Task 01:

The code for naive broadcast is quite straightforward.

- We generate random integers in processor 0 (`rank==0`);
- Send them to other processors one by one with sequential message transfer;
- The other processors receive the messgae sent from processor 0 (`else`).

```
if (rank == 0) {
  srand(time(NULL));

  for (int i = 0; i < n; ++i) {
    numbers[i] = rand();
  }

  for (int i = 1; i < size; ++i) {
    MPI_Send(numbers, n,MPI_INT32_T, i, 0, MPI_COMM_WORLD);
  }
} else {
  MPI_Recv(numbers, n, MPI_INT32_T, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

Then we compute the checksum by for-loop in each processor:

```
long long sum = 0;
for (int i = 0; i < n; ++i) {
  sum += numbers[i] & 7;
}
```

The wall-time when we need to inform 3 processors and 7 processors are given below, respectively:

# of processes:	**4**	**8**
n = 10:	0.0003	0.0005
n = 100:	0.0003	0.0007
n = 1.000:	0.0005	0.0008
n = 10.000:	0.0009	0.0018
n = 100.000:	0.0017	0.0031
n = 1.000.000:	0.0209	0.0317

The relative speed-down is approx. 2x, which is reasonable for an $O(p)$ broadcast algorithm.

Task 02:

To execute a binomial tree broadcast to get the data from process 0 to every other location at first every other process is ordered to wait to receive data. Since now process 0 is not the only thread sending out data the function `get_source` is used by every process to calculate which other process is expected to be sending the data to is using their rank. Afterward, a loop is started to send the data to the correct other processes according to the pseudocode provided in the lecture. \
The speedup of this order of operations can be observed in the runtimes measured for different values of n from 10 to 10.000.000 and using 4 and 8 processes respectively. It can be observed that the increase of the runtime from 4 to 8 processes is no longer with an approximate factor of 2 but much lower especially with larger values of n. Additionally, this broadcasting operation outperforms the previous naive implementation in each measured runtime.

Task 04:

We will divide a for-loop over the array into $n/(n/log(n)) = \log(n)$ parts for the $n/log(n)$ processors.

Define an array B to store the minimum index of first nonzero entry in the k-th part.

1. iterate over the array:

   ```
   for i from 0 to n dopar:
       if A[i] != 0:
           B[k] = i
           terminate for-loop for k-th processor
       if no nonzero entry in k-th part:
           B[k] = undefined
   ```

   n operations using $\frac{n}{\log(n)}$ processors for a runtime of $O(\log(n))$.
2. iterate over B to find the first non-undefined entry, which gives the answer. ($O(\log(n))$)

Task 05:

See next page.