14789 - BSCS - 3D

Teacher Sir Jamal Abdul Ahad

Assignment : 01

Subject : DSA

The assignment includes all the exercise questions and solutions for DSA exercises 1, 2, and 3.

Github Repository :

https://github.com/MarkhorDB/DSA-Solutions

Q. : Describe your own real-world example that requires sorting. Describe one that requires finding the shortest distance between two points.

Answer :

Consider a delivery driver who needs to drop off packages at different locations in a neighborhood. To minimize travel time, the driver must find the shortest route that connects all the delivery points. This involves calculating the shortest distance between the starting point and each delivery location.

Q. : Other than speed, what other measures of efûciency might you need to consider in a real-world setting?

Answer :

In a real-world setting, there are several measures of efficiency to consider beyond just speed. Cost is a crucial factor, as it reflects the financial resources required for a project or process. Resource utilization assesses how effectively materials, labor, and technology are employed, ensuring that nothing is wasted. Energy consumption is also important, as minimizing energy use can lead to both cost savings and environmental benefits. Quality must be maintained to ensure that the output meets standards, reducing errors and defects. Additionally, scalability is essential for evaluating how well a process can handle increased demand. Time to market is another key measure, indicating how quickly a product or service can be delivered to consumers. Finally, flexibility is vital for adapting to changing requirements, while sustainability measures the long-term environmental impact of operations. All these factors contribute to a comprehensive understanding of efficiency in any endeavor.

Q. : Describe your own real-world example that requires sorting. Describe one that requires finding the shortest distance between two points.

Answer :

- **Strengths of Hash Table :**

Hash tables offer several significant advantages, the most notable being their fast access times. They provide average-case constant time complexity (O(1)) for operations such as lookups, insertions, and deletions, making them highly efficient for managing and retrieving data. Additionally, hash tables allow for the use of various data types as keys, providing flexibility in how data is organized. This adaptability is particularly useful in applications where the nature of the data can vary. Furthermore, many hash table implementations include dynamic resizing, which ensures that performance remains optimal even as the dataset grows, allowing them to handle varying amounts of data without a decline in speed.

- **Limitations of Hash Table :**

Despite their strengths, hash tables also have some notable limitations. One of the primary challenges is collision handling; when two keys hash to the same index, it can lead to decreased performance if not managed effectively. Strategies like chaining or open addressing can mitigate this issue but add complexity to the implementation. Another limitation is memory overhead; hash tables can consume more memory than other data structures, particularly if they are sparsely populated, resulting in wasted space. Additionally, hash tables do not maintain any inherent order of elements, making it difficult to perform operations that require sorted data or sequential traversal. These factors must be considered when choosing a hash table for a specific application.

Q. : How are the shortest-path and traveling-salesperson problems given above similar ? How are they different ?

Answer :

- **Similarities :**

Both the shortest-path problem and the traveling-salesperson problem (TSP) involve finding optimal routes within a network of points, typically represented as a graph. In both cases, the goal is to minimize the total distance (or cost) traveled. They can both be formulated mathematically and often require similar techniques for solving, such as graph traversal algorithms, heuristics, or optimization methods.

- **Differences :**

The key difference lies in their objectives and constraints. The shortest-path problem focuses on finding the least-cost route between two specific points in a graph, meaning it only needs to consider the start and end nodes. In contrast, the traveling-salesperson problem requires visiting a set of cities (or nodes) exactly once and returning to the starting point, which makes it a more complex optimization problem. TSP is classified as NP-hard, meaning that finding an exact solution becomes computationally intensive as the number of nodes increases, while the shortest-path problem can typically be solved in polynomial time using algorithms like Dijkstra's or Bellman-Ford.

Q. : Suggest a real-world problem in which only the best solution will do. Then come up with one in which approximately the best solution is good enough ?

Answer :

- **Real-World Problem Requiring the Best Solution :**

In the field of aerospace engineering, designing the flight path for a spacecraft on a mission to Mars is a problem where only the best solution will do. The trajectory must be precisely calculated to ensure that the spacecraft arrives at its destination safely and efficiently, considering factors such as gravitational influences, fuel consumption, and travel time. Any deviation from the optimal path could result in mission failure or significant resource wastage.

