

Graded out of **83 points**.

Five questions.

1. **Erlang** (14 points)

- (a) (**6 points**) Let V be list of numbers – V represents a vector. Let J be an index for an element of V ; in other words, J is an integer with $1 \leq J$ and $J \leq \text{length}(V)$. Complete the implementation of function $q(J, V)$ below that returns the square of the J^{th} element of V divided by the sum of the squares of all of the elements of V .

```

% q(J, V) ->  $\frac{(J^{\text{th}} \text{ element of } V)^2}{\sum_{I=1}^{\text{length}(V)} (I^{\text{th}} \text{ element of } V)^2}$ 
% examples:
%   q(2, [1, 7, 4, 2]) -> 7*7/(1*1 + 7*7 + 4*4 + 2*2)
%                       -> 49/(1 + 49 + 16 + 4)
%                       -> 49/70
%                       -> 0.7
q(J, V) -> % write your answer below
  X = lists:nth(J, V),
  (X*X) / lists:sum([Y*Y || Y <- V]).

```

Hint: To keep this simple for the exam, you don't need to add guards to check for invalid inputs.

- (b) (**6 points**) The function `lists:max(List)` in the Erlang standard library returns the largest element of `List`. Sometimes, we also want to know the position of this element. Complete the function `maxi` below:

```

% maxi(List) -> {Max, Index} where:
%   List is a list.
%   Max is the largest element of List
%   Index is the index of this largest element.
maxi([Hd | Tl]) -> maxi(Tl, {Hd, 1, 2}).
maxi([Hd | Tl], {Max, Index, CurPos}) when Hd > Max ->
  maxi(Tl, {Hd, CurPos, CurPos+1});

maxi([Hd | Tl], {Max, Index, CurPos}) when Hd <= Max ->
  maxi(Tl, {Max, Index, CurPos+1});

maxi([], {Max, Index, CurPos}) -> % Should change CurPos to _CurPos
  {Max, Index}.                  % to avoid compiler warning.

```

- (c) (**2 points**) Is the `maxi` function that you completed above head-recursive or tail-recursive?

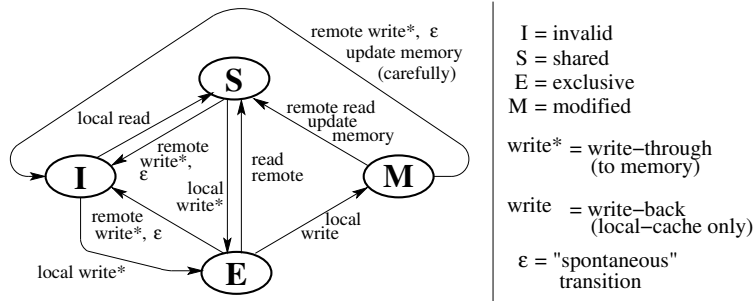
head-recursive tail-recursive

2. **Architecture** (14 points)

- (a) (**10 points**) Figure 1 shows the state-transition diagram for the MESI protocol. **Complete the table.**

The table shows a sequence of reads and writes by two processors, P0 and P1. All reads and writes in the table are to the same address. Each row of the table shows the action taken, the value in main memory, and the value and state of the cache line in each cache. The notation “E3” means the cache line is in the E state and has the value 3. If the action is a write, the value that was written is given in the action. For example, “P0: write 5” means that processor P0 writes the value 5.

Note that the last line shows the value in Memory and the state of Cache 0. For that line, you need to figure out what processor made what action and what the state is for Cache 1 (and the value if the state is not I).



Action	Memory	Cache 0	Cache 1
	?	I	I
P0: write 3	3	E3	I
P0: write 5	3	M5	I
P1: read	5	S5	S5
P0: read	5	S5	S5
P1: write 42	42	I	E42

Figure 1: MESI example for Question 2a

- (b) (2 points) I'm planning to build a message-passing parallel computer with $2^{21} \approx 2$ million processors. I'm trying to decide whether to use a 3D-torus architecture or a hypercube for the interconnect between the processors. State **one** advantage of using a hypercube architecture. Answers which state more than one advantage will receive 0.

Solution: Small (i.e. $O(\log N)$) worst-case latency

Other acceptable answer include: high bisection bandwidth, easily space shared, communication structure matches divide-and-conquer algorithms such as bitonic sort, more flexible routing avoids congestion.

- (c) (2 points) For the same parallel computer with 2^{21} processors, state **one** advantage of using a 3D-torus architecture. Answers which state more than one advantage will receive 0.

Solution: Interconnect is simpler and takes less space.

Other acceptable answer include: more volume efficient; shorter wires save energy and make lower latency between neighbours; bounded degree nodes are more modular thus easier to implement.

3. Performance (19 points)

Figure 2 shows a directed, acyclic graph for the work-span model. Vertices represent tasks, assume that all tasks execute in unit time. Edges represent dependencies – a task can execute once all of the tasks for its incoming edges have completed.

- (a) (2 points) Circle the task that must be performed first, and write "first" next to the circle.
 (b) (2 points) Circle the task that must be performed last, and write "last" next to the circle.

