Baruka's Algorithm

1 Algorithm

Baruka's algorithm will only work under the assumption that either all edge weights are distinct, or that there is some uniform tiebreaking rule.

In Kruskal's algorithm, we add edges to the MST by adding the cheapest edge that connects two trees together. On closer inspection, it becomes apparent that the lightest edge exiting *any* component is always safe, because it is guaranteed not to induce any cycles and always spans some cut of a viable solution.

```
initialize each vertex to be its own component
A = {}
while (there are 2 or more components):
   for each component C:
      find lightest (u, v) with u in C, v not in C
      add (u, v) to A (unless already there)

   DFS on (V, A) to compute the new components
```

2 Runtime

Each iteration of the loop is $\Theta(n+m)$ from the DFS. There are never more than $O(\log n)$ iterations needed. Take x to be the number of components at some stage. Each of the x components will merge with at least one other component. Afterwards, the worst case number of resulting components is $x/2$, if all components merge in pairs. Thus, the number of components decreases by at least half every iteration, so the algorithm is $\Theta(m \log n)$.

Note however that Baruka's algorithm is easily parallelizeable (given the uniform tiebreaking rule), unlike Prim's and Kruskal's algorithms.