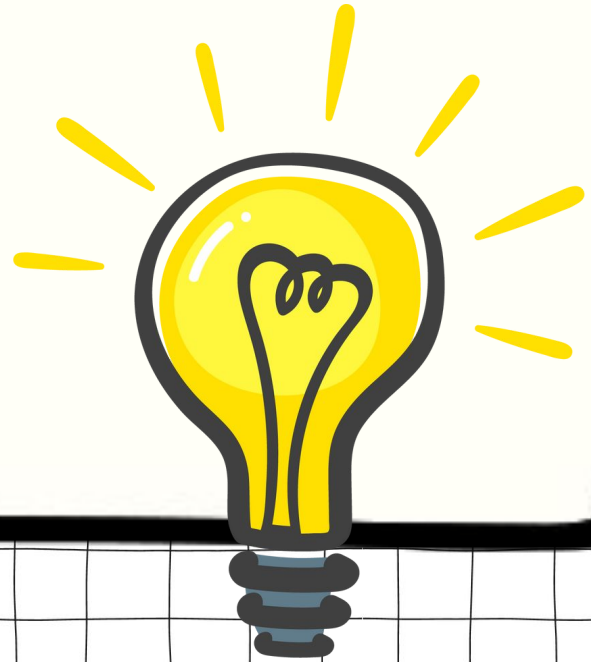


Min Coin Change

Classical Optimization Problem



Contents



01

Introduction

02

Applications

03

Approach to solve

04

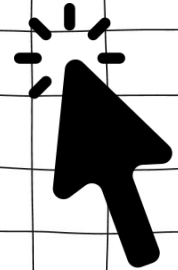
Tabular Solution

05

Parallel
Implementation

06

Results





Introduction

The minimum coin change problem is a classic problem in computer science and mathematics that involves finding the fewest number of coins needed to make change for a given amount of money.

Applications

Vending Machines:

To give the least amount of change when a customer inserts money.

ATM Withdrawals:

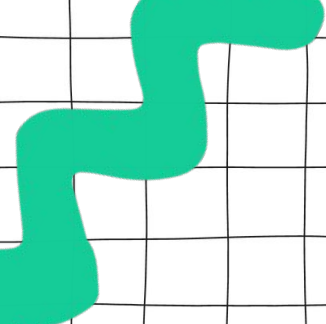
To dispense cash in an efficient manner, which includes giving out the least number of coins or bills to customers.

Financial Applications

In finance, optimizing cash flow and minimizing the number of transactions can be crucial.


Optimization Problems

its principles can also be applied in various optimization contexts, including logistics and scheduling.



Solution

For each of the available coins we have 2 options



Pick the coin



Optimization Function $\text{Min}(\text{Pick}, \text{Leave})$

Leave the coin



Heart of Dynamic Programming

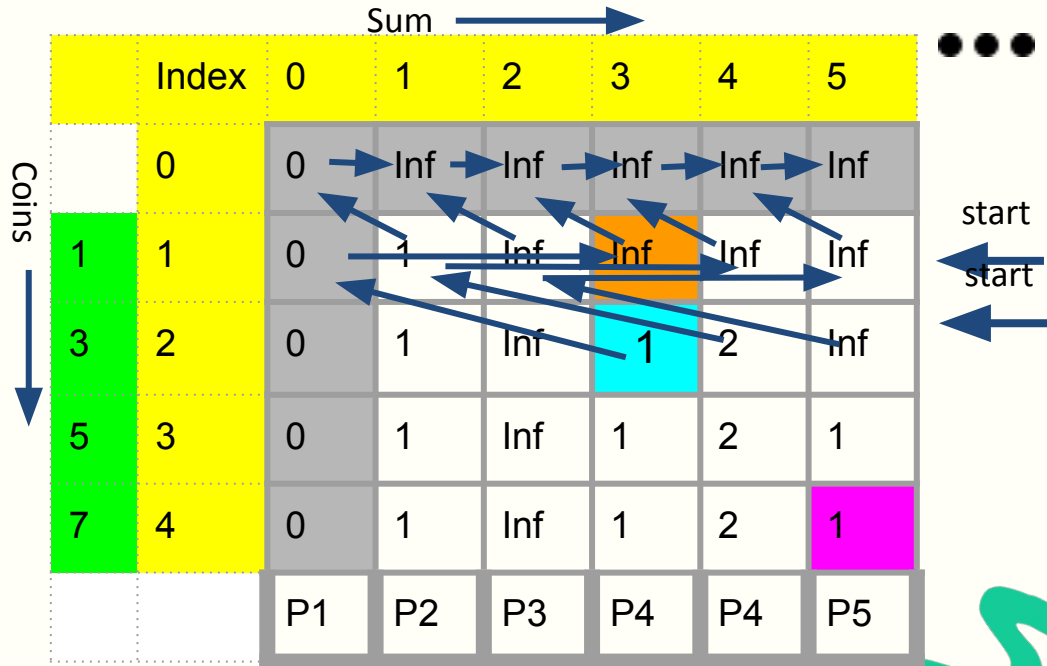
Input: [1,3,5,7], **Amount:** 5
Output:
Min number of coins or
-1 if that amount can not be
made

