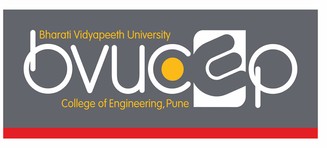


# BHARATI VIDYAPEETH

**(Deemed to be University) COLLEGE OF ENGINEERING, PUNE (2024-25)**



# In partial fulfillment of the requirement for project-based Learning of BACHELOR OF TECHNOLOGY

**In COMPUTER**

# ENGINEERING

**For**

# Subject: - Algorithm Design and Analysis

**PROJECT BASED LEARNING REPORT ON**

# Topic:

**“Optimal Merge Pattern”.**

# Project report submitted by:

**Roll No 45**

**Name**

**VINAY MALI**

**PRN**

**2114110323**

**Project Report submitted to:**

**(SIGNATURE)**

# Prof: Mrs. Sheetal S. Patil

**TABLE OF CONTENT**

**Page No.**

**01**

**02**

**03-04**

**05-06**

**06-10**

**11-13**

**14**

**15-17**

**18**

**Topic Acknowledgment Abstract**

**Introduction**

**Greedy Algorithm Optimal merge pattern**

**Code Input**

**Code Output**

**GitHub**

**Conclusion**

**Serial No.**

**01**

**02**

**03**

**05**

**06**

**07**

**08**

**09**

**10**

# Acknowledgement

We would like to express our special thanks of gratitude to our Project Supervisor, Prof. **Mrs. Sheetal S. Patil** for her sincere guidance, inspiration and cooperation in this project.

We gained a lot of knowledge on the topics that were involved in this project and we are sure that it will prove to be very useful for us in the future.

**Abstract**

Our project presents an innovative **Optimal Merge Pattern Application** that utilizes the principles of the Optimal Merge Pattern algorithm to streamline file merging processes efficiently. This application is designed to handle a variety of file sizes, ensuring quick and optimal file combinations while minimizing the total merging cost.

The Optimal Merge Pattern Application features a user-friendly interface that simplifies the input of file sizes and displays the merging steps clearly, making it accessible to users of all skill levels. This enhances the overall user experience by providing an intuitive way to visualize the merging process and its costs.

With its powerful merging capabilities, the application ensures that files are combined in a manner that reduces overall merge costs. It employs a min-heap data structure for efficient retrieval of the smallest file sizes, making the merging process both fast and efficient.

By hosting this project on GitHub, we encourage collaboration and contributions from developers and enthusiasts worldwide. This open-source initiative not only allows for ongoing improvements and feature enhancements but also serves as a valuable educational resource for understanding the optimal merge pattern and its applications in computer science. Join us in refining and expanding this cutting-edge solution for efficient file management in the digital era.

# INTRODUCTION

The **Optimal Merge Pattern Application** is an innovative digital solution designed to transform the way we handle file merging tasks. In a world that requires efficient data processing, this application aims to simplify the often complex process of merging files based on their sizes. By integrating advanced features, a user-friendly design, and robust computational methods, this project offers an intuitive platform for individuals and businesses alike.

### Key Features

Our application boasts a comprehensive set of features:

**File Size Management:** Users can input and manage a variety of file sizes, allowing for efficient calculations of merging costs based on optimal algorithms.

 **User-Friendly Interface**: The application's intuitive interface ensures accessibility for users of all technological backgrounds, promoting efficient interaction with the merging process. Users can easily input file sizes and visualize the merging steps.

 **Real-Time Visualization**: A powerful visualization feature enables users to see the merging process in action, providing a clear understanding of each step and the resultant file sizes.

 **Step-by-Step Calculation**: Keeping track of the merging process is straightforward, as users can view each step of the algorithm, including the files merged, new file sizes, and cumulative costs.

 **Algorithm Efficiency**: Stringent security measures, including password protection and encryption, safeguard sensitive contact data. Users have control over access permissions.

**Performance Metrics:** Users can review performance metrics that demonstrate the efficiency of the merging algorithm, allowing for analysis and improvement of their file management strategies.

**Security and Privacy:** Stringent security measures safeguard sensitive data during processing, ensuring that file sizes and details remain confidential.

**Greedy Algorithm**

The **Greedy Algorithm** approach is a fundamental problem-solving strategy in computer science and mathematics. It involves making a sequence of choices that are locally optimal at each step with the hope that these choices will lead to a globally optimal solution. The primary idea behind this approach is to simplify problem-solving by focusing on immediate benefits rather than considering the entire problem space.

