Dynamic Programming.

SUBSET-SUM

I/p : A set n positive integers and T

Output : Yes if a subset adds to T
No o.w.

I/p : $x[1 \ldots n]$ and T

$$SS(n, T) = \begin{cases} \text{True} & \text{if } T = 0 \\ \text{False} & \text{if } T < 0 \\ SS(n-1, T - x[n]) \vee SS(n-1, T) & \text{o.w} \end{cases}$$

Running time : $O(2^n)$.

$SS(i, T')$ = True if there is a subset of $x[1 \ldots i]$ that add up to T'.

$$\underline{SSTable[\,i\,,\,T'\,]}$$

where $0 \le i \le n$ and $0 \le T' \le T$

* DS : A two dimensional array
* Evaluation order : i : 0 to n

  T' : 0 to T

Running time : $\underline{\underline{O(nT)}}$

It is polynomial in $\underline{n}$

and absolute values of the input.

Pseudopolynomial time algorithm.

Polynomial time:

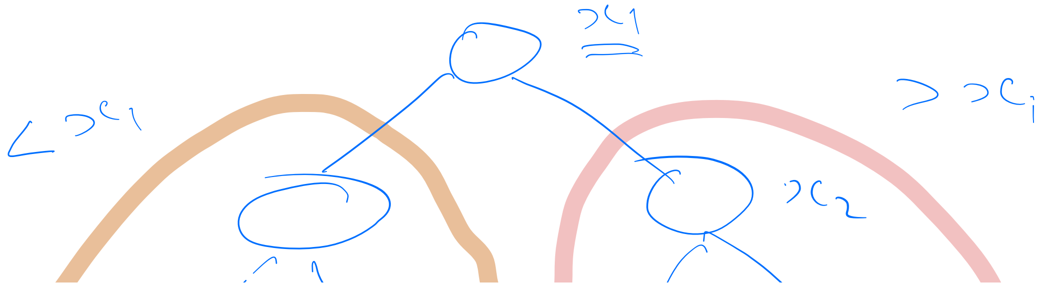
n^c, $n^{O(c)}$

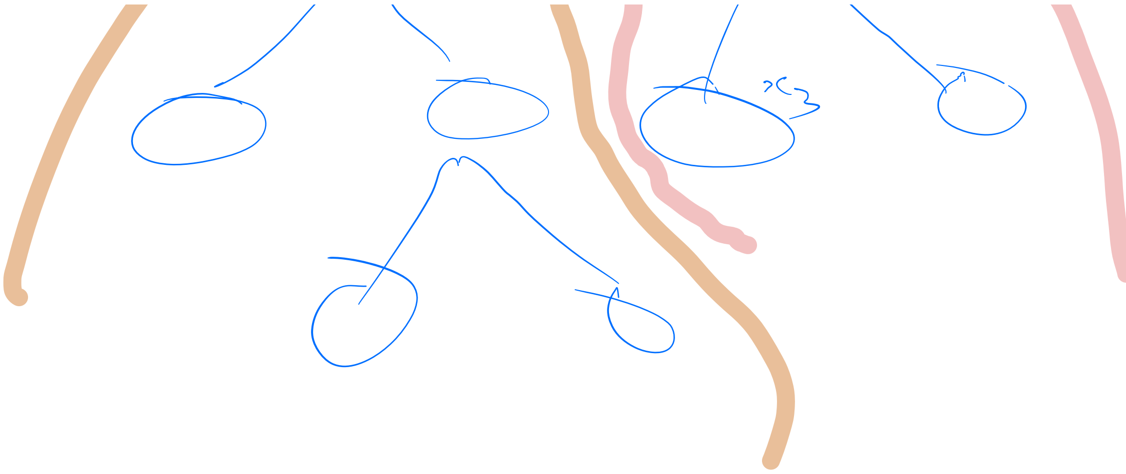
Can we get polynomial time algorithm for Subset-Sum

SUBSET-SUM is NP-hard.

"unlikely to get polynomial time algorithm".

Binary Search Tree.





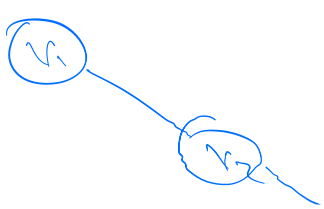
Worst-case complexity for searching:
O(depth of the tree)

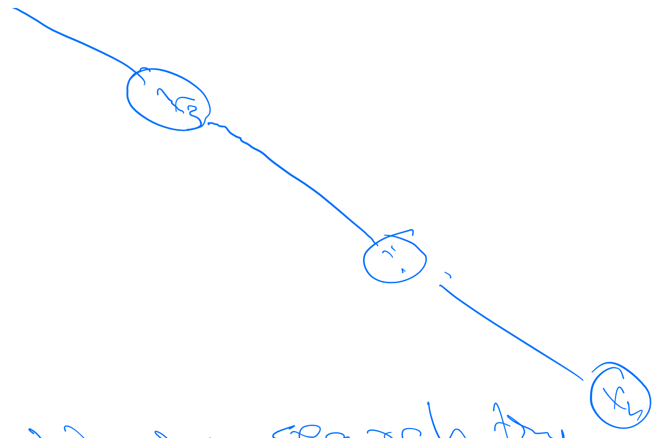
Input: A[1 - - - n]

