1(a) Use Map to convert $A[i]$ to the pair $(A[i], A[i])$ and then Reduce with the operation *minmax* which takes two pairs of numbers (a, b) and (c, d) and produces the pair $(\min\{a, c\}, \max\{b, d\})$.

Or you could simply reduce with min and reduce with max.

1(b) Use Map to produce an array $B[]$ where

$$B[i] = \begin{cases} 1 & \text{if } (A[i] > A[i-1] \text{ or } i = 0) \text{ and } (A[i] > A[i+1] \text{ or } i = n-1) \\ 0 & \textit{otherwise.} \end{cases}$$

(You could overwrite $A[i]$ with 1 or 0 (rather than use the array $B[]$), but one thread might overwrite $A[i]$ before it is read by a different thread.) Use Reduce with the sum operator.

1(c) At a high level, we want a reduction operation that takes the longest increasing subarray (LIS) within $L = A[a_L, a_L + 1, \ldots, f_L]$ and the LIS within the adjacent $R = A[a_R, a_R + 1, \ldots, f_R]$ (adjacent means $a_R = f_R + 1$) and produces the LIS in their concatenation $L \circ R = A[a_L, a_L + 1, \ldots, f_R]$. This is tricky because the LIS of $L \circ R$ might start in L and end in R (but only if $A[f_L] < A[a_R]$). This suggests that we should keep track of not only the LIS in L and R but also the LIS in L that ends at f_L and the LIS in R that begins at a_R. Later, $L \circ R$ might be combined by Reduce with an adjacent subarray on the left or the right, so in general we better keep track of the LIS that begins at a_L and the LIS that ends at f_L in L (in addition to the LIS within L) and the same for R.

Let $(a_L, b_L, c_L, d_L, e_L, f_L)$ be a six-tuple that stores all this information for L. That is, $A[a_L \ldots b_L]$ is the LIS that begins at a_L, $A[e_L \ldots f_L]$ is the LIS that ends at f_L, and $A[c_L \ldots d_L]$ is the LIS in $A[a_L \ldots f_L]$.

The Reduce operation takes the six-tuples for L and the adjacent R and produces a new six-tuple (a, b, c, d, e, f) for $L \circ R$ using the following algorithm:

```
1. $a = a_L$, $b = b_L$, $c = c_L$, $d = d_L$, $e = e_R$, $f = f_R$
2. if $d_R - c_R > d - c$ then $c = c_R$, $d = d_R$
3. if $A[f_L] < A[a_R]$ then
4.     if $b_L = f_L$ then $b = b_R$
5.     if $e_R = a_R$ then $e = e_L$
6.     if $b_R - e_L > d - c$ then $c = e_L$, $d = b_R$
```

To start, use Map to convert $A[i]$ to the six-tuple (i, i, i, i, i, i) then perform Reduce using the above operation. After Reduce produces the six-tuple (a, b, c, d, e, f) for the entire array $A[]$, the LIS is $A[c \ldots d]$.

2. Use Pack with the predicate ($i = 0$ or $W[i] \neq W[i-1]$) where $W[i]$ is the ith word in the document.

3. Use Amdahl's Law for 100 processors:

$$\frac{T_1(n)}{T_P(n)} \leq \frac{1}{s + (1-s)/P} = \frac{1}{0.25 + (1 - 0.25)/100} = 400/103 = 3.88.$$

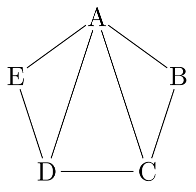
Even with an infinite number of processors, the speed up is at most $1/s = 4$.

4(a) Work is $\Theta(n)$ and span is $\Theta(1)$.

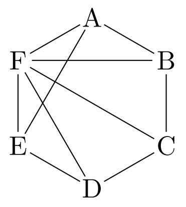
4(b) Work is $\Theta(n)$ and span is $\Theta(\log n)$.

4(c) Work is $\Theta(n^2)$ and span is $\Theta(n)$.

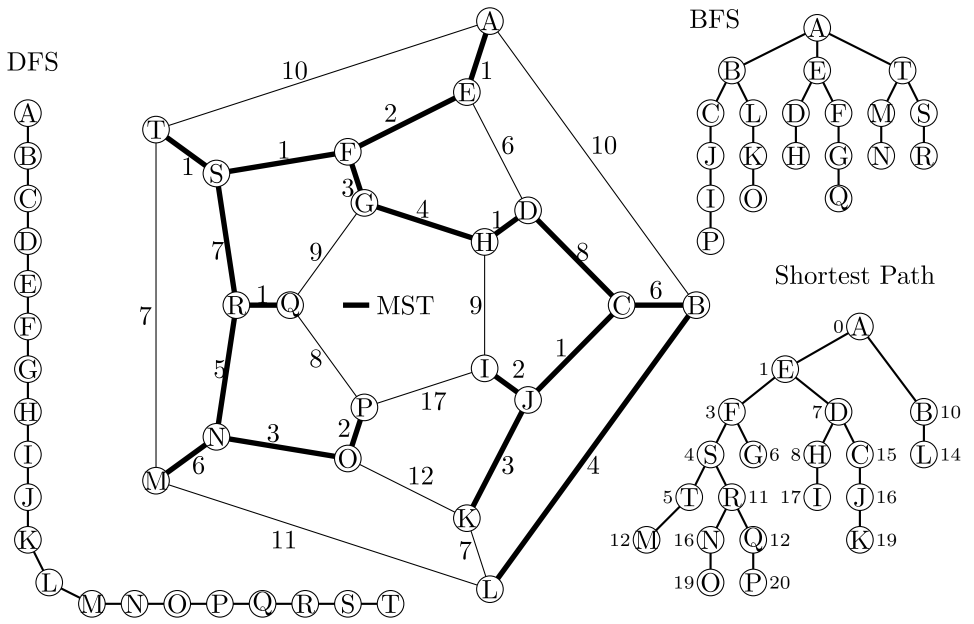
5.



6.



7.


DFS
BFS
Shortest Path
MST