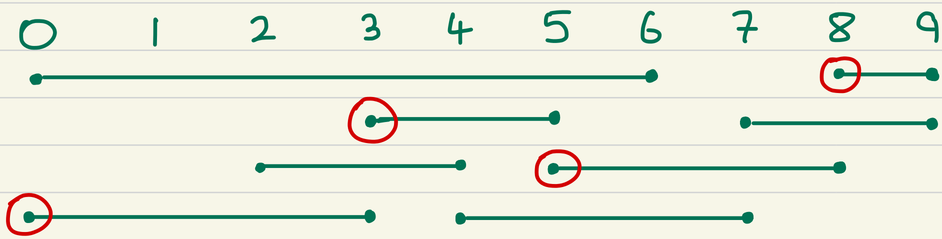
CSE 531 - HW2

Ratik Dubey

① set[n] = {1, 2, ..., n}

$i \in n \rightarrow (s_i, f_i)$

$f_i > s_i$



1a) Schedule the activity $i \in [n]$ with the latest starting time

O → activities with latest start times

Yes, the decision is safe.

Proof: It is safe to schedule intervals with latest start times because there is an optimal sol^n using this strategy

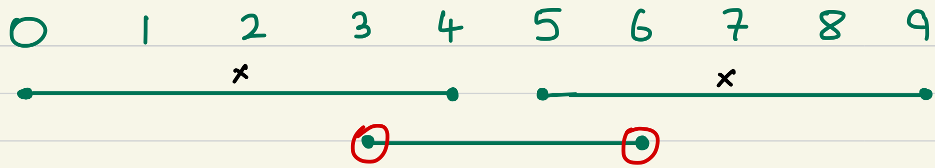
We cover all intervals in the example shown above using latest start times and get max. jobs

Intervals ⇒ [(0,3), (3,5), (5,8), (8,9)]

Job = 4

Pick any random job (ex: 2-4), it is covered in the intervals obtained from our optimal strategy.

and no. of jobs are max ⇒ DONE



1b> Pick the longest activity $i \in [N]$. If i conflicts with some other job in $[n]$, then we don't schedule i; otherwise we schedule i

O → longest activities selected

We try to choose (0,4), it is conflicting with (3,6) so we don't select it and move on

We try to choose (5,9), it is also conflicting with (3,6) so we don't select it and move on

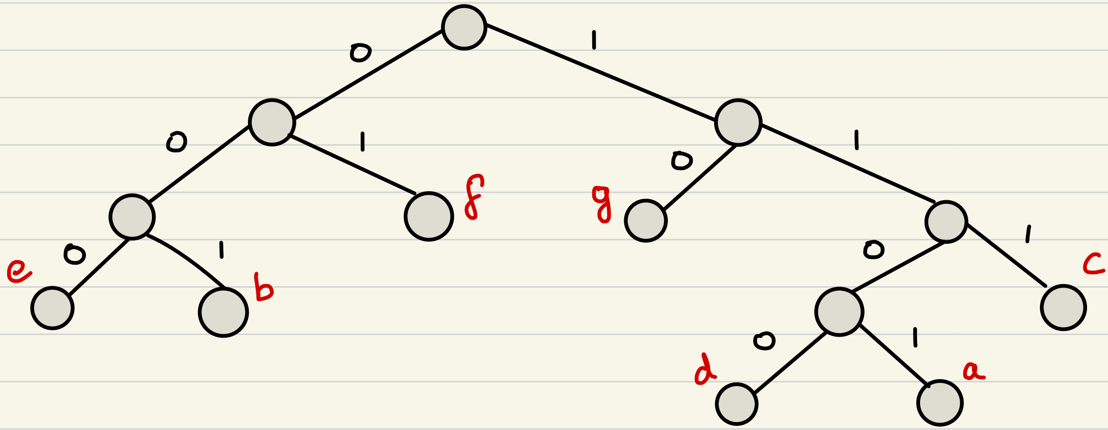
Finally we choose (3,6) as other intervals have been discarded so it is not conflicting.

But this is not the optimal sol^n as it doesn't give us max. no. of jobs. That would be given by selecting (0,4) and (5,9)

⇒ The decision is not safe.

② Huffman Code

Letters	a	b	c	d	e	f	g
Frequencies	13	42	51	24	25	70	58



Letter	a	b	c	d	e	f	g
Freq.	13	42	51	24	25	70	58
Encoding	1101	001	111	1100	000	01	10
Length	4	3	3	4	3	2	2

$$\text{Weighted Length} = (13\times4)+(42\times3)+(51\times3)+(24\times4)+ \\ +(25\times3)+(70\times2)+(58\times2)$$

$$= \underline{\underline{758}}$$

③ $1 \le k \le n$

$A[1 \ldots i] = [k, k+1, \ldots, n]$

$o/p = [b_k, b_{k+1}, \ldots, b_n]$

Python 3

```
def maxSum (A, k):

    minHeap = []
    res = []
    curSum = 0

    for i in range(k):
        heapq.heappush(minHeap, A[i])
        curSum += A[i]

    res.append(curSum)

    for i in range(k, len(A)):
        curSum += A[i]
        popped = heapq.heappushpop(minHeap, A[i])
        curSum -= popped
        res.append(curSum)

    return res
```

④ $X = \{x_1, x_2, \ldots, x_n\} \longrightarrow$ n points

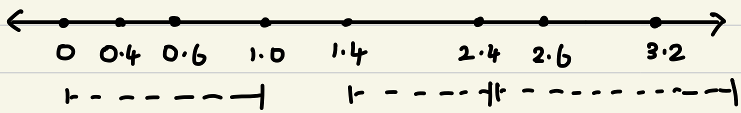
Python 3

```
def smallestIntervals(x):

    res = set()
    last = float("-inf")

    X.sort()

    for i in x:
        if i >= last:
            interval = (i, i+1)
            res.add(interval)
            last = i+1

    return res
```

- Our strategy is that if the i^{th} point on the line is greater than or equal to the last interval's end time then we add a new interval of unit length starting at i to the resut (i, i+1) and set last to (i+1). Here the intervals are closed intervals.

- It is safe to use our strategy as there is an optimal solution when we follow the strategy.

<u>Proof</u>:



Optimal sol^n = [(0,1), (1.0,2.0), (2.4, 3.4)]

Randomly select any point on line
ex - 1.4 $\rightarrow$ sol^n contains 1.4 $\Rightarrow$ <u>DONE</u>

- Lets consider the above line, when we start last = (-inf), we reach 0 on the line and add interval (0,1) to res, last = 1
  Now only consider line from 0.4 onwards, 0.4 is not >= 1 so we skip, 0.6 is also not >= 1 so we skip. 1 is >= 1 so we add new interval (1.0, 2.0) to res
  $\Rightarrow$ this is how the reduced instances work in the optimal sol^n provided.
  (we see point, add interval if needed, truncate line and <u>repeat</u>)