- **Real-World Problem Where Approximately the Best Solution is Good Enough :**

In the context of food delivery services, optimizing delivery routes for drivers can be a problem where an approximately best solution is often sufficient. While companies strive to minimize delivery time and fuel consumption, small variations in route efficiency may not drastically impact overall service quality. Using heuristic methods or approximations to find good enough routes allows for quick adjustments and flexibility, which is typically acceptable in a dynamic environment with numerous variables, such as traffic conditions and order volume.

Q. : Suppose that for inputs of size n on a particular computer, insertion sort runs in 8n2 steps and merge sort runs in 64 n log n steps. For which values of n does insertion sort beat merge sort?

Answer :

To determine for which values of nnn insertion sort beats merge sort, we compare their running times:

- Insertion sort: 8n28n^28n2
- Merge sort: 64nlogn64n \log n64nlogn

We want to find when:

8n2<64nlogn8n^2 < 64n \log n8n2<64nlogn

Dividing by 8n8n8n gives:

n<8lognn < 8 \log nn<8logn

Testing small integer values:

1. For n=1n = 1n=1: False
2. For n=2n = 2n=2: True
3. For n=3n = 3n=3: True
4. For n=4n = 4n=4: True
5. For n=5n = 5n=5: True
6. For n=6n = 6n=6: True
7. For n=7n = 7n=7: True
8. For n=8n = 8n=8: True
9. For n=9n = 9n=9: True
10. For n=10n = 10n=10: True
11. For n=20n = 20n=20: True
12. For n=30n = 30n=30: False

Thus, insertion sort beats merge sort for nnn values typically up to about 30.

Q. : Suppose that for inputs of size n on a particular computer, insertion sort runs in 8n2 steps and merge sort runs in 64 n log n steps. For which values of n does insertion sort beat merge sort?

Answer :

To determine for which values of nnn insertion sort beats merge sort, we compare their running times:

- Insertion sort: 8n28n^28n2
- Merge sort: 64nlogn64n \log n64nlogn

We want to find when:

