# OPTIMAL BINARY SEARCH TREE

Given: A set of items $\{1, 2, \ldots n\}$ and frequencies $p_i$ for $i = 1, 2, \ldots n$

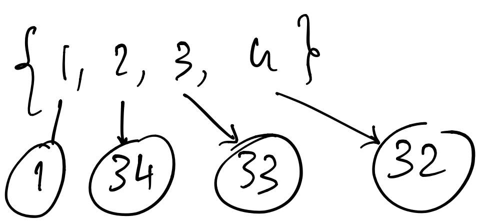
O/P: A BST $T^*$ such that

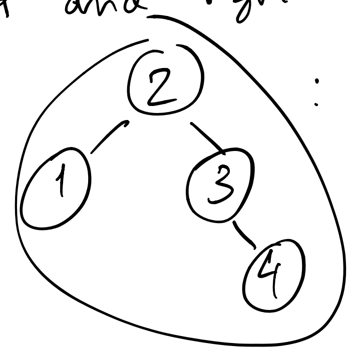
Total weighted search time $= \sum_{i=1}^{n} p_i \cdot \left[\text{search time for } i\right]$

is minimized

Attempt:- ▷ Fix item with highest freq. as root

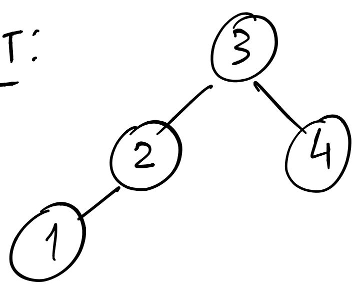
▷ Recurse on left and right.




: 198

OPT:




: 168

▷ Attempt 2 :- Fix Median of freq. (Above example fails)

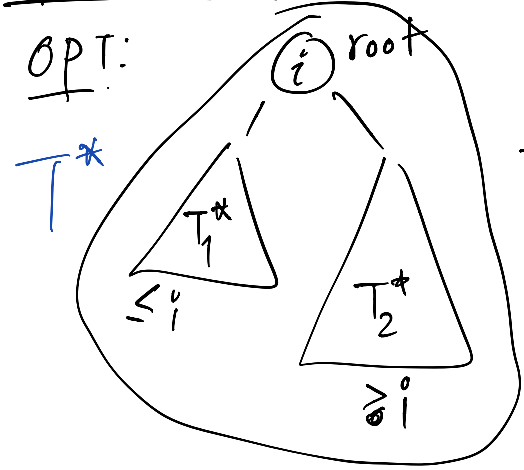
▷ Attempt 3 :- Use the ~~highest freq.~~ lowest freq. item as a leaf and go bottom up.

(Bad example :- HW)

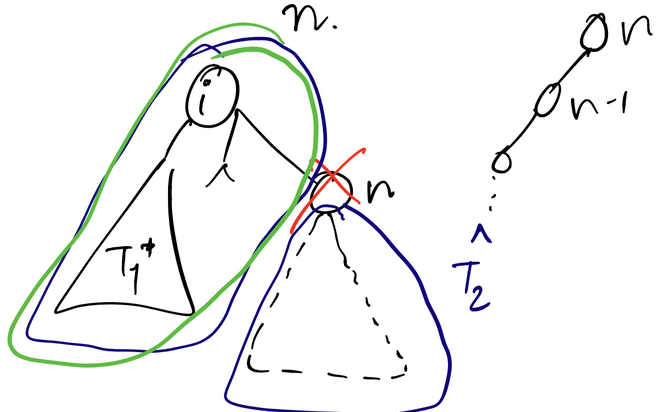
In general : Greedy choices don't work !

# Dynamic Program

OPT:




▷ Consider the last item $n$.




let $c(T^*)$ denote the cost of $T^*$.

$$c(T^*) = c(T_1^*) + c(\hat{T}_2) + \boxed{\sum_{l \in \hat{T}_2} p(l) \cdot level(n)}$$

depends on subproblem!!