		Sum →					
		0	1	2	3	4	5
Coins ↓	0	0	Inf	Inf	Inf	Inf	Inf
	1	0	1	Inf	Inf	Inf	Inf
	3	0	1	Inf	1	2	Inf
	5	0	1	Inf	1	2	1
	7	0	1	Inf	1	2	1

- Each cell in the table stores the minimum number of coins

```
if(coins[i-1]<=j)
    ans[i][j] = Math.min(ans[i-1][j-coins[i-1]]+1, ans[i-1][j]);
else
    ans[i][j] = ans[i-1][j];
```

Multiple Processors with Single Column



- To Processor** - $\text{rank} + \text{denominations}[i-1]$
- Value to be send** - $\text{ansPerProcessor}[i-1][0]$
- From Processor** - $\text{rank} - \text{denominations}[i-1]$

Multiple Processors
with
Multiple Columns

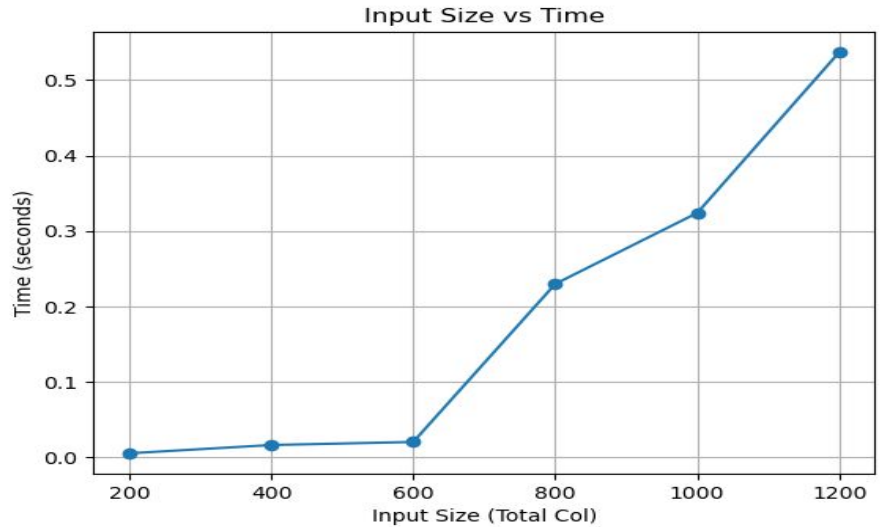
Sum →

		index	0	1	2	3	4	5	...
Coins ↓	0	0	Inf	Inf	Inf	Inf	Inf	Inf	
	1	1	0	1	Inf	Inf	Inf	Inf	
	3	2	0	1	Inf	1	2	Inf	
	5	3	0	1	Inf	1	2	1	
	7	4	0	1	Inf	1	2	1	
			P1		P2		P3		