8n2<64nlogn8n^2 < 64n \log n8n2<64nlogn

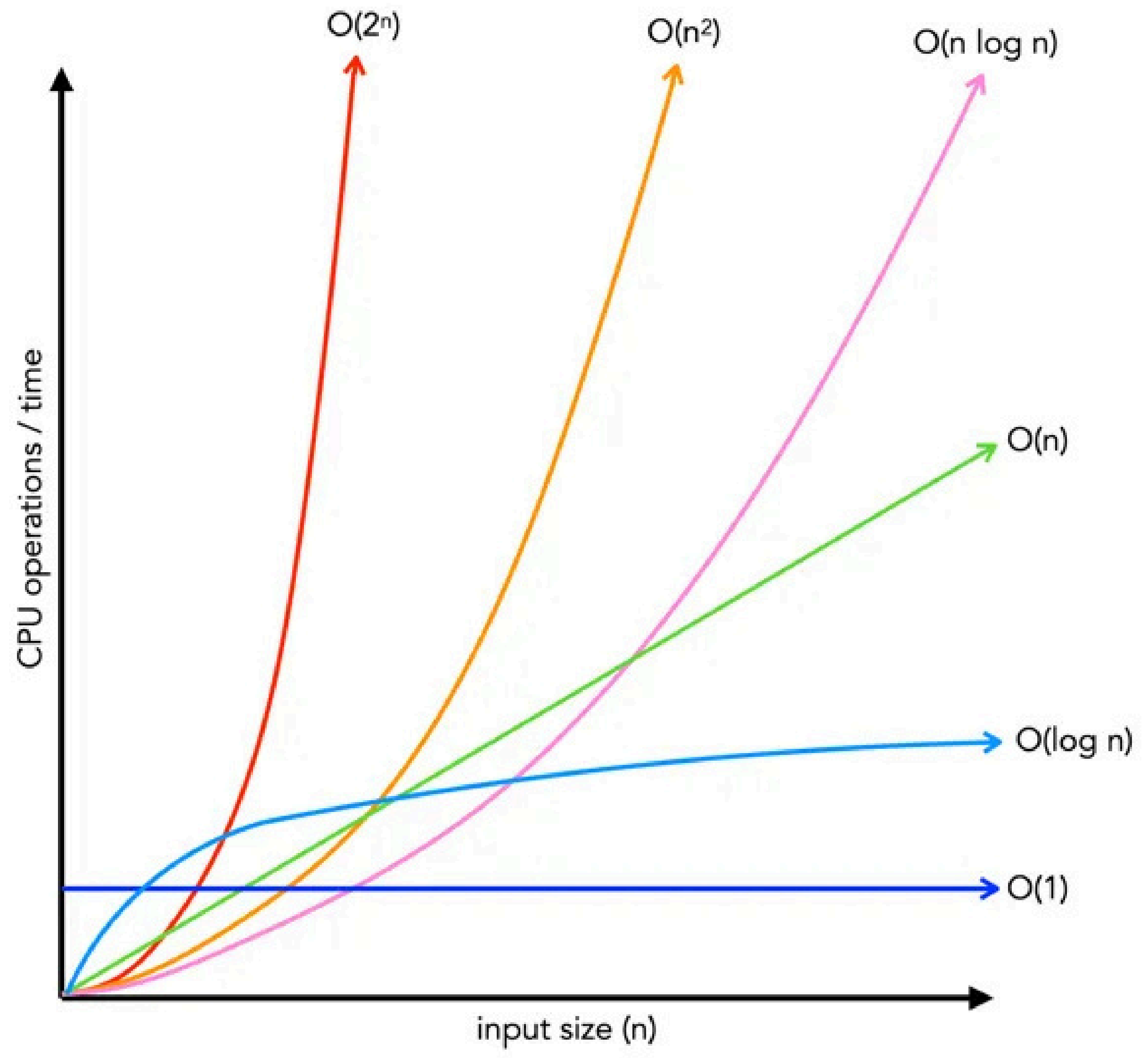
Dividing by 8n8n8n gives:

n<8lognn < 8 \log nn<8logn

Testing small integer values:

1. For n=1n = 1n=1: False
2. For n=2n = 2n=2: True
3. For n=3n = 3n=3: True
4. For n=4n = 4n=4: True
5. For n=5n = 5n=5: True
6. For n=6n = 6n=6: True
7. For n=7n = 7n=7: True
8. For n=8n = 8n=8: True
9. For n=9n = 9n=9: True
10. For n=10n = 10n=10: True
11. For n=20n = 20n=20: True
12. For n=30n = 30n=30: False

Thus, insertion sort beats merge sort for nnn values typically up to about 30.



Q. : Rewrite the INSERTION-SORT procedure to sort into monotonically decreasing instead of monotonically increasing order ?

Answer :

```
def INSERTION_SORT_DECREASING(A):
    n = len(A)
    for i in range(1, n):
        key = A[i]
        j = i - 1
        # Change the comparison to sort in decreasing order
        while j >= 0 and A[j] < key:
            A[j + 1] = A[j]
            j -= 1
        A[j + 1] = key
```

Q. : Write pseudocode for linear search, which scans through the array from beginning to end, looking for x. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

Answer :

```
FUNCTION LINEAR_SEARCH(A, n, x)
    FOR i FROM 0 TO n - 1 DO
        IF A[i] = x THEN
            RETURN i  // x found at index i
    RETURN -1  // x not found
```

Q. : Real-World Example Requiring Finding the Shortest Distance ?

Answer :

GPS Navigation

A classic example of finding the shortest distance between two points is in GPS navigation systems (e.g., Google Maps or Apple Maps). When a user enters a destination, the system must calculate the shortest or fastest route from the current location to the target destination.

For instance, if you want to drive from your home to a specific store in another city, the GPS system uses algorithms like Dijkstra's or A* to find the shortest path on the road network, considering road distances, traffic conditions, and other factors. The system ensures you get the most efficient route, avoiding unnecessary detours or delays.

