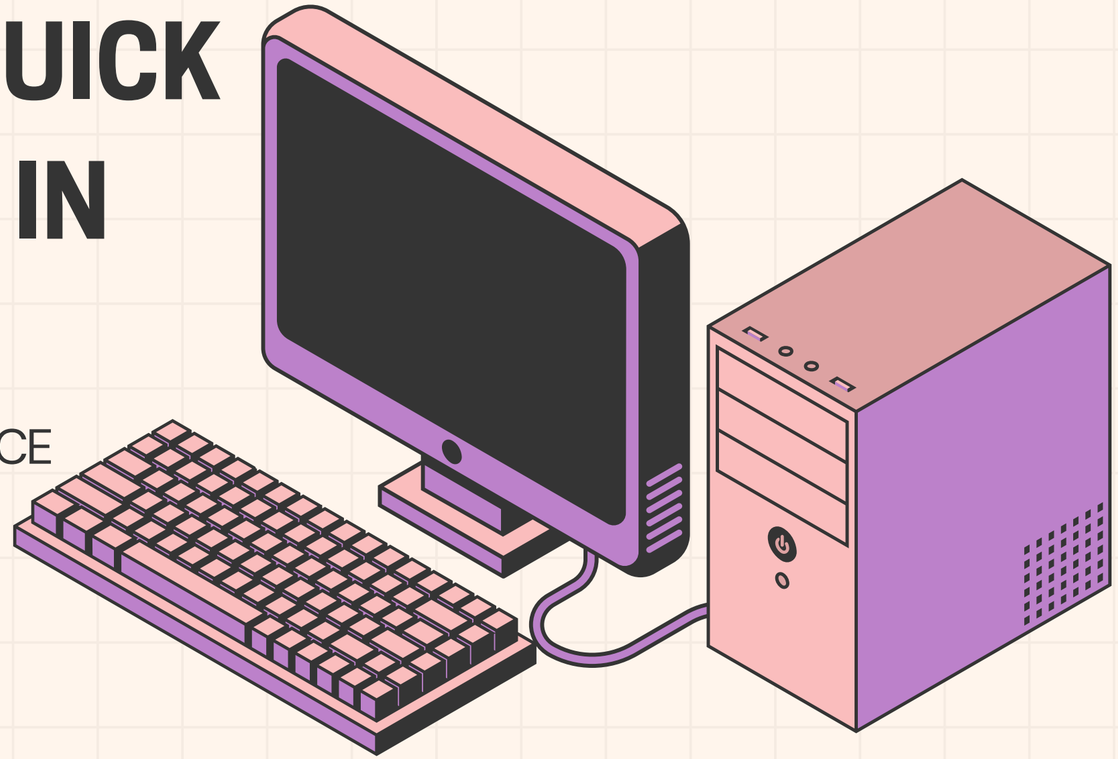
OPTIMIZING E-COMMERCE PRODUCT SORTING USING QUICK SORT AND ITS APPLICATION IN ONLINE RETAILERS

A STUDY ON EFFICIENT SORTING ALGORITHMS IN E-COMMERCE



Team Members :

