9 Feb 2018: Finishing Weighted Interval Scheduling

Recap of Max Weight Interval Schedule (MWIS) algorithm from last time...

MWIS (intervals $1, \ldots, k$):

1. If $A[k] \neq$ NULL return $A[k]$. Else...
2. $S_0 = MWIS(1, \ldots, k-1)$
3. $S_1 = MWIS$ (the set of intervals that finish before s_k)
4. Compute $w(S_0)$, $w(S_1 \cup \{k\})$.
5. Store the better of these two sets as $A[k]$.
6. Return $A[k]$.

To verify that this works, these must be the first j intervals, for some $j < k$.

Running time devoted to MWIS(k):
$O(k)$ first time the subroutine is called. (Line 4)
$O(1)$ every subsequent time.

$\therefore O(n)$ in total. $(= O(k) + O(n-k))$

That's $O(n)$ for each $k = 1, \ldots, n$.
So the algorithm runs in $O(n^2)$ total.

Improving the algorithm.

[1] Change the arguments of MWIS from a set to an integer k in $0, \ldots, n$.
Representing the set of intervals $1, 2, \ldots, k$.

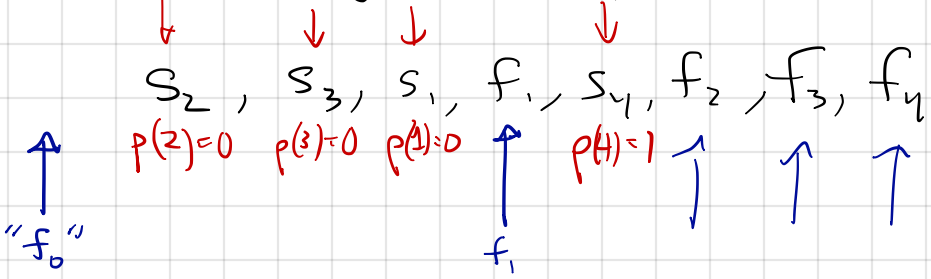
[2] Change the return value from Set to (Set, Int) — where the second part of the ordered pair is the weight of the set.

[3] Finding latest finish time before s_k (line 3) naively requires binary search — $O(\log n)$ repeated n times

Instead of binary search we can do $O(n \log n)$ preprocessing to compute a lookup table that gives the answer for all s_k.

Preprocessing. Sort $\{s_1, s_2, \ldots, s_n, f_1, \ldots, f_n\}$ in increasing order.

Assume for simplicity that intervals numbered s.t. $f_1 \le f_2 \le \ldots \le f_n$.



This builds an array $p(k)$ in $O(n)$ time after sorting step, such that $p(k)$ is largest index such that $f_{p(k)} < s_k$, $\forall k$.
Or $p(k)=0$ if no such index exists.

0. Initialize $A[k] = \text{NULL}$ $\forall k = 1, \ldots, n$
1. Sort $\{s_1, \ldots, s_n, f_1, \ldots, f_n\}$. Renumber intervals if necessary so $f_1 \le \ldots \le f_n$.
2. Create the array $p(k)$ of "predecessor pointers."
3. Call MWIS(n).

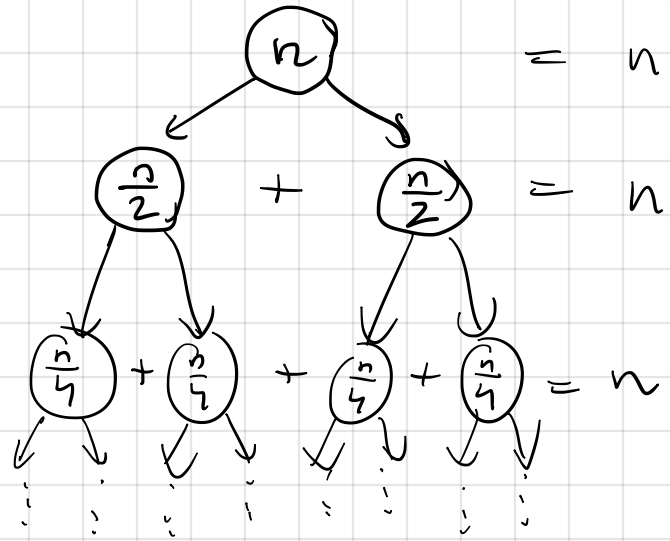
MWIS(k):

```
if A[k] ≠ NULL return A[k].
if k = 0 return (∅, 0).
if k > 0
    let (S_0, W_0) = MWIS(k-1).
    let (S_1, W_1) = MWIS(p(k)).
    if W_0 > w_k + W_1 set A[k] = (S_0, W_0)
    else set A[k] = (S_1 ∪ {k}, w_k + W_1)
    return A[k]
endif
```

Assessing running time of recursive algorithms.

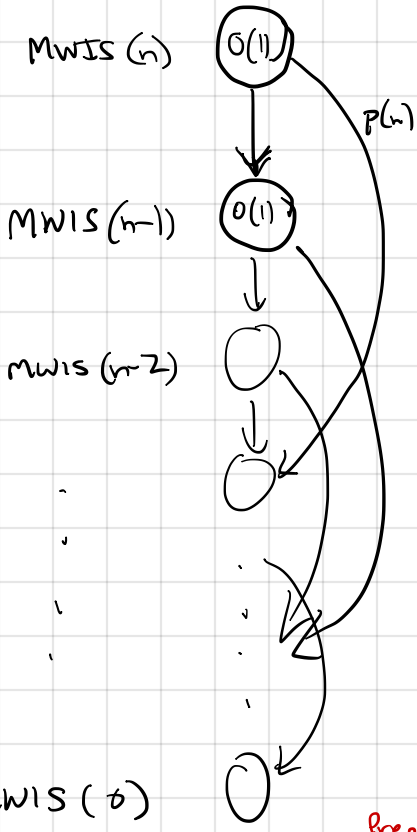
A diagrammatic way of describing the running time:
- create a node for each argument you pass to the recursive procedure.
- create arrows from the calling procedure to the procedure it calls.
- label each node with (an upper bound on) the time spent in that procedure call, ignoring time spent in recursive sub-calls.

E.g. MergeSort



Depth = $\log(n)$
Total = $n \log n$.

MWIS



Preprocessing

nodes = $n+1$

work per node = $O(1)$

Running time = $O(n) + O(n \log n) = O(n \log$

Loop MWIS:

1. Init: $A[k] = \text{NULL} \quad \forall k$.
2. Sort $\{s_1, \ldots s_n, f_1, \ldots f_n\}$
3. Precompute $p[k]$ array.
4. for $k = 0, \ldots, n$:
   - if $k = 0$ $\quad A[k] = (\emptyset, 0)$.
   - if $k > 0$
     - let $(S_0, W_0) = A[k-1]$
     - let $(S_1, W_1) = A[p(k)]$
     - if $(W_0 > W_1 + w_k)$
       - set $A[k] = (S_0, W_0)$
     - else
       - set $A[k] = (S_1 \cup \{k\}, W_1 + w_k)$
   - endif

   endfor
5. Output $A[n]$.