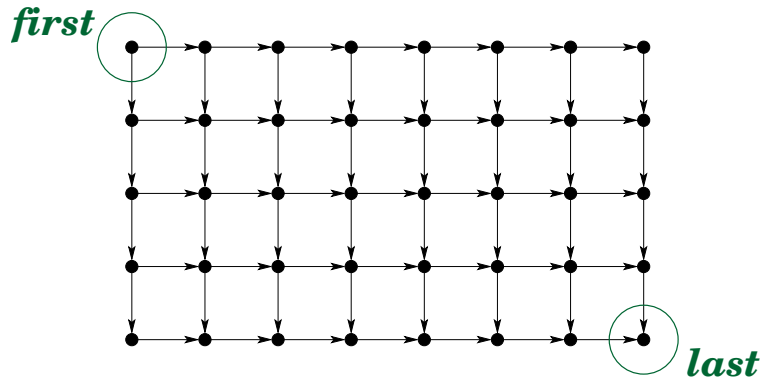


Figure 2: Work-Span Graph for Question 3

- (c) (2 points) What is the “work” for the computation depicted by the graph in Figure 2? Your answer should be a positive integer.

Solution: The graph has five rows, eight columns, and one vertex for each (row, column) pair.
 $Work = 5 * 8 = 40.$

- (d) (2 points) What is the “span” for the computation depicted by the graph in Figure 2? Your answer should be a positive integer.

Solution: The span is the number of vertices on longest path from first to last. We can go from first, all the way to the right and then down to last. There are 8 vertices in the top row, and 5 vertices in the rightmost column. The upper-right vertex is in both. We conclude:
 $Span = 5 + 8 - 1 = 12.$

- (e) (3 points) What is the maximum possible speed-up for the computation depicted by the graph in Figure 2?

Solution:

$$MaximumSpeedUp = \frac{Work}{Span} = \frac{40}{12} = 3\frac{1}{3}$$

- (f) (2 points) Can the work-span model be used to derive upper bounds for speed-up (yes or no)?

yes no

- (g) (2 points) Can the work-span model be used to derive lower bounds for speed-up (yes or no)?

yes no

- (h) (2 points) Does the work-span model account for communication delays (yes or no)?

yes no

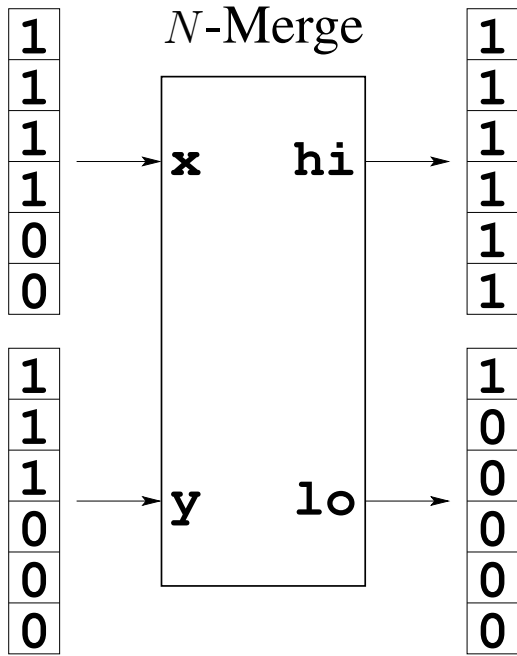
- (i) (2 points) Which of the following is most often associated with Gordon Moore (check one)?

Amdahl’s Law Denard Scaling the golden rule
 Gustafson’s Law Little’s formula Moore’s Law
 the work-span model

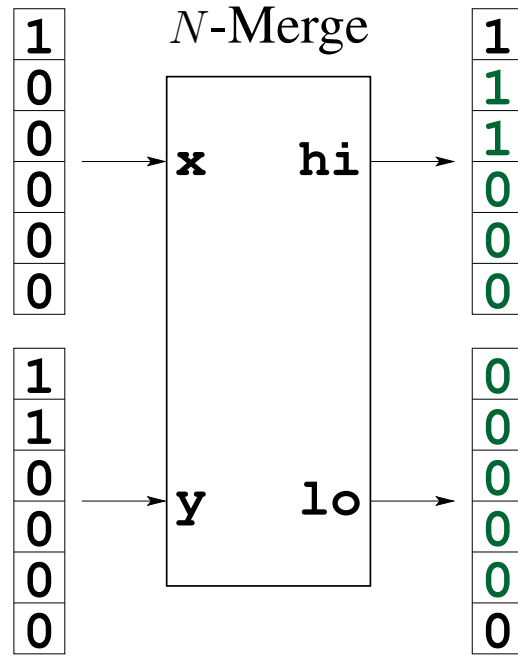
4. **Sorting** (16 points)

In class, Mark described a “ N -merge” that has inputs x and y and outputs lo and hi . Each of these is a sorted list (or array) of N elements. The merge element merges the lists x and y to produce a list, z of length $2N$. The smallest N elements of z are output as lo , and the greatest N elements are output as hi .