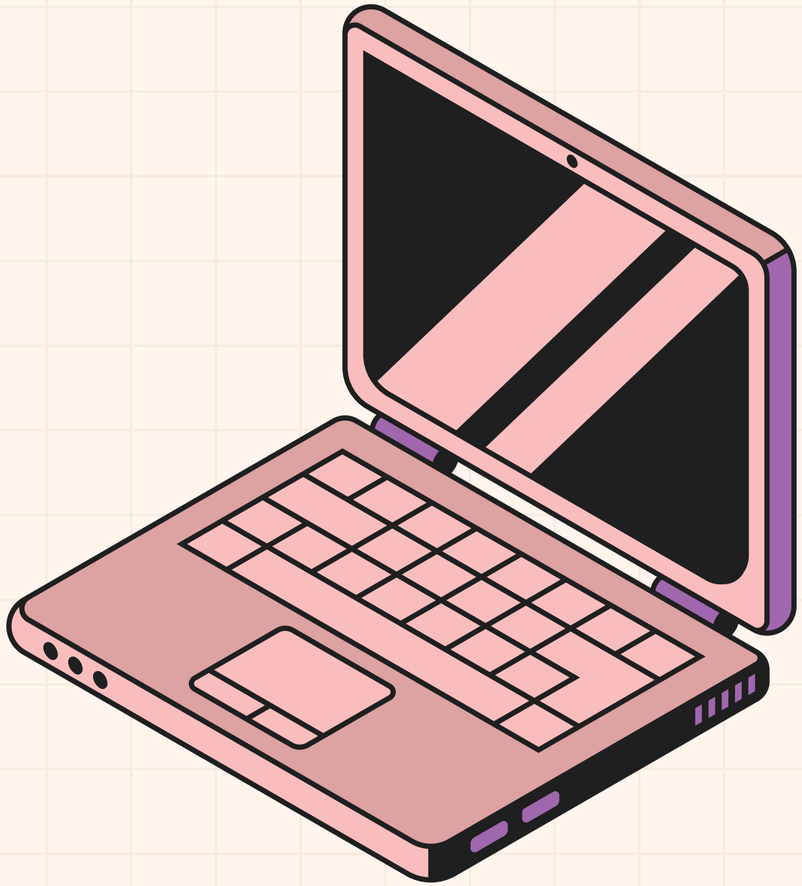
Anish Sethi [RA2311003010365]

Yanamala Uhas Reddy [RA2311003010363]

Kritak Prasad [RA2311003010367]

INTRODUCTION

- E-commerce platforms deal with large volumes of product data that need to be organized efficiently.
- Sorting algorithms are essential for optimizing user experience and search functionality.
- Quick Sort is one of the most efficient algorithms used in sorting.



PROBLEM SELECTION AND DESCRIPTION

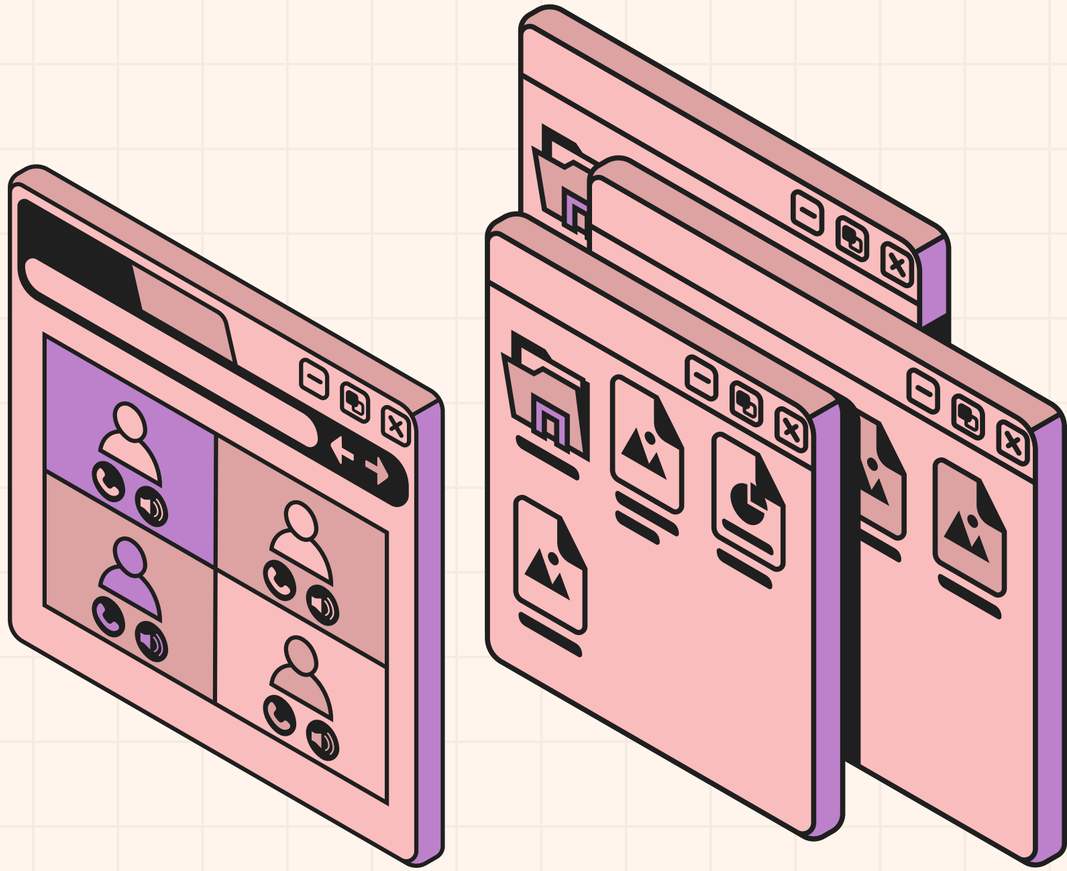
Problem Overview:

