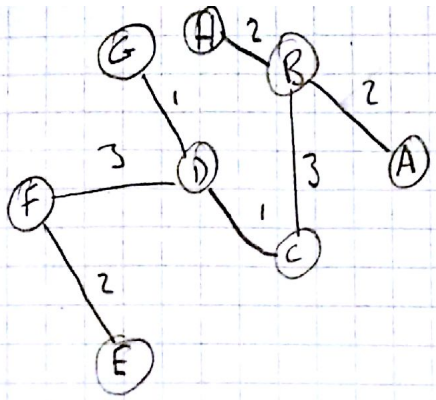
Ahmed Alaattara

1.



Order
1. B C
2. D G
3. B A
4. B H
5. F E
6. B C
7. F D

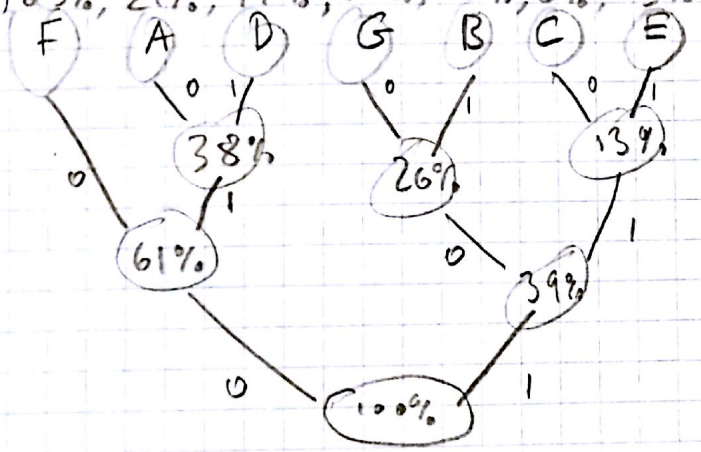
Pseudo code:

2. Maximize:
Sort (A) //ascending
sort (B) //ascending
return $\prod_{i=1}^{n} a_i^{b_i}$.

Since we've sorted the sets A & B, then our time is $O(n)$. This is optimal because any other pairing other than a_i with b_i will result in values less than 1. e.g.

if $a_1 > a_x$ & $b_1 > b_y$ then $a_1^{b_1} > a_x^{b_y}$ and comparing the two we can see $a_x^{b_y}$ gives us the less optimal solution.

3. a) Sort
23%, 21%, 17%, 15%, 11%, 8%, 5%
F A D G B C E

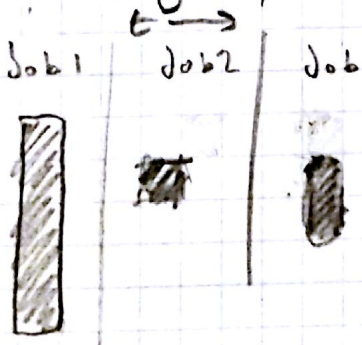


b)

A	0 1 0	3
B	1 0 1	3
C	0 1 1	3
D	1 1 0	3
E	1 1 1	3
F	0 0	2
G	0 0 1	3

c) 3 regardless (3 to represent it we ~~need~~ have 3 encoding bits).

4. a) Earliest ~~start~~ time: Not optimal, earliest starting job can block other jobs from running (especially if it runs for a long time)



e.g. Scheduling job 1 first increases wait time for job 2 & 3 substantially, inefficient!

c) Shortest interval: Not optimal! Could miss jobs that run before the shortest job.

b) Earliest finish time: Optimal!

Minimizes wait times substantially, we can get the most jobs done per unit time with this algorithm