The worst case of the algorithm is that **all the projects are not fully funded until the last dollar of the last donation is spent.**

Let **n** be the number of donations, **m** the number of projects, **x** the total number of dollars worth of donations available, **y** the total number of dollars required to fund all the projects:

Every time canAllocateHelper is run,
- every project will be iterate and see if they are fully funded. So it takes O(m) times.
- Check if index i is at the end of the donations list. So it takes O(1) time.
- Get the donation takes O(1) time.
- Check if the project is fully funded for each available project takes O(1) time.

So **every time canAllocateHelper is run in the worst case** takes **O(m) times.**

For each donation, canAllocateHelper will be run a times where a is the total dollar has been spent to the available projects (Remember: canAllocateHelper will allocate 1 dollar at a time). So, canAllocateHelper will be run for every dollar in every donation.

So **the total number of time that canAllocateHelper will be ran in the worst case** is the sum of every dollar in every donation **$O(a_1 + a_2 + a_3 \ldots\ldots\ldots\ldots + a_n = x)$**.

In the worst case, canAllocateHelper will be run x times. So the **total running time** for the **worst case** is **O(m*x)**.

Example of the worst case:

P0 10

P1 10

P2 10

D0 5 {P0 P1 P2}

D1 10 {P0 P1 P2}

D2 5 {P0 P1 P2}

D2 10 {P0 P1 P2}

All the projects are not fully funded until the last dollar of the last donation is spent. So in this case, this situation in this algorithm will take O(m*x) where m = 3 and x = 30