- Goal: Efficiently sort a large catalog of products (by price, rating, popularity, etc.) in online retail stores.
- Challenges:
  - The problem of sorting millions of products.
  - Sorting needs to be fast, especially in peak hours.
  - Ensuring the sorting process does not impact the user experience negatively.
- Importance for Online Retailers:
  - Affects search accuracy, user satisfaction, and conversion rates.
  - Improper sorting can lead to poor product discovery and lost sales

ALGORITHMIC TECHNIQUE - QUICK SORT

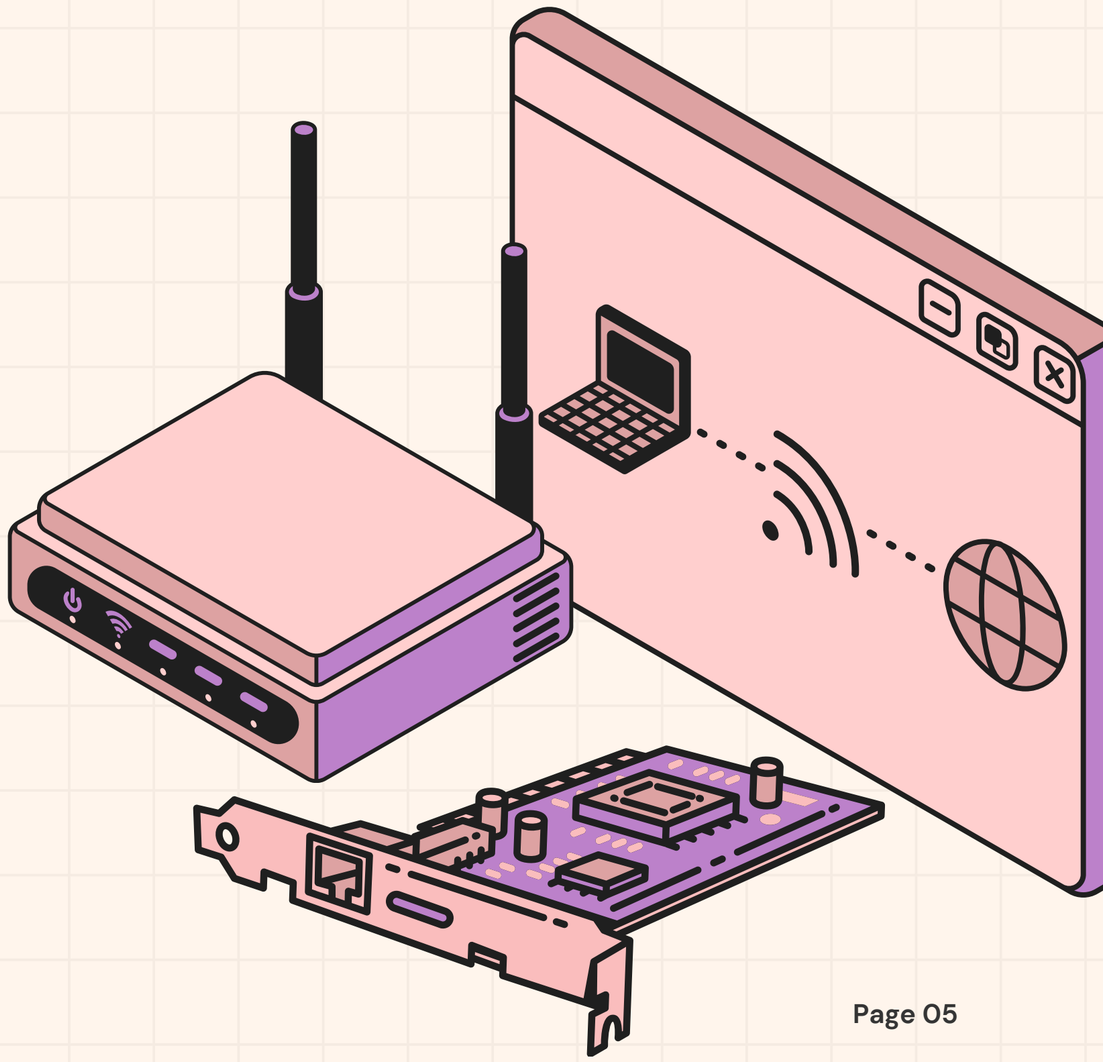
- Quick sort algorithm overview:
  - What is Quick Sort?
    - A divide-and-conquer algorithm.
    - It works by selecting a 'pivot' element and partitioning the array into two sub-arrays: one with elements smaller than the pivot and one with elements greater than the pivot.
    - Recursively sorts the sub-arrays.
- Steps in Quick Sort:

1. Choose a pivot element (which can be the first, middle, or random element).
2. Partition the list into two parts: the left part with smaller elements and the right part with larger elements.
3. Recursively apply Quick Sort on the left and right sub-arrays.
4. Combine the sub-arrays to form the sorted array.
   - Visualization of Quick Sort Process (Optional: Add a flowchart or example).

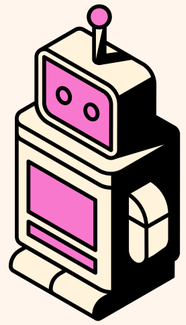


WHY QUICK SORT?

- Why is Quick Sort Ideal for E-commerce?
  - Efficiency: O(n log n) average time complexity for large data sets, which is better than $O(n^2)$ in other algorithms like bubble sort.
  - Space Efficient: Quick Sort uses in-place sorting, which reduces memory overhead.
  - Scalability: It is efficient for the massive datasets typical of e-commerce platforms.
  - Flexibility: Can be customized for sorting by different attributes (price, rating, etc.).



HOW QUICK SORT WORKS?



01.
CHOOSE A PIVOT

SELECT A PIVOT ELEMENT FROM THE LIST. THIS CAN BE DONE IN VARIOUS WAYS, SUCH AS PICKING THE FIRST ELEMENT, THE LAST ELEMENT, THE MIDDLE ELEMENT, OR USING A RANDOM ELEMENT.

02.
PARTITIONING

REARRANGE THE LIST SO THAT ALL ELEMENTS LESS THAN THE PIVOT COME BEFORE IT, AND ALL ELEMENTS GREATER THAN THE PIVOT COME AFTER IT. THE PIVOT IS NOW IN ITS FINAL SORTED POSITION.

03.
RECURSIVELY APPLY THE SAME LOGIC

RECURSIVELY SORT THE SUBLIST OF ELEMENTS BEFORE THE PIVOT AND THE SUBLIST OF ELEMENTS AFTER THE PIVOT.
CONTINUE THIS PROCESS UNTIL THE BASE CASE IS REACHED, WHERE THE LIST HAS ONLY ONE ELEMENT OR IS EMPTY.