$r_1 < \dots < r_n$

F[1 - - - n]

r_i will be searched for F[i] # of times

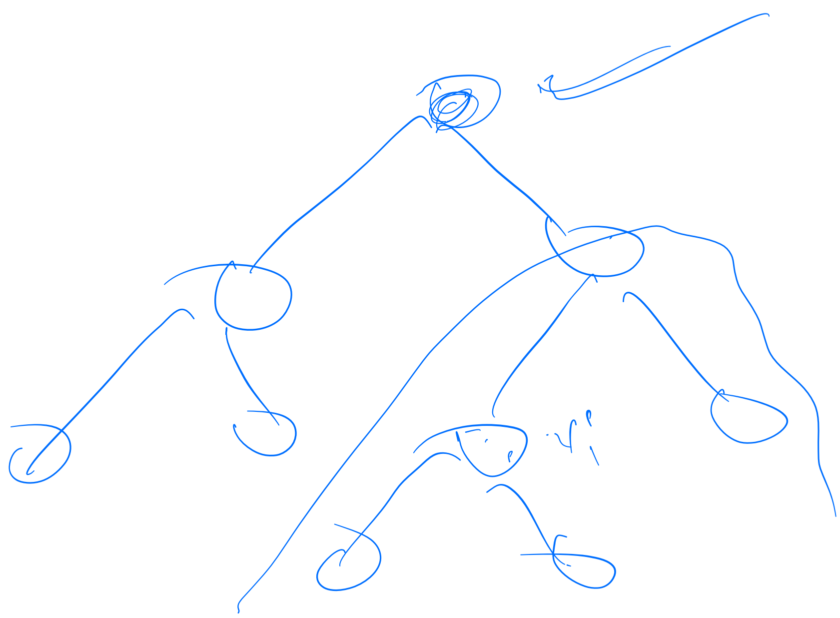




Let T be a binary search tree.

$$\text{cost}(n, T) = \sum_{i=1}^{n} F[i] \cdot \#\text{ of ancestors of } x_i \text{ in } T$$

x_i

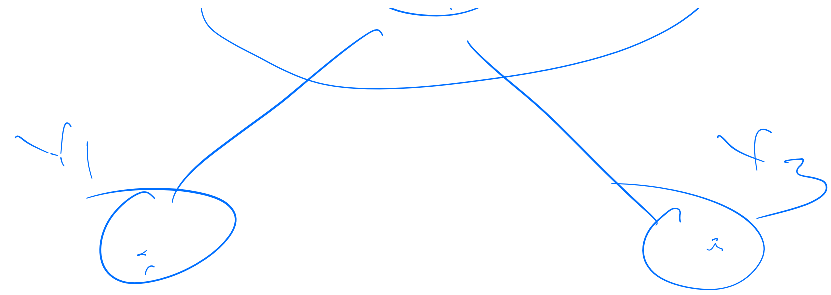


$$\text{cost}(1, n, T) = \sum_{i=1}^{n} F[i] \; +$$

$cost(1, r-1, left(T,r)) +$

$\rightarrow \underline{cost(r+1, n, right(T,r))}$

	r	x_2	x_3
	1	2	3
F:	10	4	7



$\text{Cost} = \cancel{4 \times 1 + 10 \times 2 +}$

$\cancel{7 \times 2}$

$= 4 + 10 + 7 +$

$\rightarrow \text{cost}(1, 1, \text{left}(T))$

$\rightarrow + \text{cost}(3, 3, \text{right}(T))$

$= \cancel{4 + 10 + 7 +}$

$\cancel{8\ 10 +}$

$$OptCost(1, n) = \sum_{i=1}^{n} F[i] \; +$$

$$\min_{1 \le r \le n} \left\{ optcost(1, r-1) + optcost(r+1, n) \right\}$$

$$\text{OptCost}(i, j) = \begin{cases} 0 & \text{if } j > i \\ \displaystyle\sum_{k=i}^{j} F[k] + \min_{i \le r \le j} \text{OptCost}(i, r-1) + \text{OptCost}(r+1, j) \end{cases}$$

OptCost(1, n).

Dynamic programming.

DS: OC[1..n, 1..n]

Evaluation order:

Opt(i, j) for all

$j - i = 0$.

Opt(i, j) for all

$j - i = 1$

$\vdots$

For $d = 0$ to n

For $i = 1$ to n

Compute OptCost$(i, i+d)$

```
OptimalBST(f[1..n]):
    InitF(f[1..n])
    for i ← 1 to n+1
        OptCost[i, i−1] ← 0
    for d ← 0 to n−1
        for i ← 1 to n−d          ⟨⟨...or whatever⟩⟩
            ComputeOptCost(i, i+d)
    return OptCost[1, n]
```

$O(n^3)$. ✓

Edit Distance.

FOOD ⟶ MONEY

$\text{FOOD} \xrightarrow{\text{edit}} \text{MOOD}$

$\downarrow$ edit

MOND

$\downarrow$ insert

MONED

$\downarrow$ edit

MONEY

min # of operations
(replace, insert, delete letters)

(replace, insert, [illegible]

to change a source string to a destination string.

Exc: Write a recursive formula for this problem.