

HW 2

Sahar Eitam 318283116

Q1

$$1) P=0.75, N=4$$

$$S = \frac{1}{(1 - 0.75) + \frac{0.75}{4}}$$

$$S = \frac{1}{0.25 + 0.1875}$$

$$S = \frac{1}{0.4357}$$

$$S \approx 2.2857$$

$$2) P=0.6, S=3$$

$$3 = \frac{1}{(1 - 0.6) + \frac{0.6}{N}}$$

$$3 = \frac{1}{0.4 + \frac{0.6}{N}}$$

$$1.2 + \frac{1.8}{N} = 1$$

$$\frac{1.8}{N} = -0.2$$

$$-9 = N$$

This is not possible since the result is negative, indicating a calculation error or an unreachable speedup for this proportion of parallel and sequential code.

$$3) P=0.9, N=32$$

$$S = \frac{1}{(1 - 0.9) + \frac{0.9}{32}}$$

$$S = \frac{1}{0.1 + 0.028125}$$

$$S \approx 7.8049$$

$$4) P=0.85, N = \infty$$

$$S = \frac{1}{(1 - 0.85) + \frac{0.85}{\infty}}$$

$$S = \frac{1}{0.15}$$

$$S \approx 6.6667$$

$$5) P=0.8, N = 4$$

$$S4 = \frac{1}{(1 - 0.8) + \frac{0.8}{4}}$$

$$S4 = \frac{1}{0.4}$$

$$S4 \approx 2.5$$

$$S32 = \frac{1}{(1 - 0.8) + \frac{0.8}{32}}$$

$$S4 = \frac{1}{0.225}$$

$$S32 \approx 4.4444$$

$$\Delta S = 4.4444 - 2.5$$

$$\Delta S = 4.4444 - 2.5 = 1.9444$$

Q2

Run	Process Time (ms)	Thread Time (ms)
1	13	1
2	15	2
3	16	1
4	14	2
5	15	3
6	14	1
7	17	3
8	15	2
9	14	2
10	15	1

Explanation: Each process operates in its own memory space and utilizes dedicated resources. Starting a process requires allocating these resources, which takes more time. Threads within the same process share the same memory space and resources, making them more lightweight. Creating a thread involves less overhead since it uses the resources of the parent process. Additionally,

context switching between processes is more costly, as it involves switching the entire memory space and resources. Context switching between threads is faster because they share the same memory space, reducing the overall overhead.

Q3

	Details								
	Name	Threads	PID	Status	User name	CPU	Memory (ac...	Architec...	Description
Processes	ijhi_service.exe	2	5144	Running	SYSTEM	00	4 K	x64	Intel(R) Dynamic ...
Performance	LocationNotification...	2	18268	Running	sahar	00	16 K	x64	LocationNotificat...
App history	LockApp.exe	18	14180	Suspended	sahar	00	0 K	x64	LockApp
Startup apps	lsass.exe	12	1048	Running	SYSTEM	00	7,004 K	x64	lsass
Users	MatrixMultiplier.cs.exe	38	20280	Running	sahar	83	71,468 K	x64	MatrixMultiplier.cs
Details	Microsoft.ServiceHub...	13	18984	Running	sahar	00	10,448 K	x64	Microsoft.Service...
Services	Microsoft.SharePoint...	31	1296	Running	sahar	00	11,040 K	x64	Microsoft ShareP...
	MoUsoCoreWorker.exe	3	9044	Running	SYSTEM	00	968 K	x64	MoUSO Core W...
	MpDefenderCoreSer...	8	5392	Running	SYSTEM	00	2,684 K		MpDefenderCor...
	MSBuild.exe	10	17328	Running	sahar	00	16,776 K	x64	MSBuild
	msedgewebview2.exe	20	22760	Running	sahar	00	33,380 K	x64	Microsoft Edge ...
	msedgewebview2.exe	21	23872	Running	sahar	00	5,716 K	x64	Microsoft Edge ...
	msedgewebview2.exe	7	5856	Running	sahar	00	2,988 K	x64	Microsoft Edge ...

The extra threads in your process are probably the result of overhead from the runtime environment, system operations, or external libraries used during execution.

Q4

A) Multithreading in the merge-sort algorithm allows sorting operations to run concurrently, which can improve performance, especially with large datasets.

1. **Recursive Division:** The input array is split into two halves. This division continues recursively until the size of each subarray is less than or equal to a predefined threshold (n_{Min}).
2. **Parallel Sorting:** If the size of the subarray is larger than the threshold, it is sorted concurrently by creating a new thread. This allows multiple subarrays to be sorted at the same time, utilizing multiple CPU cores for better efficiency.
3. **Thread Creation:** For each half of the array, a new thread is created to perform the sorting on that subarray. As a result, sorting happens in parallel for different parts of the array.
4. **Merging:** Once the subarrays are sorted, they are merged back into a single sorted array. The merge operation is a fundamental part of merge-sort, where two sorted arrays are combined into one sorted array.
5. **Threshold and Efficiency:** The threshold (n_{Min}) determines the minimum size of subarrays that will be sorted using multithreading. If the subarray size is less than or equal to this threshold, sorting is performed sequentially, without creating additional threads. This avoids the overhead of thread creation for small subarrays, which might not benefit much from parallel execution.

B)