Here's a brief definition and introduction to the Greedy Algorithm approach:

**Definition:**

The Greedy Algorithm is a problem-solving technique that involves making the best choice at each step based on a specific criterion without reconsidering previous choices. The process can be summarized in three main steps:

1. **Selection:** At each step, the algorithm makes a choice that looks the best at that moment. This choice is often based on a heuristic or a set of rules that determine the most promising option available.
2. **Feasibility:** The chosen option is checked for feasibility to ensure it does not violate any constraints of the problem. If it is feasible, it is included in the solution set.
3. **Solution:** The algorithm continues to make selections and includes them in the solution until a complete solution is formed.

This approach is typically iterative and does not involve backtracking. Greedy Algorithms are widely used in various applications, including optimization problems, scheduling, and resource allocation, and they are particularly useful for problems where local optimization leads to a global solution.

# Algorithm

* **Initialization:**

1. Begin with a list of file sizes that need to be merged.
2. Utilize a min-heap (or priority queue) to efficiently retrieve the smallest file sizes.

* **Merge Process:**

1. While there is more than one file size in the heap:
   * Extract the Two Smallest Files:
     + Remove the two smallest file sizes from the heap.
   * Merge:
     + Calculate the cost of merging these two files by adding their sizes together.
   * Insert the Merged File:
     + Insert the merged file size back into the heap.
   * Track the Total Cost:
     + Accumulate the merging costs to determine the total cost of the optimal merge.

* **Completion:**

1. Continue the process until only one file size remains in the heap, which represents the final merged file.
2. The total accumulated cost reflects the optimal cost for merging all files.

# Pseudo code

FUNCTION OptimalMergePattern(file\_sizes):

# Step 1: Initialize a min-heap with the given file sizes

min\_heap = MinHeap(file\_sizes)

total\_cost = 0 # To keep track of the total merge cost

# Step 2: Continue merging until only one file remains

WHILE min\_heap.size() > 1:

# Step 2a: Extract the two smallest files

first\_smallest = min\_heap.extract\_min()

second\_smallest = min\_heap.extract\_min()

# Step 2b: Calculate the merge cost

merge\_cost = first\_smallest + second\_smallest

# Step 2c: Insert the merged file size back into the heap

min\_heap.insert(merge\_cost)

# Step 2d: Update the total merge cost

total\_cost += merge\_cost

# Step 3: Return the total merge cost

RETURN total\_cost

**Explanation:**

* **MinHeap:** A data structure that allows for efficient retrieval of the smallest elements.
* **extract\_min():** Removes and returns the smallest element from the heap.
* **insert(value):** Inserts a new value into the heap.
* The algorithm continues to merge the smallest files until only one file remains, tracking the total cost along the way.

# Example

A diagram of a tree

Description automatically generated

**Step-by-Step Merging Process:**

1. **Initial Input**: The file sizes are **2, 3, 4, 5, 6, 7**.
2. **Create a Min-Heap**:
   * We start by placing all file sizes into a min-heap (or priority queue), which allows us to efficiently access the smallest elements.
   * Min-Heap: **[2, 3, 4, 5, 6, 7]**
3. **First Merge**:
   * **Merge**: Take the two smallest files, **2** and **3**.
   * **Cost of Merge**: **2 + 3 = 5**
   * **New File Size**: **5**
   * **Updated Min-Heap**: Add the new file back into the heap.
   * Min-Heap after merge: **[4, 5, 5, 6, 7]**
   * **Total Cost**: **5**
4. **Second Merge**:
   * **Merge**: Take the next two smallest files, **4** and **5**.
   * **Cost of Merge**: **4 + 5 = 9**
   * **New File Size**: **9**
   * **Updated Min-Heap**:
   * Min-Heap after merge: **[5, 6, 7, 9]**
   * **Total Cost**: **5 + 9 = 14**
5. **Third Merge**:
   * **Merge**: Take the two smallest files, **5** and **6**.
   * **Cost of Merge**: **5 + 6 = 11**
   * **New File Size**: **11**
   * **Updated Min-Heap**:
   * Min-Heap after merge: **[7, 9, 11]**
   * **Total Cost**: **14 + 11 = 25**
6. **Fourth Merge**:
   * **Merge**: Take the two smallest files, **7** and **9**.
   * **Cost of Merge**: **7 + 9 = 16**
   * **New File Size**: **16**
   * **Updated Min-Heap**:
   * Min-Heap after merge: **[11, 16]**
   * **Total Cost**: **25 + 16 = 41**
7. **Final Merge**:
   * **Merge**: Finally, take **11** and **16**.
   * **Cost of Merge**: **11 + 16 = 27**
   * **New File Size**: **27**
   * **Updated Min-Heap**: The heap is now empty.
   * **Total Cost**: **41 + 27 = 68**

