MergeSearch: A Smart Disaster Response System

Abdullah Mohammad Sadman [1] Fahim Shahriar [2] Arman Uddin [3] MD Alvee Sarker [4]
Abdullah Al Hossain[5] Tanjil Sarker[6]

Department of Computer Science and Engineering
Independent University, Bangladesh
Dhaka, Bangladesh.
{2330322[1],2310688[2],2311300[3],2311249[4],2310778[5],2312026[6]}@iub.edu.bd

A Dive into Merge Sort and Binary Search Trees

Merge Sort and Binary Search Tree are both **Divide-and-conquer algorithm with a Dynamic Structure that guarantees stable sorting and searching** each playing a crucial role in efficient data processing and management. Efficient data processing and management are pivotal in various domains, from disaster response to database optimization. Merge Sort and Binary Search Tree stand out as foundational tools for achieving these objectives. Merge Sort is an efficient, stable sorting algorithm that divides data into smaller subsets, sorts them, and merges the results. A Binary Search Tree (BST) stores data in a hierarchical structure, allowing fast searching, insertion, and deletion. Combining Merge Sort and BST enhances both sorting and searching, providing an effective solution for data organization and efficient querying in real-world applications.

Origin and Background

Merge sort, developed by **John von Neumann in 1945**, is a divide-and-conquer algorithm that efficiently sorts data. It was introduced in his paper **"Algorithms for Solving Mathematical Problems."** The method splits arrays into smaller subarrays, sorts them, and merges them back together, making it particularly effective for large datasets and linked lists.

Binary search, an efficient algorithm for finding an item in a sorted list, dates back to ancient times. Its origins can be traced to the works of mathematicians like **Euclid and later formalized by John von Neumann in the mid-20th century**. The method halves the search space with each step, making it significantly faster than linear search, especially for large datasets.

How Both Algorithms are used in Disaster Management

Merge Sort and Binary Search Trees are useful in **disaster management** for efficiently handling large datasets. They help in sorting resources, managing real-time data, and making quick decisions during critical situations.

$MergeSort$ **helps by sorting resources, rescue operations, and GIS data, ensuring quick allocation and decision-making.**

BST **allows efficient storage and retrieval of real-time data like affected areas, rescue teams, and available resources, speeding up searches and updates for better coordination.**

Breakdown

Merge Sort:

- Splits the dataset into smaller parts.
- Sorts each part and then merges them into a final ordered dataset.
- Helps prioritize and allocate resources (e.g., medical supplies, food) quickly in disaster situations.

Binary Search Tree (BST):

- Data is stored in a hierarchical structure where each node has a left and right child.
- Allows efficient search, insertion, and deletion of real-time data (e.g., locations of affected areas, available rescue teams).
- Enhances coordination by providing faster access to critical information during a disaster response.