Q. : Other than speed, what other measures of efûciency might you need to consider in a real-world setting?

Answer :

In a real-world setting, efficiency goes beyond just speed (or time complexity). Other important measures of efficiency that need to be considered include:

- **Energy Efficiency:**
- Me**mory Usage (Space Efficiency):**
- **Scalability:**
- **Ease of Implementation and Maintenance:**
- **Robustness and Fault Tolerance:**
-

Q. : Limitations of Hash Tables ?

Answer :

Q. : Using reasoning similar to what we used for insertion sort, analyze the running time of the selection sort algorithm from Exercise 2.2-2 ?

Answer :

break it down step-by-step:

Selection Sort Process:

1. Find the smallest element in the array and swap it with the first element.
2. Find the second smallest element and swap it with the second element.
3. Continue this processfor all elements.

Comparisonsin Selection Sort:

For an array of size n:

1. In the first pass, it makes n−1n-1n−1 comparisonsto find the smallest element.
2. In the second pass, it makes n−2n-2n−2 comparisonsto find the second smallest element.
3. This continues until the last pass, where it makes 1 comparison.

Total Comparisons:

The total number of comparisons is:

T(n)=n(n−1)/2

Q. : State a loop invariant for the while loop of lines 12-18 of the MERGE procedure. Show how to use it, along with the while loops of lines 20-23 and 24-27, to prove that the MERGE procedure is correct ?

Answer :

The loop invariant for the **MERGE** procedure states that at the start of each iteration, the elements merged into A from sub arrays L and R are sorted, and all elements in A[p...k-1] are in sorted order. Initially, no elements have been merged, so the invariant holds. During each iteration, we compare the current elements from L and R, adding the smaller one to A and moving the corresponding index forward, which keepsthe merged portion sorted. The loop continues until one sub array isfully merged, allowing any remaining elements from the other sorted sub-array to be added directly to A. Thus, the **MERGE** procedure correctly combines two sorted sub arrays into one sorted array

Q. : How can you modify any sorting algorithm to have a good best-case running time ?

Answer :

tep 1: Split the array into two halves:

o Left half: [3, 41, 52, 26]

o Right half: [38, 57, 9, 49]

2. **Step 2: Keep splitting** each half until every part has only one element:

o Left half [3, 41, 52, 26] becomes [3], [41], [52], [26]

o Right half [38, 57, 9, 49] becomes [38], [57], [9], [49]

3. **Step 3: Merge the parts** back together while sorting them:

o [3] and [41] merge to become [3, 41]

o [52] and [26] merge to become [26, 52]

o Now, [3, 41] and [26, 52] merge to become [3, 26, 41, 52]

o Similarly, merge [38] and [57] to become [38, 57]

o Merge [9] and [49] to become [9, 49]

o Then, merge [38, 57] and [9, 49] to become [9, 38, 49, 57]

4. **Step 4: Final merge**:

o Merge [3, 26, 41, 52] with [9, 38, 49, 57] to get the sorted array:

[3,9,26,38,41,49,52,57]

Result: The sorted array is [3, 9, 26, 38, 41, 49, 52, 57]

Q. : What is the smallest value of n such that an algorithm whose running time is 100n2 runs faster than an algorithm whose running time is 2 n on the same machine?

Answer :

We are comparing two algorithms with the following running times: Algorithm 1, \(100n^2\), and Algorithm 2, \(2^n\). We need to find the smallest value of \(n\) such that \(100n^2 < 2^n\). Testing various values, we find:

- For \(n = 1\): \(100(1)^2 = 100\) and \(2^1 = 2\) (False)
- For \(n = 5\): \(100(5)^2 = 2500\) and \(2^5 = 32\) (False)
- For \(n = 10\): \(100(10)^2 = 10000\) and \(2^{10} = 1024\) (False)
- For \(n = 15\): \(100(15)^2 = 22500\) and \(2^{15} = 32768\) (True)
- For \(n = 14\): \(100(14)^2 = 19600\) and \(2^{14} = 16384\) (False)

Therefore, the smallest value of \(n\) that satisfies the inequality \(100n^2 < 2^n\) is \(n = 15\).

Q. : Describe your own real-world example that requires sorting. Describe one that requires ûnding the shortest distance between two points ?