To Processor with rank - $(rank * COLS + j + coins[i - 1]) / COLS$
 Sending Tag - $rank * COLS + j$
 To receive from - $(rank * COLS + j - coins[i - 1]) / COLS$
 Receiving tag - $rank * COLS + j - coins[i - 1]$

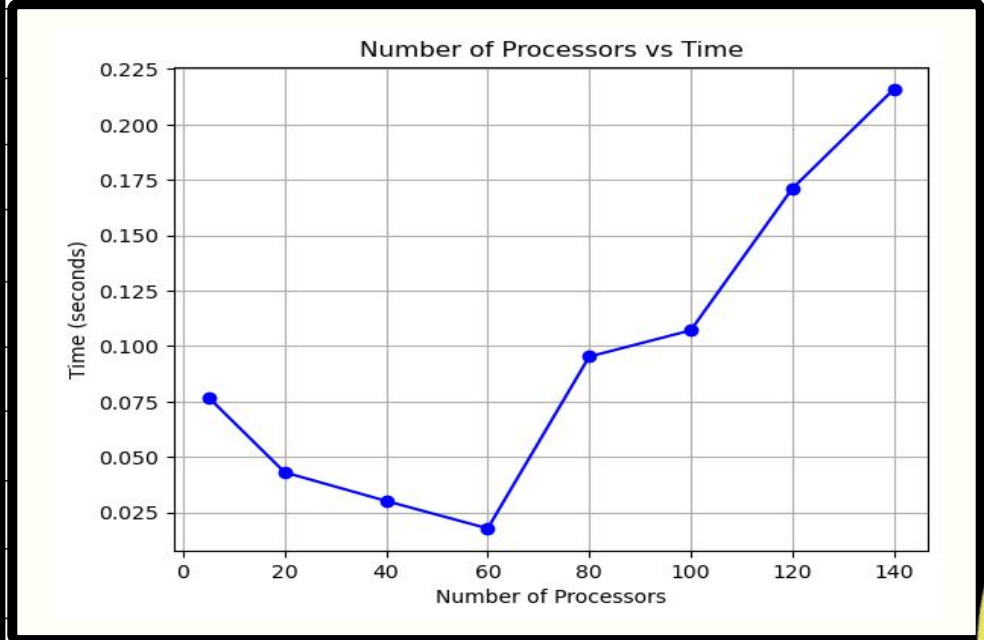
Sequential execution Time

PEs	Input Size (Total Col)	Time (in seconds)
1	200	0.005372
1	400	0.016355
1	600	0.020413
1	800	0.22985
1	1000	0.323971
1	1200	0.537114