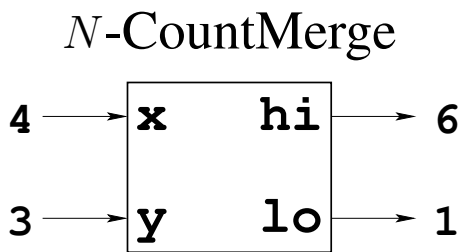
- (a) (2 points) How long does it take to perform a N -merge on lists of arbitrary integers (e.g. a C `int`) using a sequential computer such as an x86? Your answer should be something like $\mathcal{O}(1)$, $\mathcal{O}(\log N)$, $\mathcal{O}(N)$, or



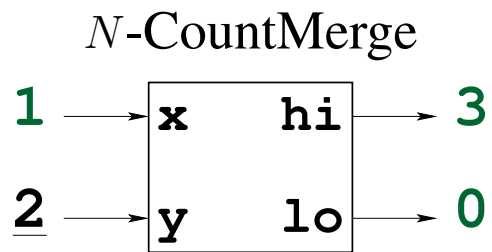
Example for Question 4b



Question 4b



Example for Question 4c



Question 4c

Figure 3: Merge Networks

$\mathcal{O}(N^2)$.

Solution: $\mathcal{O}(N)$

- (b) **(3 points)** Fill in the upper right part of the Figure 3 with the values for the lists `lo` and `hi`. We provided the upper-left part of the figure as an example. To get you started, we've provided the values for the smallest value of `lo` and the largest value of `hi`.

For questions 4c and 4d we consider N -merge elements where the inputs and outputs are lists consisting only of 0s and 1s. With this assumption, we can represent each input or output of a N -merge with an integer k , where k is the number of 1s in the list. When we use integers instead of lists, we call the operation a N -CountMerge. The lower left part of Figure 3 shows the example from the upper left, where the lists are represented as integers.

- (c) **(3 points)**

Fill in the lower right part of Figure 3 with the integer values corresponding to the 6-Merge from the upper right part of the figure (i.e. from Question 4b). To get you started, we've provided the value for `y`.

- (d) **(8 points)** In class, we showed that the outputs of a N -CountMerge are given by:

$$\begin{aligned} \text{hi} &= \min(x + y, N) \\ \text{lo} &= \max((x + y) - N, 0) \end{aligned}$$

We say that an input or output of a N -CountMerge is “clean” if its value is either 0 or N . Conversely, the value is “dirty” if it is greater than 0 and less than N . Prove that for any inputs `x` and `y` to a N -CountMerge, at least one of the outputs, `lo` or `hi` must be clean. My proof has two cases. I've provided the first case; you need to complete the second case.

Proof:

case $x + y \leq N$:

Subtracting N from both sides, we get $(x + y) - N \leq 0$.

Thus, `lo` = $\max((x + y) - N, 0) = 0$.

For this case, `lo` is clean.

case $x + y > N$:

Solution: Because $x + y > N$, `hi` = $\min((x + y), N) = N$.

For this case, `hi` is clean.

5. Reduce (20 points)

Implement a parallel version of `maxi` using `wtree.reduce`. There are four parts to the problem:

- (a) **(2 points)** Choose a data structure to be returned by the leaf function and used by the combine and root functions. Your choices are:
- `Max`, the largest value in the segment of the list for the subtree rooted at this node.
 - `{Max, Index}`, where `Max` is the largest value in the segment, and `Index` is the index of the largest value.
 - `{Left, Max, Index, Right}`, where `Max` and `Index` are as defined for options i and ii; `Left` is the first element of the segment; and `Right` is the last element of the segment.
 - `{Max, Index, Length}`, where `Max` and `Index` are as defined for options i and ii, and `Length` is the length of the segment.
- (b) **(8 points)** Implement the function `maxi_leaf` in Figure 4 below. You may use the `maxi` function as specified in Question 1b.
- (c) **(8 points)** Implement the `maxi_combine` function in Figure 5 on page 6.
- (d) **(2 points)** Implement the `maxi_root` function in Figure 5 on page 6.

```

maxi_par(W, Key) ->
  wtree:reduce(W,
    fun(ProcState) -> maxi_leaf(workers:get(ProcState, Key)) end, % Leaf
    fun(Left, Right) -> maxi_combine(Left, Right) end, % Combine
    fun(RootTally) -> maxi_root(RootTally) end, % Root
  ).

% Question 5b (8 points)
% Complete the function maxi_leaf below.
% For simplicity, you may assume that List is a non-empty list.
maxi_leaf(List) ->
  {Max, Index} = maxi(List),
  {Max, Index, length(List)}.

```

Figure 4: Code for Question 5 (part 1)

```

% Question 5c (8 points)
% Write the function maxi_combine.
maxi_combine({LMax, LIndex, LLength}, {RMax, RIndex, RLength})
  when LMax >= RMax ->
  {LMax, LIndex, LLength+RLength};
maxi_combine({LMax, LIndex, LLength}, {RMax, RIndex, RLength}) ->
  {RMax, LLength+RIndex, LLength+RLength}.

% Question 5d (2 points)
% Write the function maxi_root.
maxi_root({Max, Index, _Length}) ->
  {Max, Index}.

```

Figure 5: Code for Question 5 (part 2)