▷ Consider removing the root $i$.

$$c(T^*) = c(T_1^*) + c(T_2^*) + \sum_{l=1}^{n} p(l)$$

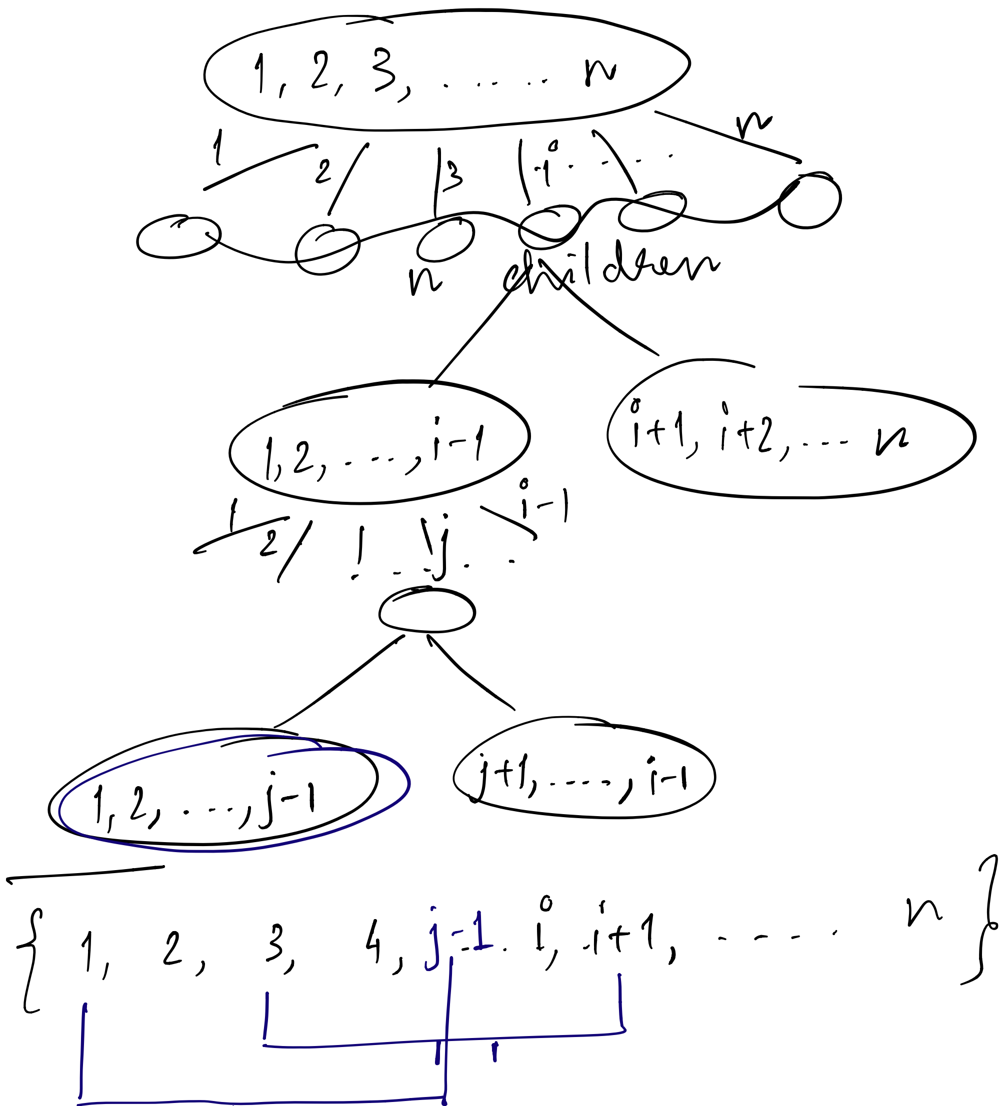
independent of the subproblems

Claim. $c(T_1^*)$, $c(T_2^*)$ are optimal for items $\{1, 2, \ldots i-1\}$, $\{i+1, \ldots, n\}$ respectively, where $i$ is the root.