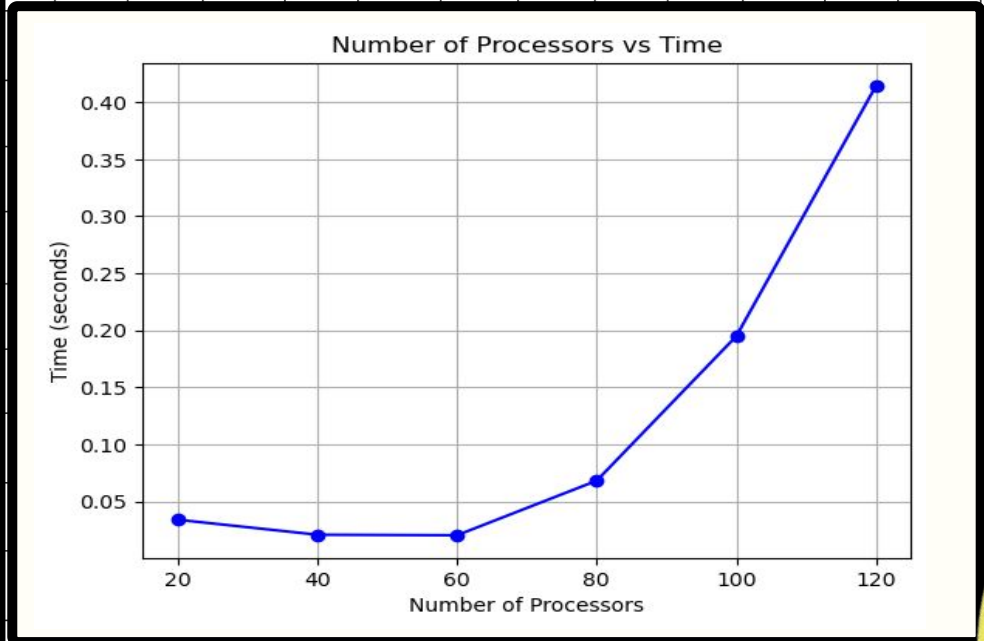
Amdahl's Law

processors	Data per processor	Input Size (Total Col)	Nodes, core	Time (in seconds)
5	500	2000	5, 1	0.076854
20	100	2000	20, 1	0.043044
40	50	2000	40, 1	0.030136
60	34	2000	60, 1	0.017746
80	25	2000	80, 1	0.095246
100	20	2000	100, 1	0.107106
120	17	2000	120, 1	0.171037
140	15	2000	140, 1	0.215846