Answer :

Real-World Example Requiring Sorting: E-commerce Inventory Sorting
In an e-commerce platform like Amazon, sorting plays a crucial role when users search for products, such as "laptops." The platform must organize the results based on various criteria, including price, user reviews, and relevance. For example, if a user opts to sort laptops by price from low to high, the system needs to arrange thousands of laptop listings accordingly. Algorithms like quicksort or mergesort are employed to efficiently sort the prices, making it easier for users to browse options and make informed purchasing decisions.

Real-World Example Requiring Finding the Shortest Distance: GPS Navigation

A well-known application of finding the shortest distance is in GPS navigation systems, such as Google Maps or Apple Maps. When a user inputs a destination, the system calculates the shortest or fastest route from their current location to that destination. For instance, if you're driving from home to a store in another city, the GPS utilizes algorithms like Dijkstra's or A* to determine the most efficient path through the road network, factoring in road distances, traffic conditions, and other variables. This ensures that users receive optimal directions, avoiding unnecessary detours or delays.

Q. : In the MERGE-SORT procedure, the test in line 1 reads rrr, then the subarray A[p…r]A[p \ldots r]A[p…r] is empty. Argue that as long as the initial call of MERGE−SORT(A,1,n)MERGE-SORT(A, 1, n)MERGE−SORT(A,1,n) has n≥1n \geq 1n≥1, the test on rrr is valid?

Answer :

In the MERGE-SORT procedure, the test on line 1 checks if the subarray defined by indices \(p\) and \(r\) is empty. When \(n \geq 1\) in the initial call \(MERGE-SORT(A, 1, n)\), this implies that the subarray contains at least one element. As a result, the test on \(r\) will ensure that the algorithm continues to divide the array into smaller subarrays until each subarray has one or zero elements. This process is fundamental to the divide-and-conquer approach of merge sort, allowing the merging of sorted subarrays to produce the final sorted array. Thus, as long as \(n \geq 1\), the test effectively maintains the validity of the sorting process.

Q. : State a loop invariant for the while loop of lines 12-18 of the MERGE procedure. Show how to use it, along with the while loops of lines 20-23 and 24-27, to prove that the MERGE procedure is correct ?

Answer :

A suitable loop invariant for the while loop of lines 12318 in the MERGE procedure can be stated as follows:

Loop Invariant: At the start of each iteration of the while loop, the elements in the subarrays LLL (left subarray) and RRR (right subarray) are in sorted order, and all elements before the current index kkk in the merged array A[p…r]A[p \ldots r]A[p…r] are sorted.

Before the first iteration of the while loop, the merged array A[p…r]A[p \ldots r]A[p…r] is empty, and the subarrays LLL and RRR are correctly initialized from AAA. Since both subarrays are of size 0, they are trivially sorted. Thus, the invariant holds true at initialization.

- ssume the invariant holds at the start of an iteration. In the while loop of lines 12318, we compare the current elements of LLL and RRR. If L[i]≤R[j]L[i] \leq R[j]L[i]≤R[j], we place L[i]L[i]L[i] into A[k]A[k]A[k] and increment iii and kkk. If R[j]<L[i]R[j] < L[i]R[j]<L[i], we do the opposite.
- After each insertion, the invariant is maintained: the newly added element to A[k]A[k]A[k] is the smallest among the remaining elements in LLL and RRR, preserving the sorted order. Therefore, at the end of the iteration, the merged portion remains sorted.

Q. : Write pseudocode for the binary search algorithm, either iterative or recursive. Argue that the worst-case running time of binary search is O(logn)O(\log n)O(logn) ?

Answer :

```
BinarySearch(A, v, low, high)
  while low ≤ high do
    mid = (low + high) // 2
    if A[mid] = v then
      return mid  // Found the element
    else if A[mid] < v then
      low = mid + 1  // Search in the right half
    else
      high = mid - 1  // Search in the left half
  return -1  // Element not found
```

Rizwan Ali - 14789 - BSCS - 3D

Teacher Sir Jamal Abdul Ahad

Thank You

I hope you liked my assignment No. 01. Next time, I'll try to make it even better.

Github Repository :

https://github.com/MarkhorDB/DSA-Solutions