Minimizing Average Completion Time of Tasks

1. Greedy Algorithm

To minimize the average completion time of all tasks, the optimal strategy is to process tasks in the order of their shortest processing times first. This is known as the **Shortest Processing Time (SPT)** rule.

Algorithm Steps

1. **Input**: A list of tasks with their processing times $\{(a_i, p_i)\}$ for $i = 1, 2, \ldots, n$.
2. **Sort**: Arrange the tasks in ascending order based on their processing times p_i.
3. **Schedule**: Process the tasks in the sorted order.
4. **Output**: The ordered list of tasks that minimizes the average completion time.

2. Proof of Optimality

We need to prove that scheduling tasks in non-decreasing order of their processing times minimizes the average completion time.

Detailed Proof

Let there be n tasks, and let σ be any schedule of these tasks. Let C_i denote the completion time of task i in schedule σ, and p_i its processing time.

Our objective is to minimize the total completion time:

$$T = \sum_{i=1}^{n} C_i$$

Exchange Argument

Suppose there exists an optimal schedule σ that is not in SPT order. Then, there must be at least one pair of adjacent tasks where a longer task precedes a shorter one.

Let tasks i and j be such a pair, with $p_i > p_j$, and task i scheduled immediately before task j in σ.

We will show that swapping tasks i and j results in a schedule with a lower total completion time.

Calculation Before Swap

Let S_i be the start time of task i. Then:

$$\begin{aligned} C_i &= S_i + p_i \\ C_j &= C_i + p_j = S_i + p_i + p_j \end{aligned}$$

Calculation After Swap

After swapping tasks i and j:

$$\begin{aligned}
C'_j &= S_i + p_j \\
C'_i &= C'_j + p_i = S_i + p_j + p_i
\end{aligned}$$

Change in Total Completion Time

The difference in total completion time due to the swap is:

$$\begin{aligned}
\Delta T &= (C'_i + C'_j) - (C_i + C_j) \\
&= [(S_i + p_j + p_i) + (S_i + p_j)] - [(S_i + p_i) + (S_i + p_i + p_j)] \\
&= [2S_i + 2p_j + p_i] - [2S_i + 2p_i + p_j] \\
&= (2p_j + p_i) - (2p_i + p_j) \\
&= (2p_j - p_j) - (2p_i - p_i) \\
&= p_j - p_i
\end{aligned}$$

Since $p_j < p_i$, the difference $\Delta T = p_j - p_i < 0$. Therefore, swapping tasks i and j reduces the total completion time.

Iterative Improvement

By repeatedly applying this exchange to all such inversions (where a longer task precedes a shorter one), we can reduce the total completion time until there are no inversions left—that is, the tasks are in SPT order.

Final Conclusion

Therefore, scheduling tasks in order of non-decreasing processing times minimizes the total completion time and, consequently, the average completion time, since:

$$\text{Average Completion Time} = \frac{1}{n}\sum_{i=1}^{n} C_i = \frac{T}{n}$$

3. Time Complexity Analysis

The algorithm involves sorting the list of tasks based on their processing times and then calculating the completion times.

Sorting Complexity

Using **Merge Sort**, which has a recurrence relation:

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

where:

- $T(n)$ is the time to sort n elements.
- $2T\left(\frac{n}{2}\right)$ is the time to sort the two halves.
- n is the time to merge the two sorted halves.

Solving the Recurrence Relation

We can solve the recurrence relation using the method of repeated substitution.

Assuming n is a power of 2 for simplicity:

$$\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n \\
&= 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n \\
&= 4T\left(\frac{n}{4}\right) + n + n \\
&= 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n \\
&= 8T\left(\frac{n}{8}\right) + n + n + n \\
&\vdots \\
&= 2^k T\left(\frac{n}{2^k}\right) + kn
\end{aligned}$$

When $k = \log_2 n$, $\frac{n}{2^k} = 1$, so $T(1)$ is a constant c.

Thus:

$$T(n) = nT(1) + n\log_2 n = nc + n\log_2 n$$

Since $T(1)$ is a constant, we can write:

$$T(n) = n\log_2 n + kn$$

where k is a constant representing $T(1)$.

Overall Time Complexity

After sorting, we need to compute the completion times, which requires traversing the list once and performing constant-time operations per task.

The time required is proportional to n.

Therefore, the total time complexity $T_{\text{total}}(n)$ is:

$$T_{\text{total}}(n) = n\log_2 n + kn + n = n\log_2 n + (k+1)n$$

This expression shows that the total time required by the algorithm increases proportionally to $n\log_2 n$ for large n.

4. Implementation and Testing

Explanation of the Algorithm

Code for the implementation and output text file can be found here. The algorithm schedules tasks in ascending order of their processing times to ensure the minimum average completion time. The steps are:

1. **Sort** the list of tasks based on their processing times.
2. **Calculate** the completion times for each task by accumulating the processing times.
3. **Compute** the average completion time by summing the completion times and dividing by the number of tasks.

Sample Input and Expected Output

For example, consider two tasks:

- Task a_1 with $p_1 = 5$
- Task a_2 with $p_2 = 3$

Scheduling in SPT Order:

- Schedule a_2 first, then a_1.
- Completion times:

$$\begin{aligned} C_2 &= p_2 = 3 \\ C_1 &= C_2 + p_1 = 3 + 5 = 8 \end{aligned}$$

- Average completion time:

$$\frac{C_1 + C_2}{2} = \frac{8+3}{2} = 5.5$$

Additional Test Case

Consider four tasks:

- Task a_1 with $p_1 = 2$
- Task a_2 with $p_2 = 1$
- Task a_3 with $p_3 = 4$
- Task a_4 with $p_4 = 3$

Scheduling in SPT Order:

- Sorted tasks: a_2, a_1, a_4, a_3
- Completion times:

$$\begin{aligned} C_2 &= p_2 = 1 \\ C_1 &= C_2 + p_1 = 1 + 2 = 3 \\ C_4 &= C_1 + p_4 = 3 + 3 = 6 \\ C_3 &= C_4 + p_3 = 6 + 4 = 10 \end{aligned}$$

- Average completion time:

$$\frac{C_2 + C_1 + C_4 + C_3}{4} = \frac{1+3+6+10}{4} = \frac{20}{4} = 5.0$$

Conclusion

The greedy algorithm schedules tasks in ascending order of their processing times, ensuring the minimum average completion time. The detailed proof confirms the optimality of the SPT rule. The time complexity analysis shows that the algorithm operates in time proportional to $n \log_2 n$ due to the sorting step.