Lecture 5 exercise solutions

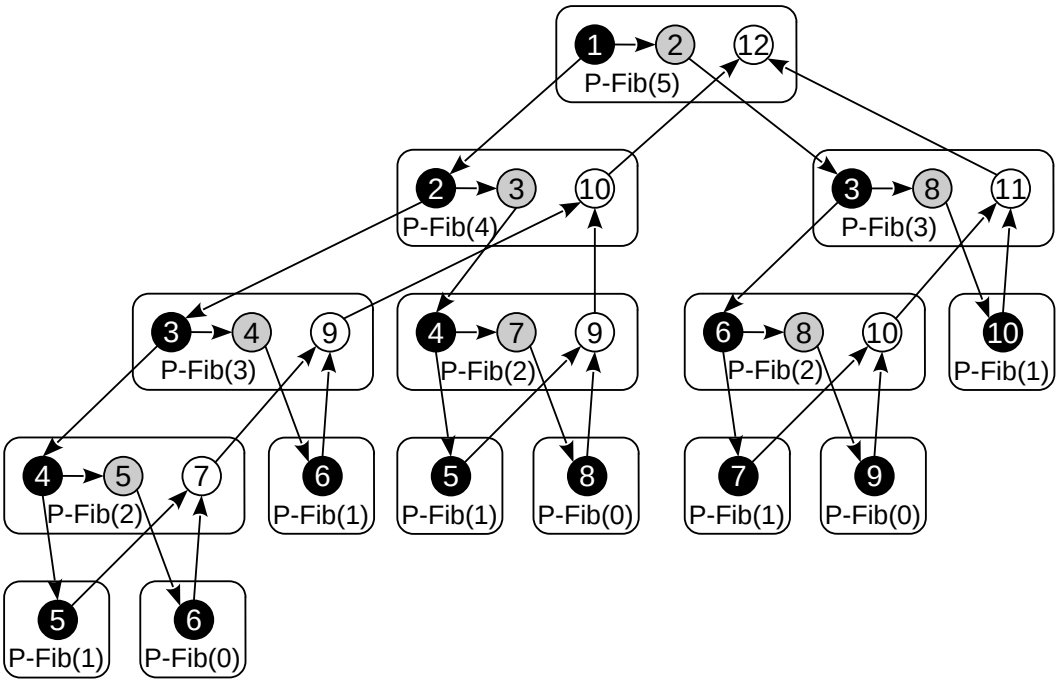
1.

The answer is **P=2.** When there is a perfect linear speedup, then T_1 / T_P = P, i.e., $T_P = T_1 / P$, but according to the law of span, $T_P \geq T_\infty$. In case of P =2, T_P = 17 / 2 = 8.5 $\geq$ 8, which means it is achievable. If P=3, prefect linear speedup would mean T_P = 17 / 3 = 5.66, which less than 8 and thus not achievable.

CLRS4 26.1-2 (CLRS3 27.1-1)

In this case, the parent thread is waiting idle for the other two threads to finish. The trace (computation DAG) would not change (except for the shading of rounded rectangles showing which are called and which are spawned – in this case, all would be spawned). The asymptotic work, span, and parallelism are not effected.

CLRS4 26.1-3 (CLRS3 27.1-2)



Each strand (vertex) is labeled with the time step in which it is executed. Note that a lot of other possible greedy schedules are possible. At each iteration, the scheduling algorithm simply picks arbitrary three of the waiting strands. A strand *x* can be scheduled, only if all strands with edges from them to *x* are completed.

Work =29; Span = 10; Parallelism = 29/10 = 2.9.

3.

This is one way to solve it (call **`LargestSpikeP`**`(A, 1, n))`:

```
LargestSpikeP(A, l, r)
01 if l == r then return -∞      // just one element – no pair to check
02 else
03   q = ceil((l+r)/2))      // round up
04   spawn ls1 = LargestSpikeP(A, l, q-1)
05         ls2 = LargestSpikeP(A, q, r)
06   sync
07   return max(A[q] – A[q-1], ls1, ls2)
```

Note that in the combining step we check the pair composed of the rightmost element of the left subproblem and the leftmost element of the right subproblem.
The work is defined by the recurrence: $W(n) = 2W(n/2) + \Theta(1)$ and the span by recurrence $S(n) = S(n/2) + \Theta(1)$. Thus, the work is $\Theta(n)$ and the span is $\Theta(\lg n)$, and parallelism is $\Theta(n / \lg n)$.

CLRS4 26.3-2 (CLRS3 27.3-2)

To solve this exercise correctly, it is very important to take care of details and formulate the assumptions precisely. Let the algorithm from Exercise CLRS4 9.3-10 (CLRS3 9.3-8) be called MEDIAN-OF-TWO(T, p_1, r_1, p_2, r_2) and let it get two ranges of sorted elements in the same array and return an index q (in one of the ranges) of the global median of the two sub-arrays. It is assumed that one of the two ranges, $[p_1..r_1]$ or $[p_2..r_2]$, may be empty.

Let $n = r_1 - p_1 + 1 + r_2 - p_2 + 1$ be the total number of elements in the two ranges. If the returned q is in $[p_1..r_1]$ then all elements in $T[p_2 .. p_2 + \lfloor n/2 \rfloor - (q - p_1 + 1) - 1]$ are smaller or equal than $T[q]$. If the returned q is in $[p_2..r_2]$ then all elements in $T[p_1 .. p_1 + \lfloor n/2 \rfloor - (q - p_2 + 1) - 1]$ are smaller or equal than $T[q]$.

Then the new version of parallel merging would look like this (merges sorted sequences $T[p_1..r_1]$ and $T[p_2..r_2]$ into $A[p_3 ...]$):

```
P-MERGE1(T, p1, r1, p2, r2, A, p3)
01 n = r1 - p1 + r2 - p2 + 2
02 if n > 0 then    // if at least one of the ranges is not empty
03    q = MEDIAN-OF-TWO(T, p1, r1, p2, r2)  //  0(lg n)
04    if q ≥ p1 and q ≤ r1 then
05       q2 = p2 + ⌊n/2⌋ – (q – p1 + 1)
06       q3 = p3 + (q – p1) + (q2 - p2)
07    else
08       q1 = p1 + ⌊n/2⌋ – (q – p2 + 1)
09       q3 = p3 + (q1 - p1) + (q – p2)
10    A[q3] = T[q]
11    if q ≥ p1 and q ≤ r1 then
12       spawn P-MERGE1(T, p1, q-1, p2, q2-1, A, p3)
13       P-MERGE1(T, q1+1, r1, q2, r2, A, q3+1)
14    else
15       spawn P-MERGE1(T, p1, q1-1, p2, q-1, A, p3)
16       P-MERGE1(T, q1, r1, q+1, r2, A, q3+1)
17    sync
```

The analysis of this algorithm is simpler than the analysis of P-MERGE (because the problem is always divided into two equal subproblems), but the results of the analysis are the same. The span can be described by the recurrence $S(n) = S(n/2) + \Theta(\lg n)$, which can be easily solved to $\Theta(\lg^2 n)$. The recurrence for work is also simple: $W(n) = 2W(n/2) + \Theta(\lg n)$. It is the first case of the master method, thus the solution is $\Theta(n)$.