1. I have used the dynamic programming approach to solve this problem.
2. Read a bridge and set of hiker info from yaml file. Compute min time for this bridge and hikers. Program will read information of next bridge and new set of hikers (if exists). They will join with hikers from previous bridges and will cross this new bridge and so on. The program will always use slower person’s (between two who crosses a bridge in a given time) speed to calculate time. For every bridge do following:
3. Sort the hikers of this bridge in ascending order of their speed.
4. Start with two Hikers with fastest speed
5. Always, a hiker with fastest speed will return with torch to help out other hikers who need help crossing. This will minimize the total cross time.
6. A pair with slowest speed will cross bridge to minimize total cross time.
7. Use step (d) and (e) above. Repeat until all hikers crossed this bridge.
8. For next bridge: Hikers from previous bridges will join new hikers (if exists) and will do steps (b) to (f) above and will continue to do so until all hikers from all bridges crossed.
9. Steps (c) to (f) is explained below with examples for 4 hikers named A, B, C, D. The time to cross the bridge once are 1, 2, 5, 10 respectively. These hikers will cross from left to right.  
     
   **Hikers(Start) Crossing ElapsedTime Hikers( End)**

A(1), B(2), C(5), D(10) None 0 None  
C(5), D(10) A(1), B(2) 2 A(1), B(2)  
A(1), C(5), D(10) A(1) 1 B(2)   
D(10) A(1), C(5) 5 A(1), B(2), C(5)

A(1), D(10) A(1) 1 B(2), C(5)

None A(1), D(10) 10 A(1), B(2), C(5), D(10)   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Total Elapsed Time = 19   
  
In the above example, it is seen that, the fastest person A(1) made the most trips back and forth. But the above is not the optimal solution which can be seen by following trip arrangement:

**Hikers(Start) Crossing ElapsedTime Hikers( End)**

A(1), B(2), C(5), D(10) None 0 None  
C(5), D(10) A(1), B(2) 2 A(1), B(2)  
A(1), C(5), D(10) A(1) 1 B(2)   
A(1) C(5), D(10) 10 B(2), C(5), D(10)

A(1), B(2) B(2) 2 C(5), D(10)

None A(1), B(2) 2 A(1), B(2), C(5), D(10)   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Total Elapsed Time = 17   
In this 2nd solution above, returned trips are swaped. The two fastest hikers (A, B) cross together on the 1st and 5th trips, the two slowest hikers (C,D) cross together on the 3rd trip, and either of the fastest hikers returns on the 2nd trip, and the other fastest hiker returns on the 4th trip.

1. **Architecture and design:** Please see UML diagram, file: *BridgeHikerArchAndDesignDiagram.pdf*. I’ll explain, how I have arranged yaml input file and related classes for the architecture and design. The yaml has following entries:  
   (a) **Tag** **BridgeAndHikers:** The corresponding class name is *BridgeAndHikerInfo*. This class will contain information of a bridge and list of Hikers who will cross this bridge. There can be any number of the BridgeAndHikers tag in the file. There two sub tags under this tag:  
    (i) **Bridge:** The corresponding class name for this is *BridgeInfo*. It has two entries: (A) **id**: I have used it for test and debugging purpose. In future, the program can be enhanced to use this id to print stat for a specific bridge. (B) **length:** This is length of the bridge. It used to calculate time taken by a hiker for one cross. For every bridge in yaml file, I calculate this “one cross time” for hikers involved for a given bridge. I have sort the array of hikers in ascending order using this time. The purpose of the sorting is described in step number 1 above.  
    (ii) **Hikers**: The corresponding class name for this is *HikerInfo*. There can any numbers of hikers under this tag. This sub tag is mandatory for first entry of tag BridgeAndHikers in yaml. If it is missing in first entry, the program will complain and exit. This tag is optional for remaining other tag BridgeAndHikers. The Hikers from previous bridges (which are already crossed) will use new bridge length to compute crossing time. There can be any number hikers under this sub tag. Each hiker will have two entries: (A) **-name**: name of hiker. I use this for test and debugging (B) **speed:** to indicate speed of a hiker.  
   (b) **Solution**: This is the driver class of the program. Object of this class will have list of BridgeAndHikerInfo. Each entry of BridgeAndHikerInfo will have a bridge and list of hikers who crossed this bridge.
2. **Complexity:** In worst case scenario, this program will take O((n^2)\*(2^n)) to compute minimum time for n number of hikers. The space complexity if O(2^n).
3. The code is tested with different sets of bridge and hikers. It is tested for invalid entry like no hikers as first entry in yaml file. At the end of processing all bridges, the program prints details statistics on the console for each bridge and summary of all bridges.
4. I have used object-oriented design and programming paradigm which I have explained in details in item# 2 above.