**Q.(1)**

**[I]**

Yes. The strategy to sort the intervals in decreasing order of their start times, will ENSURE finding the optimal solution.

Suppose we have a set S = {a1, a2, a3, …, an} of n proposed activities.

Each activity i has a start time si and a finish time fi, where si ≤ fi.

Assume that the activities are ordered by decreasing order of their start time

s1 ≥ s2 ≥ … ≥ sn.

How do we know that this strategy works?

Suppose {a5, a7, a8, a12, a15} is an optimal solution. Consider the following activity set {a1, a7, a8, a12, a15}. Is this set an optimal solution?

Yes

* + - Why?
    - Since a5 and a7 are part of an optimal solution, the end time of a7 must be before the start time of a5. Since the start time of a1 is later than that of the start time of a5, there will be no conflict between a1 and a7. Therefore, the activity set {a1, a7, a8, a12, a15} will also be an optimal solution.

**[II]**

**No.** The strategy to sort the intervals in decreasing order of their lengths, will NOT ENSURE finding the optimal solution.

Consider following example:

Algorithm will pick **1** Orange job, but we could have picked **3** Blue jobs.

**[III]**

**No.** The strategy to sort the intervals in increasing order of the reciprocal values of their lengths, will NOT ENSURE finding the optimal solution.

Note: As we know, as decreasing order of numbers is same as increasing order of their reciprocals.

Example: a>b then 1/a < 1/b

So again, it is same as sort the intervals in decreasing order of their lengths.

Consider following example:

Algorithm will pick **1** Orange job, but we could have picked **3** Blue jobs.

**Q.(2)**

**[I]**

**Algorithm-**

Sort jobs in increasing order of their run time and consider them in this order in the algorithm. Return this order

**Proof of correctness-**

Average waiting time of jobs will be minimized if the total time run by the jobs in the system is minimized.

Let I = (i1 i2 … in) be any permutation of the integers {1, 2, …, n}. If the jobs are ran in the order I, the total time passed in the system by all jobs is

T(I) = ti1 + (ti1+ ti2) + (ti1+ ti2+ ti3) + … + (ti1+ … + tin)

= nti1 + (n-1)ti2 + (n-2)ti3 + … + tin

= Σk=1 to n (n–k+1)tik

Suppose now that, I is such that we can find two integers a and b with a<b and tia > tib

New run order I’ is obtained from I by interchanging the run orders of a and b

T(I’) = (n-a+1)tib+ (n-b+1)tib+ Σk=1 to n, k≠a,b(n-k+1)tik

run order 1 2 … a … b … n

ran jobs i1 i2 ia ib in

(from I)

Diagram, timeline

Description automatically generatedrun duration

After exchange

of ia and ib

run duration

ran jobs i1 i2 ib ia in

(from I’)

Thus,

T(I) – T(I’) = (n-a+1) (tia- tib) + (n-b+1) (tib-tia)

= (b-a)(tia- tib)

> 0

We can therefore improve any schedule in which a job is run before any other job which requires less run time. The **only schedules that remain are those putting the jobs in non-decreasing order of run time**. **---------- (a)**

All such schedules are clearly equivalent, and therefore they are all optimal.

**[II]**

No. There can’t be multiple schedules which give minimum average waiting time if the time taken for each of the job is distinct.

Explanation-

As we proved above, from statement (a), **correct order is non-decreasing order** of run time:

If all job will have distinct run time, there exists **ONLY ONE non-decreasing order**.

**Q.(3)**

This algorithm will NOT always generate a minimum node cover.

Icon

Description automatically generatedLet’s take following example:

A tree with A as a root node. It has B, C, D as child nodes. B has E, C has F and D has G as child node.

The algorithm above will recognize that node A is the vertex of greatest degree (= 3) and add it to the arrangement set. Then, note that every one of the excess nodes have greatest degree = 1 and there are three edges left in the tree. Thusly, three hubs should be added to the arrangement set to cover every one of the node in the first tree. Consequently, the cardinality of the arrangement set following this calculation will be 4. Nonetheless, the ideal cardinality is 3, when we simply select nodes (B, C, D). Subsequently, this calculation won't necessarily deliver a minimum node cover.

**Q. (4)**

Chart

Description automatically generated with medium confidence

Initially all have 0 used capacity

For forward labeling Delta = Capacity- flow

For backward labeling Delta = flow

Lets consider following augmented paths:

**AP1: S->v2->v4->T**

Here Delta is 13,14,4

V4->T decides throttling.

So flow for all will increase by 4

A picture containing text, watch

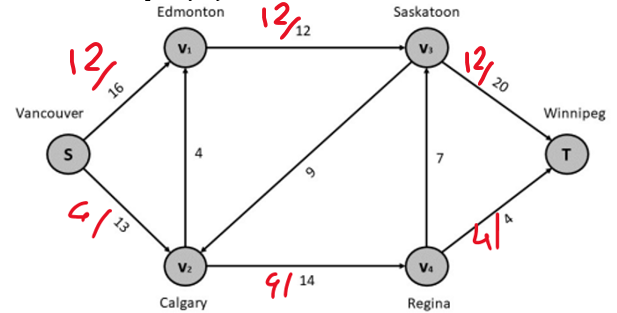
Description automatically generated

**AP2: S->v1->v3->T**

Delta is 16, 12, 20

V1->v3 decides throttling.

Increase used capacity by 12

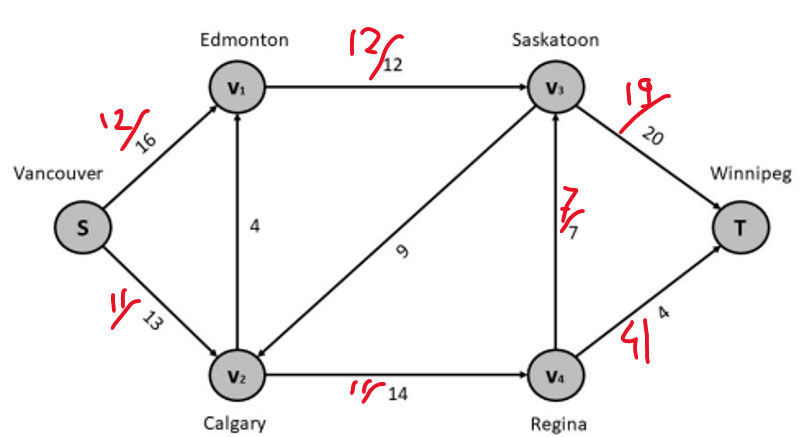


**AP3: S->v2->v4->v3->T**

Delta is 9, 10, 7, 8

V4->v3 decides throttling.

Increase used capacity by 7



No more augmented path cam be found. So we end our search.

So **maximum flow will be 19+4 = 23.**

**The minimum cut C(S:S¯) is across S={S,v1,v2,v4} and S¯= {v3,T}.**