* **CONCLUSION:**
* The **Optimal Merge Pattern** is a highly effective algorithmic approach, particularly in scenarios requiring efficient data merging, such as file compression and transmission. Utilizing a **greedy algorithm** strategy, it minimizes overall merging costs by continuously selecting the smallest files to merge.
* In our example with file sizes **2, 3, 4, 5, 6, 7**, the total merging cost was **68**, illustrating the method's efficiency. This approach embodies **divide and conquer** principles by breaking down the merging process into smaller steps, allowing larger merges to occur less frequently.
* The use of a **min-heap** enables logarithmic time complexity for insertion and removal operations, enhancing efficiency even with numerous files. The Optimal Merge Pattern has practical applications in various fields, including data compression techniques and resource management in networks.
* In summary, this algorithm highlights the significance of strategic problem-solving in computer science, emphasizing how greedy algorithms and efficient data structures can lead to optimal solutions in complex computational challenges.

# Performance Evaluation

**Analysis of the Optimal Merge Pattern Time Complexity:**

### Best Case Time Complexity: The best case occurs when the file sizes are such that the smallest files can be merged without many comparisons. Even in this scenario, the optimal merge pattern must traverse all files, resulting in a time complexity of O(nlogn). This is because each merge operation involves logarithmic comparisons based on the heap structure.

### Average Case Time Complexity

### In typical situations, where file sizes are varied and not specifically arranged, the average case complexity remains O(nlogn). The merging process requires consistent access to the smallest elements in the min-heap, leading to logarithmic operations for each of the n files.

### Worst Case Time Complexity

## The worst-case scenario occurs when the file sizes are arranged in such a way that each merge requires maximum comparisons, similar to the average case. The time complexity still remains O(nlogn), as the algorithm continuously manages the min-heap to ensure the smallest files are selected for merging.

# CODE

## PYTHON CODE (USING STREAMLIT FOR INTERFACE)-

import streamlit as st

import heapq

import time

import matplotlib.pyplot as plt

def draw\_visualization(file\_sizes, merge\_cost, step):

    # Create the figure and axis

    fig, ax = plt.subplots(figsize=(8, 3))

    ax.set\_title(f"Step {step}: Merged file size = {merge\_cost}", fontsize=14)

    x\_offset = 0

    colors = ['#FF6F61', '#6FA6FF', '#FFD27F', '#95E06C', '#F392B4']

    # Draw the rectangles

    for i, size in enumerate(file\_sizes):

        rect\_width = size  # scaled for better fit

        ax.barh(1, rect\_width, left=x\_offset, color=colors[i % len(colors)], edgecolor='black')

        ax.text(x\_offset + rect\_width / 2, 1, str(size), va='center', ha='center', color='white', fontsize=12)

        x\_offset += rect\_width

    ax.set\_yticks([])

    ax.set\_xticks([])

    ax.set\_xlim(0, sum(file\_sizes) + 10)

    st.pyplot(fig)

    time.sleep(1)  # Pause to simulate animation

def optimal\_merge\_pattern(file\_sizes):

    steps = ""

    total\_cost = 0

    heapq.heapify(file\_sizes)

    step = 1

    placeholder = st.empty()

    while len(file\_sizes) > 1:

        first\_smallest = heapq.heappop(file\_sizes)

        second\_smallest = heapq.heappop(file\_sizes)

        merge\_cost = first\_smallest + second\_smallest

        total\_cost += merge\_cost

        heapq.heappush(file\_sizes, merge\_cost)

        steps += f"Merge {first\_smallest} and {second\_smallest}: New file = {merge\_cost}, Total cost = {total\_cost}\n"

        with placeholder.container():

            draw\_visualization(file\_sizes, merge\_cost, step)

        step += 1

    steps += f"\nFinal total merge cost: {total\_cost}"

    return steps

# Streamlit UI

st.title("Optimal Merge Pattern (Heap-based)")

file\_sizes\_str = st.text\_input("Enter file sizes separated by spaces", "10 20 30 40 50")

compute\_button = st.button("Compute")

if compute\_button:

    try:

        file\_sizes = list(map(int, file\_sizes\_str.split()))

        if len(file\_sizes) < 2:

            st.error("Please enter at least two file sizes.")

        else:

            result = optimal\_merge\_pattern(file\_sizes)

            st.write(result)

    except ValueError:

        st.error("Please enter valid integers separated by spaces.")

