1 Problem 1

The runtime of mergesort is $\mathcal{O}(n \log n)$, and with $n = 2^k$, we get

$$O\left(2^k \log_2 2^k\right) = \mathcal{O}\left(2^k \cdot k\right)$$

Because we have that $\log_b b^x = x$.

Next, we are going to design a parallel mergesort algorithm using $p = \frac{n}{2}$ processes. The way we do this is to assign one process to every merge-operation of two lists. The algorithm starts by merging all lists of length one (there are $\frac{n}{2}$ such merge operations), before proceeding with merge operations of lists of length two and so on until the array is sorted. Since the number of merge operations is at a maximum $\left(\frac{n}{2}\right)$ at the start we will always have enough processes available. Derive the parallel runtime of this algorithm? What is the speedup and efficiency?

We have that merge step 0 takes time 1, step 1 takes time 2, step 2 takes time 4, and step 3 takes time 8 and so on. We have to do $\log n$ merge steps, so the sum of all merge steps can be expressed with

$$S = \sum_{i=0}^{\log n} 2^i$$

To solve this summation, we have the common ratio $r = 2$, which gives

$$2S = \sum_{i=0}^{\log n} 2^{i+1}$$

And when we substract S, we get

$$\begin{aligned} 2S - S &= 2 + 2^2 + 2^3 + \ldots + 2^{k+1} - (1 + 2 + 2^2 + \ldots + 2^k) \\ &= 2^{k+1} - 1 \\ &= 2^{\log n + 1} - 1 \\ &= n^{\log 2} \cdot 2^1 - 1 \\ &= 2n - 1 \end{aligned}$$

And so the runtime of parallelized mergesort is $\mathcal{O}(n)$.
The speedup is, given $t_s = n \log n, t_p = n, p = \frac{n}{2}$

$$\text{Speedup}(\frac{n}{2}) = \frac{n \log n}{n} = \log n$$

Which is nothing to write home about...

2 Problem 2

a) Benchmarking of a sequential program reveals that 95% of the execution time is spent inside functions that are amenable to parallelization. What is the maximum speedup we could expect from executing a parallel version of this program using 10 processes?

We have a program where 95% can be parallelized, and 5% cannot be parallelized, this gives

$$\text{Speedup} = \frac{t_s}{\frac{0.95t_s}{10} + 0.05t_s} = \frac{1}{0.095 + 0.05} = \frac{1}{0.145} \approx 6.9$$

b) For a problem size of interest, 6% of the operations of a parallel program are inside I/O functions that are executed on a single process. What is the minimum number of processes needed in order for the parallel program to exhibit a speedup of 10?

For this program, 94% can be parallelized, so we have

$$\begin{aligned} S(x) = 10 &\geq \frac{t_s}{\frac{0.94t_s}{x} + 0.06t_s} \\ 10 &\geq \frac{1}{\frac{0.94}{x} + 0.06} \\ 1 &\geq \frac{9.4}{x} + 0.6 \\ 0.4x &\geq 9.4 \\ x &\geq \frac{9.4}{0.4} \\ x &\geq 24 \end{aligned}$$

3 Problem 3

A sequential algorithm spends $\frac{1}{4}$ of its running time in an initial preprocessing phase, and $\frac{1}{4}$ is spent in a postprocessing phase. Neither the preprocessing nor the postprocessing phase can be parallelized. What is the maximum speedup attainable when moving to a parallel version of the algorithm?

The maximum possible speedup is given by

$$S(p) = \frac{1}{\frac{0.5}{p} + 0.5}$$

Where p is the amount of processes.

4 Problem 4

Let A,B,C and D be four programs that each take one minute to execute and that have to be run in sequence. Thus A must finish before B can start and so on. Assume that porting your programs to a parallel computer will give A a speedup of 100, B a speedup of 10, C a speedup of 1, and D a speedup of 0.1.

- What is the combined running time of the programs after porting all of them to the parallel computer?
- Which programs would you recommend to move to the parallel computer and what is the combined running time of your solution

After parallelizing the program, they will have the following execution time in seconds:

$$A = 0.6$$
$$B = 6$$
$$C = 60$$
$$D = 600$$

So the total execution time is $0.6 + 6 + 60 + 600 = 666.6$ seconds.

If we only parallelize A and B (and C if we want to do more work), we get a execution time of $0.6 + 6 + 60 + 60 = 126.6$ seconds.

$$T\left(w, h\right) = \bigwedge_{i=1}^{h} T(h, i)$$