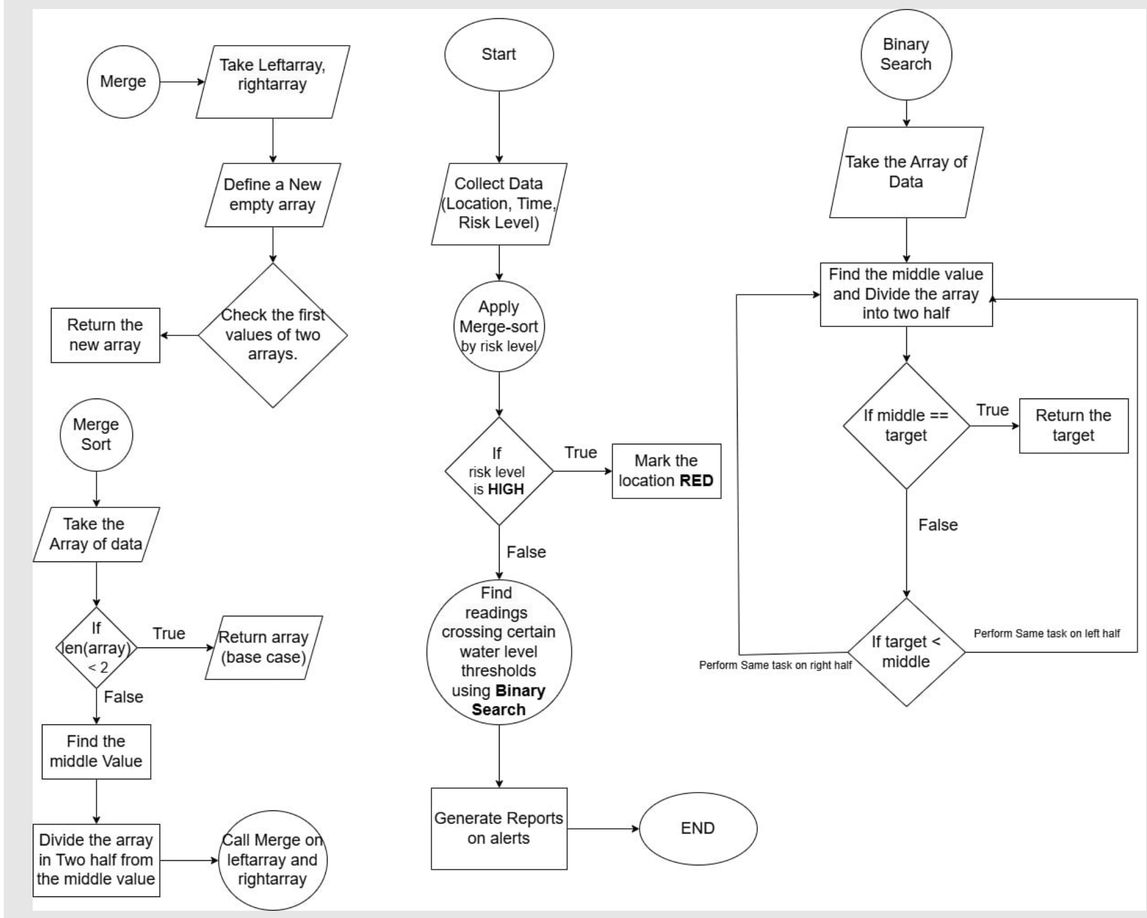
Python Implementation

Here's a python program showcasing the implementation of Merge and BST in Disaster Management

```python
class BST:
    #structure of Binary search tree
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]
        merge_sort(left_half)
        merge_sort(right_half)
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1
        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1
def manageResources(resources):
    merge_sort(resources)           # Sorting the resources ordered by priority
    print("Sorted resources:", resources)
def manageDisasterAreas():
    tree = BST() # insertion of disaster areas with their severity levels
    tree.insert("Area A: Severe")
    tree.insert("Area B: Moderate")
    tree.insert("Area C: Critical")
    tree.insert("Area D: Mild")
    area = "Area C: Critical" # Searching for a critical area
    searchedNode = tree.search(area)
    if searchedNode is not None:
        print("Found in BST:", searchedNode.value)
    else:
        print("Location not found in BST")
resources = [#Resources in unsorted Order] # Step 3: Example Workflow for Disaster Management
print("Unsorted resources:", resources)
manageResources(resources)
manageDisasterAreas()
```

Flow Chart



Complexities Analysis

Choice of Stable Algorithm: The implementation combines Merge Sort for sorting resources and Binary Search Tree (BST) operations for inserting and searching disaster areas. Merge Sort provides stable sorting of input elements, while the BST enables efficient organization and lookup of disaster areas.
Divide and Conquer Approach: Merge Sort follows a divide-and-conquer strategy. The array is repeatedly divided into two equal parts, and sorting occurs as the recursion unwinds. The division takes$O(logn)$ levels, and merging the divided parts takes $O(n)$ time at each level, leading to a total runtime of $O(nlogn)$
Time Complexity:$O(nlogn)$ **Space Complexity:**$O(n)$
Binary Search Tree (BST) Operations:
Time Complexity: $O(h)$ where h is the height of the tree. For a balanced BST,$h = logn$ giving a time complexity of $O(logn)$
Overall Efficiency: Merge Sort and BST operations combine to create an efficient workflow for sorting and managing resources, maintaining the overall time complexity of $O(nlogn)$

Real Life Scenario

Flood Forecasting and Warning Centre of Bangladesh collects and analyzes data(where the given sorting and searching plays vital role) from various sources to predict potential flood events. By issuing timely warnings based on these forecasts, the center helps authorities and communities prepare and respond effectively, minimizing the impact of floods on lives and infrastructure.

Conclusion

In conclusion, the implementation of Merge Sort and Binary Search algorithms in a disaster management system plays a crucial role in ensuring efficient data organization and quick retrieval of critical information during emergency situations. By combining these powerful algorithms, the system can effectively sort and search through data, enabling timely decision-making and response efforts. This innovative approach enhances the system's capability to handle large volumes of data and streamline disaster management processes, ultimately contributing to improved disaster response and mitigation outcomes.