Pf: Suppose not and $\hat{T}_1$ is a better sol$^n$ for $\{1, 2, \ldots, i-1\}$

$$\Rightarrow \quad c(\hat{T}_1) + c(T_2^*) + \sum_{l=1}^{n} p(l) < c(T^*)$$

[contradiction]

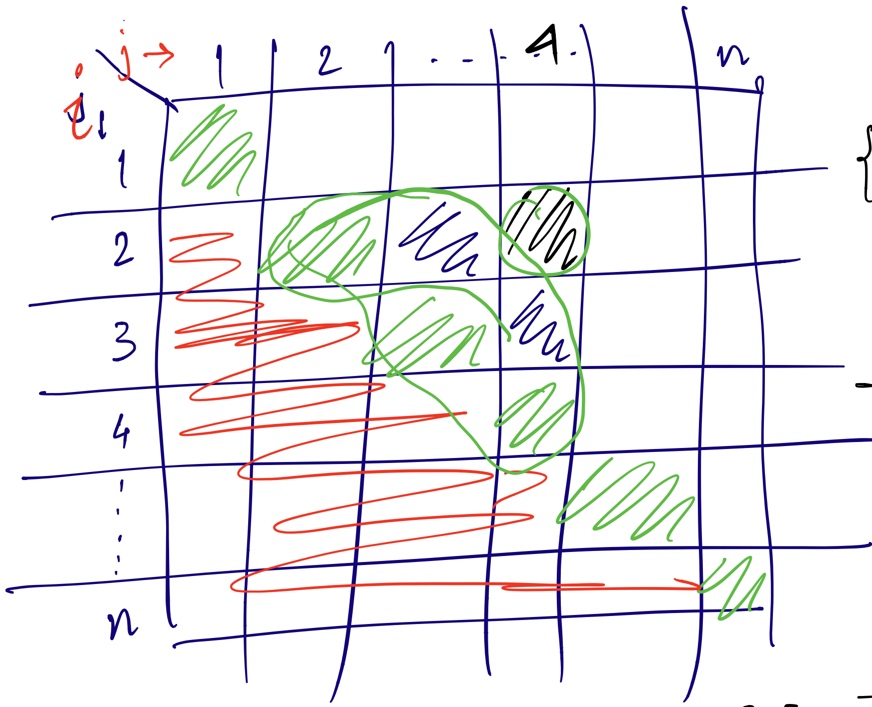



- Every subinterval $[i, j]$, $1 \le i \le j \le n$ will have a subproblem.

$T(i,j)$ = Optimal solution for $\{i, i+1, \ldots, j\}$

$$T(i,j) = \min_{i \le k \le j} T(i, k-1) + T(k+1, j) + \sum_{r=i}^{j} p(r).$$

Base cases:- $T(i,i) = p(i)$, $T(i,j) = 0$ , $j < i$




$T(2,4)$

$\{T(2,2),\ T(2,3),\ T(3,3),\ T(3,4),\ T(4,4)\}$

Proceed from one diagonal to the next.

**Algorithm:** Initialize $T[1\ldots n][1\ldots n]$

for $s = 0$ to $n-1$ ▷ "width" of a diagonal $= s$

for $i = 1$ to $n-s$

$$T[i, i+s] = \min_{k=i}^{i+s} \left[ \sum_{r=i}^{i+s} p(r) + T[i, k-1] + T[k+1, i+s] \right]$$

Return $T[1,n]$.

▷ Runtime: ▷ Filling up $n^2/2$ cells require $\Theta(n^2)$ time.

▷ At least half of these cells require $\Theta(n/2)$ computation each.

▷ Total runtime $\Theta(n^3)$.

Fun Fact 1. Knuth '70's designed ~~an~~ a $\Theta(n^2)$ algorithm for this problem.

Fun Fact 2. Splay Trees: (Sleator, Tarjan '80s)

→ "Almost" optimal performance without even knowing frequencies!!

→ "Almost" optimal $\sim \Theta(1) \cdot \mathrm{OPT}$.