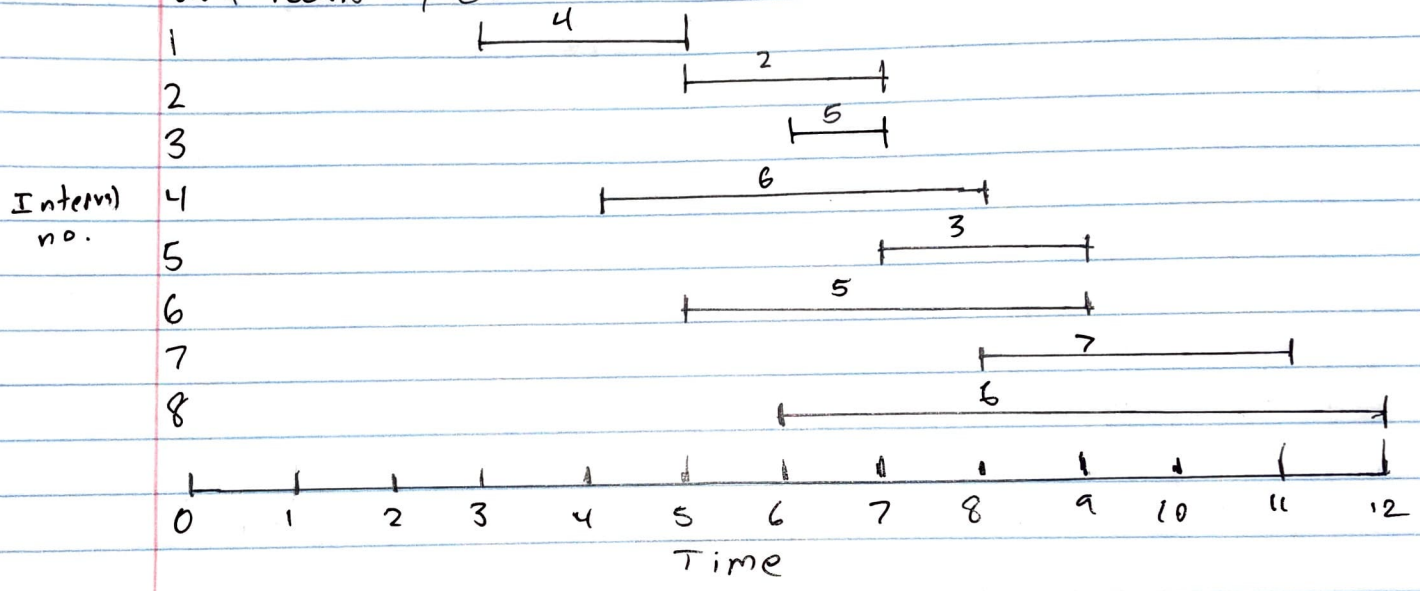
CS330 hw 7 Gordon Ng

1i) def WeightedIntervalSchedule(Intervals):

~~[illegible]~~

opt(j) = max(v_j + opt(p(j)), opt(j-1))

Since the schedule is already sorted by ETFT, we look for the most optimal situation by utilizing the max function. We check in on the greatest weighted arrays/schedules and decrement from there. If the previous/next greatest fits in the schedule with no overlaps, add it to the optimal, if it is not, we go to the next latest finish time and recursively check from there.



The most opt solution is # 1, 3, 7, where (4+5+7=16)

ii)

j	1	2	3	4	5	6	7	8
p(j)	0	1	1	0	3	1	4	1

If the schedule before the current schedule array fits/has no time conflicts with current, it is incremented. Here we see that array 7 can be paired with 3 other schedules without conflict. Compare time of finish of last array with time of start of current array.

iii) M is the memoization with the function

max{v_j + m-Compute-opt(p(j)), m-compute-opt(j-1)}.

j	0	1	2	3	4	5	6	7	8
M	0	4	6	9	9	12	12	16	16

max(0,0) max(4,0) max(5+1,4) max(5+4,6) max(9,6) max(3+5+4) max(12,3+5+4) max(16)

2i) def MaxSub (L, len(L)):

```
int maxcurr = 0;
int maxall = 0;
for i => len(L):
    maxcurr = maxcurr + L[i]
    if (maxcurr < maxall):
        maxcurr = maxall
    if(maxall <0)
        maxall = 0;
return maxcurr
```

This is correct because it has 1 for loop, it keeps track of the current max with index i to the beginning and the max of the whole array. It finds the max sum of the array and sets -1 or negative numbers to zero.

Since only 1 loop is used, it is O(n).

ii) L = -2 4 -3 5 -2 3 1 -5 5 2 -4

int maxcurr = 0

int maxall = 0

index	0	1	2	3	4	5	6	7	8	9	10
maxcurr = 0	0	4	1	6	1	6	7	3	8	10	6
maxall = 0	0	4	4	6	4	7	8	8	8	10	10

It holds the max ending at index and continues to add to get the max contiguous sum.