04.
FINAL SORTED LIST

ONCE ALL PARTITIONS ARE SORTED, THE ENTIRE LIST WILL BE SORTED

STEPS INVOLVE IN QUICK SORT

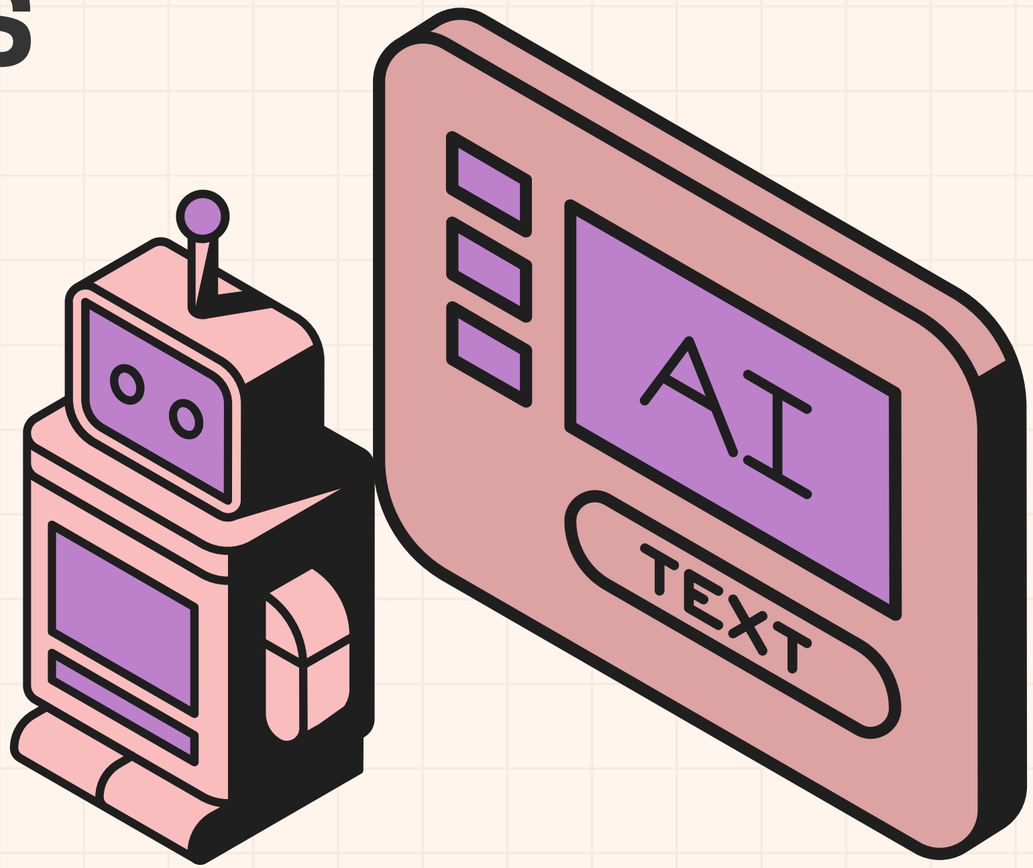
PSEUDO-CODE

```
def quickSort(products, low, high):
  if low < high:
      pivot_index = partition(products, low, high)
quickSort(products, low, pivot_index - 1)
quickSort(products, pivot_index + 1, high)

def partition(products, low, high):
  pivot = products[high]
  i = low - 1
  for j in range(low, high):
        if products[j].price <= pivot.price: # Change this line if sorting by other criteria like
rating
        i += 1
products[i], products[j] = products[j], products[i]
  products[i + 1], products[high] = products[high], products[i + 1]
  return i + 1
```

