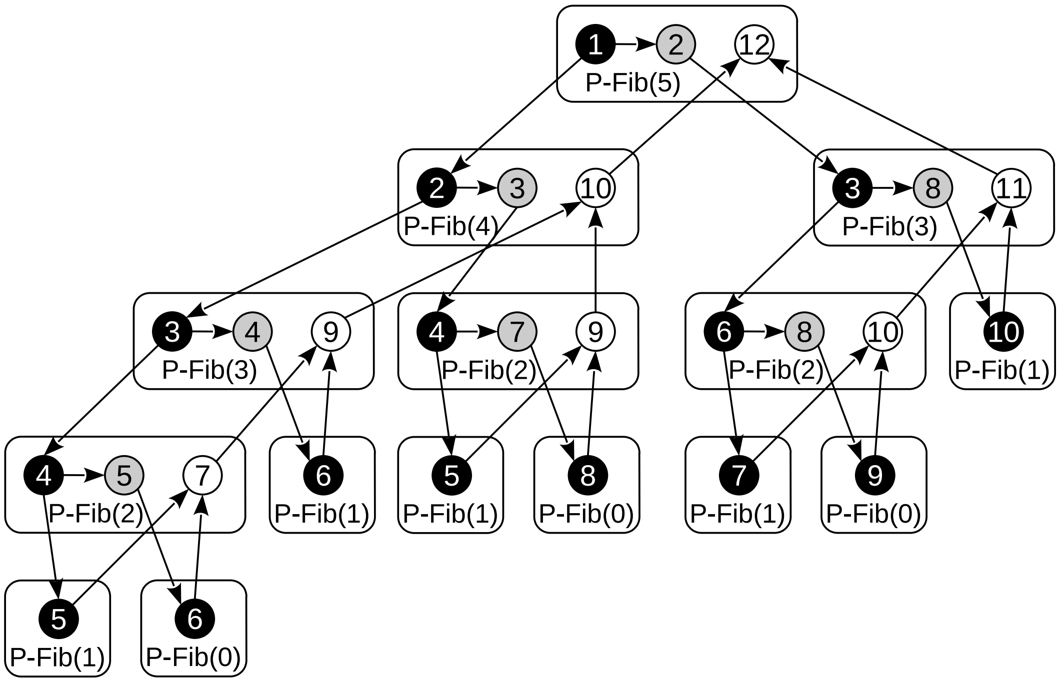
Lecture 5 exercise solutions

CLRS3 27.1-1.

In this case, the parent thread is waiting idle for the other two threads to finish. The asymptotic work, span, and parallelism are not effected.

CLRS3 27.1-2.



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.

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

2.

Call **LargestSpikeP**(A, 1, n).

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

Note that in the combining step we have to remember to 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 $\Theta(n / \lg n)$.

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 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 multithreaded merging would look like this:

```
P-MERGE1(T, p1, r1, p2, r2, A, p3)
n = r1 - p1 + r2 - p2 + 2
if n > 0 then    // if one of the ranges is not empty
    q = MEDIAN-OF-TWO(T, p1, r1, p2, r2)  //  Θ(lg n)
    if q ≥ p1 and q ≤ r1 then
        q2 = p2 + ⌊n/2⌋ - (q - p1 + 1)
        q3 = p3 + (q - p1) + (q2 - p2)
    else
        q1 = p1 + ⌊n/2⌋ - (q - p2 + 1)
        q3 = p3 + (q1 - p1) + (q - p2)
    A[q3] = T[q]
    if q ≥ p1 and q ≤ r1 then
        spawn P-MERGE1(T, p1, q-1, p2, q2-1, A, p3)
        P-MERGE1(T, q1+1, r1, q2, r2, A, q3+1)
    else
        spawn P-MERGE1(T, p1, q1-1, p2, q-1, A, p3)
        P-MERGE1(T, q1, r1, q+1, r2, A, q3+1)
    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 $\Theta(n)$.