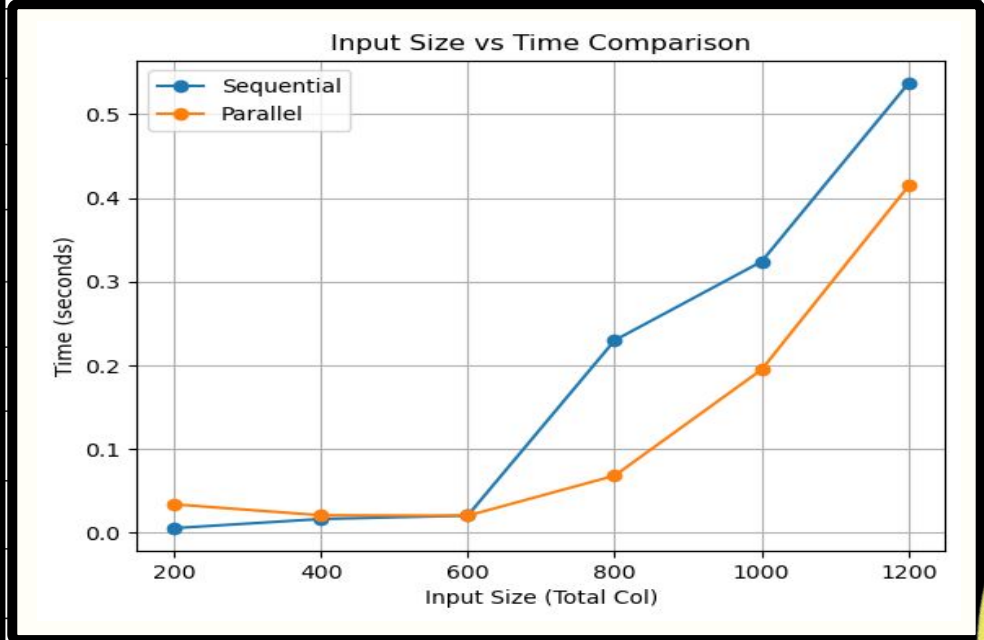
Data Per Processor Constant

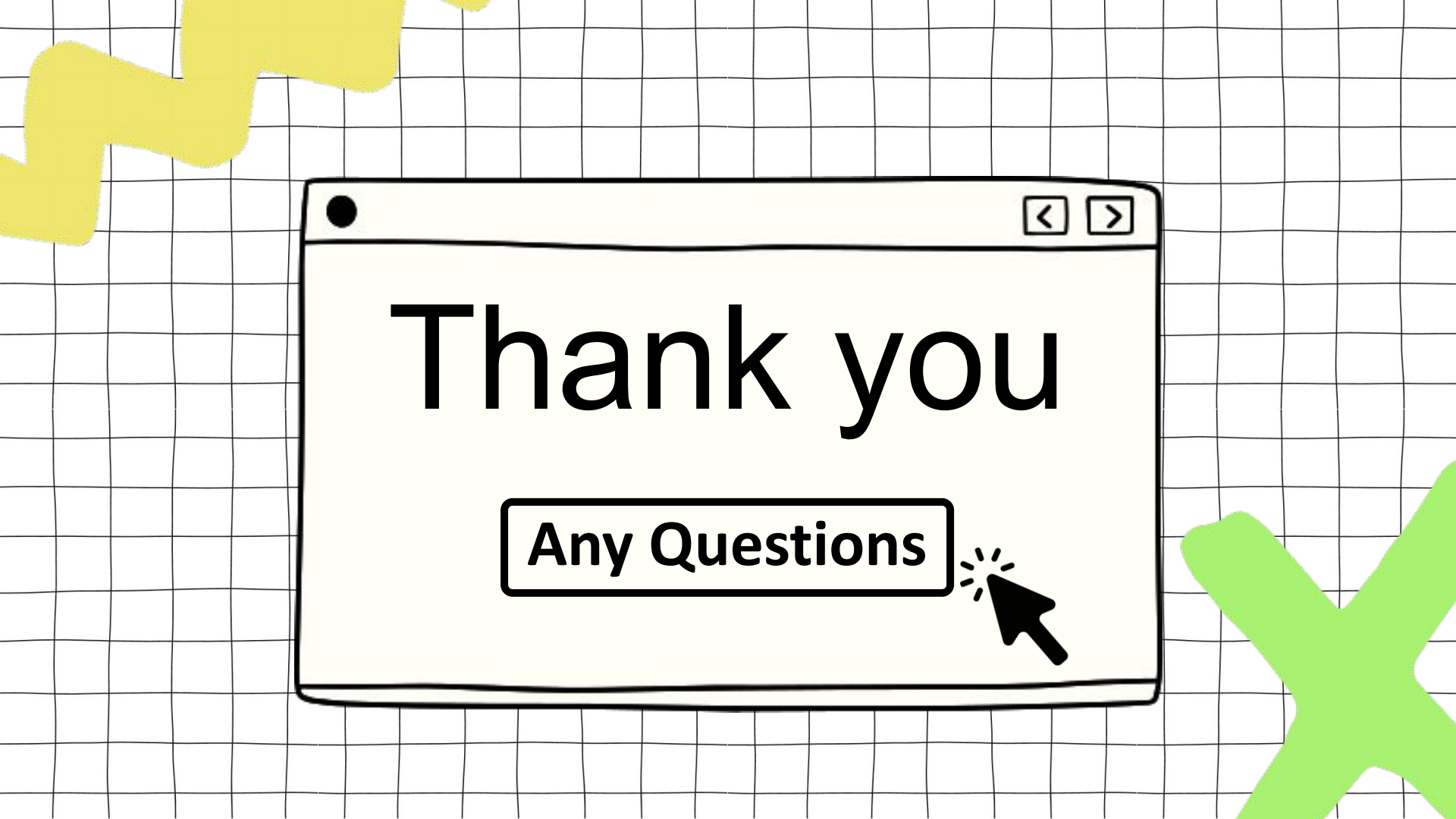
processors	Data per processor	Input Size (Total Col)	Nodes, cores	Time (in seconds)
20	10	200	20, 1	0.033918
40	10	400	40, 1	0.020783
60	10	600	60, 1	0.020312
80	10	800	80, 1	0.068161
100	10	1000	100, 1	0.194891
120	10	1200	120, 1	0.41438



Gustafson's Law

Data per processor	Input Size (Total Col)	Time (in seconds) serial	Nodes, cores	Time (in seconds) parallel
10	200	0.005372	20, 1	0.033918
10	400	0.016355	40, 1	0.020783
10	600	0.020413	60, 1	0.020312
10	800	0.22985	80, 1	0.068161
10	1000	0.323971	100, 1	0.194891
10	1200	0.537114	120, 1	0.41438





Thank you

Any Questions