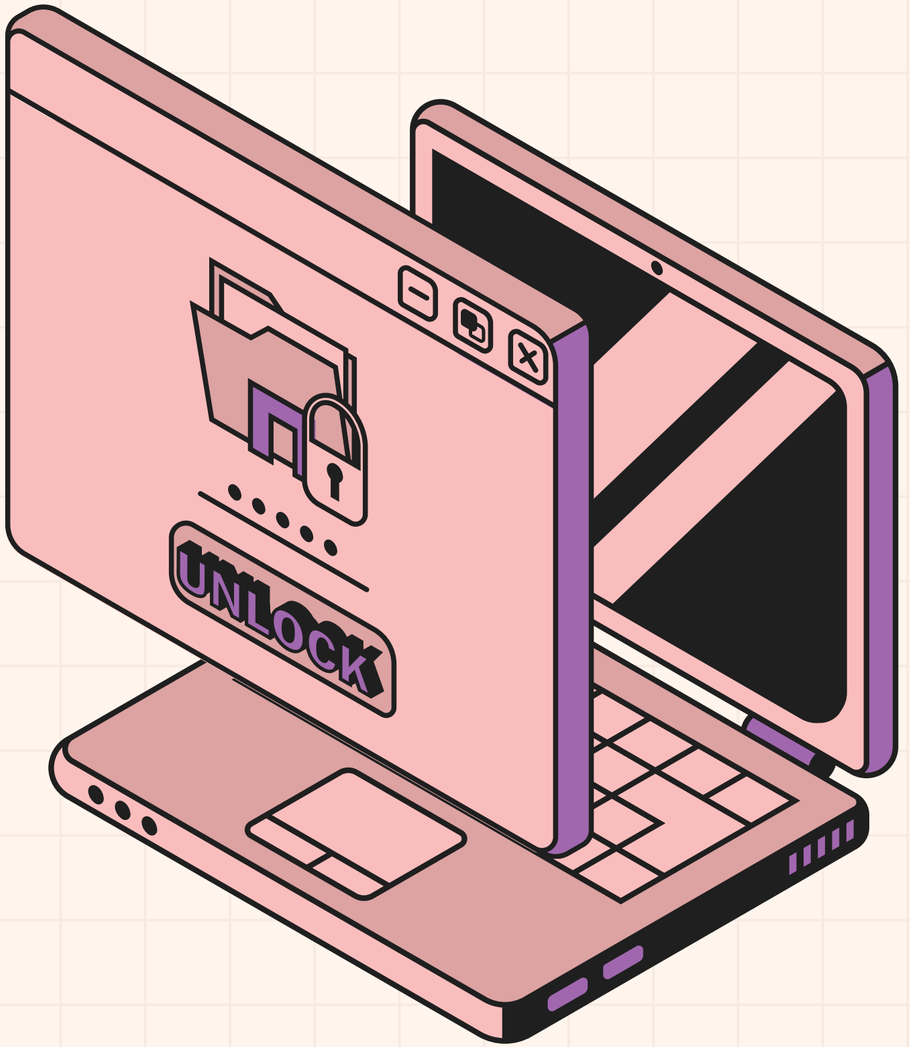
TIME COMPLEXITY ANALYSIS

- Best, Worst, and Average Case Time Complexity:
  - Best Case: O(n log n) (when the pivot divides the array evenly).
  - Worst Case: $O(n^2)$ (when the pivot is the smallest or largest element, leading to unbalanced partitions).
  - Average Case: O(n log n) (which is typically expected in practical use cases).
- Space Complexity: O(log n) for the recursive stack (in the best case).
- Why Quick Sort's Time Complexity Matters:
  - Importance for E-commerce:
    - Quick Sort handles large datasets efficiently.
    - Reduces latency in sorting product catalogs, ensuring faster page load times and improved user experience.
- Comparison to Other Algorithms:
  - Compared to bubble sort ($O(n^2)$) or insertion sort ($O(n^2)$), Quick Sort performs much better with larger data sets.



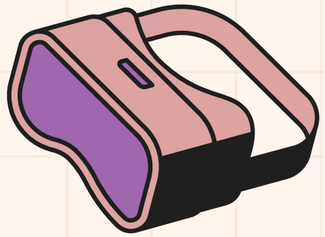
OUTCOMES OF USING QUICK SORT IN E-COMMERCE

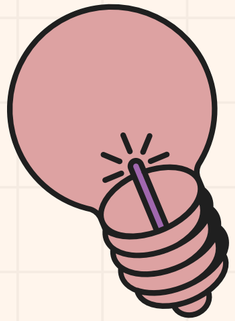
- Performance Impact:
  - Improved sorting time and user experience, leading to faster product discovery.
  - Reduces server load by using in-place sorting.
- Scalability:
  - Can handle millions of products during peak times (e.g., Black Friday, Cyber Monday).
- Practical Applications:
  - Sorting products by price, reviews, popularity, or newly arrived items.
  - Sorting results on the backend for faster front-end rendering.
  - Quick Sort helps handle the dynamic nature of e-commerce data (frequent updates in products and prices).



CONCLUSION

- Summary of the Problem and Solution:
  - Quick Sort provides an efficient solution for sorting large product catalogs in e-commerce.
  - It minimizes sorting time, enhancing user experience and search functionality.
- Impact on Online Retailers:
  - Increased conversion rates due to better user interaction with the site.
  - Handling large product databases with minimal delays.





THANK YOU