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

# Git Hub

GitHub is a web-based platform that provides version control, collaboration, and project management tools for software development. It allows developers and teams to host, manage, and track changes to their code repositories, making it easier to work on projects collaboratively, keep track of code revisions, and coordinate development efforts. GitHub is widely used for version control using Git, facilitating collaboration among developers, and hosting open-source and private software projects. It offers features like code hosting, issue tracking, pull requests, code review, continuous integration, and more, making it a central hub for software development and collaboration.

### Here are some key aspects of GitHub:

**Version Control**: GitHub is primarily known for its version control capabilities, particularly with the Git version control system. Git is a distributed version control system that allows developers to track changes in their codebase, collaborate with others, and manage different versions of their software projects. GitHub provides a hosted environment for Git repositories.

**Code Hosting**: Developers can create repositories on GitHub to store and manage their source code. These repositories can be public (visible to anyone) or private (restricted access).

**Documentation**: GitHub provides tools for documenting projects, including wikis and README files. These help developers create documentation that accompanies their code, making it easier for others to understand and use their software.

**Package Management**: GitHub offers a package registry where developers can publish and share packages, libraries, and dependencies. This makes it convenient to manage and distribute software components.

**Community and Support**: GitHub has a large and active community of developers, which can be helpful for seeking advice, getting answers to technical questions, and finding contributors for projects.

**Continuous Integration and Deployment (CI/CD)**: GitHub integrates with various CI/CD tools, allowing developers to automate testing and deployment processes. Popular CI/CD services like GitHub Actions can be used to build, test, and deploy code directly from GitHub repositories.

GitHub is widely used by individual developers, open-source projects, and organizations of all sizes for managing software development projects. Its features and collaboration capabilities have made it an essential platform in the world of software development and open-source contributions.

Github link for our project :

**Output:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

# Conclusion

In conclusion, the development of an **Optimal Merge Pattern** demonstrates significant advancements in efficiently managing file merging processes using the optimal merge algorithm. This project highlights the effectiveness of algorithmic techniques in minimizing merge costs while providing a user-friendly interface for inputting file sizes and visualizing merging steps.

The "Modern Phone Book Application" represents a pivotal step forward in the realm of contact management, offering a sophisticated and user-centric solution for individuals and organizations alike. This project acknowledges the evolving landscape of digital communication and adapts to the needs of the modern world, providing a range of powerful features that redefine how we manage, access, and share our contact information.

**Key Takeaways**

1. **Efficiency in Contact Management**: The application simplifies and streamlines the process of managing contact information, ensuring that users can effortlessly organize, update, and retrieve their contacts.

2. **User-Centric Design**: With an intuitive and user-friendly interface, the application caters to users of all technological backgrounds, making it accessible and convenient for a diverse audience.

3. **Powerful Search Capabilities**: The robust search feature allows users to quickly locate specific contacts using various criteria, saving time and effort in finding the right information.

4. **Security and Privacy**: Stringent security measures, including encryption and access control, protect users' sensitive contact data, ensuring their privacy and data integrity.

5. **Cross-Platform Accessibility**: The application's compatibility across web, mobile devices, and desktops ensures that users can access their contacts whenever and wherever they need them.

6. **Collaboration and Sharing**: The ability to share contact information seamlessly fosters networking, collaboration, and efficient communication among users, whether for personal or professional purposes.

7. **Reminders and Notifications**: Users can set reminders for important events, helping them stay organized and on top of crucial tasks related to their contacts.

8. **Data Backup and Recovery**: The application includes automated backup options and recovery mechanisms, safeguarding users against data loss and offering peace of mind.



By hosting the project on GitHub, we emphasize collaboration and version control, inviting developers to contribute to its ongoing improvement. This initiative illustrates how modern development practices can address real-world challenges in file management.