Review

The DSelect Algorithm

Elements that are > or < m

$$3\left(\lceil\frac{1}{2}\lceil\frac{n}{5}\rceil\rceil-2\right)\geq\frac{3n}{10}-6$$

Thus, the *upper bound* of the number of elements we put into the recursive call:

$$n-\left(\frac{3n}{10}-6\right)=\frac{7n}{10}+6$$

Recurrence

With this in mind, our recursion complexity looks along the lines of

$$T(N)=\begin{cases}1, N=1\\ T(n)\leq cn+T\left(\frac{n}{5}\right)+T\left(\frac{7n}{10}\right)\end{cases}$$

Hope and Check

Hope: There is some constant a [independent of a] such that $T(n)\leq an, \forall n\geq 1$
If true, $T(n)=O(n)$

Analysis

Claim: Let $a=10c$, then $T(n)\leq an, \forall n\geq 1$

Proof: By induction on n.

Base Case: $T(1)=1\leq a\times 1 (n=1,$, since $a\geq 1)$

Induction Step: $n>1$

$$\text{Induction Hypotheseis: } T(k)\leq a\times k, \text{ for } k<n$$

$$\begin{aligned}\text{We have } T(n)&\leq cn+T(n/5)+T(\frac{7n}{10})\\ &\leq cn+a(n/5)+a\left(\frac{7n}{10}\right)\\ &=n(c+\frac{9a}{10})=an\end{aligned}$$

Dynamic Programming

Dynamic programming is a method for designing efficient algorithms for recursively solvable problems with the following properties:

1. Optimal Substructure: An optimal solution to an instance contains an optimal solution to its sub-instances
2. Overlapping Subproblems: The number of subproblems is small so during the recursion same instances are referred to over and over again.

Basically fancy recursion

Four steps in solving a problem using dynamic programming:

1. Characterize the structure
2. Recursively define the value of a solution
3.

A more refined / intuitive list: [1]

1. Define subproblems
2. Write down the recurrence that relates the subproblems
3. Recognize and solve the best cases

Fibonacci

Sequence: $1, 1, 2, 3, 5, 8, 13, \ldots$

Naive Recursion Approach

```
def fib(n):
    if n <= 2:
        return 1
    return fib(n - 1) + fib(n - 2)
```

Runtime

$O(2^n)$ We calculate the same value multiple times.

$$T(n) = T(n-1) + T(n-2) + 2$$

[1] Recursion + memorization

Memoized DP Approach

```
memo = {}
def fib(n):
    if n in memo: return memo[n]
    if n <= 2:
        f = 1
    else
        f = fib(n - 1) + fib(n - 2)
    memo[n] = f
    return f
```

Bottom-Up DP Approach

```
fib = {}
for k in range(n):
    if k <= 2:
        f = 1
    else:
        f = fib[n - 1] + fib[n-2]
    fib[k] = f
return fib[n]
```

Coin Change

Given n, find the number of different ways to write n as the sum of $1, 3, 4$

Example: $n = 5$

$$\begin{aligned} 5 &= 1+1+1+1+1 \\ &= 1+1+3 \\ &= 1+3+1 \\ &= 3+1+1 \\ &= 1+4 \\ &= 4+1 \end{aligned}$$

$$\left.\begin{aligned} 5 &= 1+1+1+1+1 \\ &= 3+1+1 \\ &= 1+3+1 \\ &= 4+1 \end{aligned}\right\} D_4$$

$$= 1+1+3$$

Define Subproblems

$$\text{Let} D_n \text{ be the number of ways to write} n \text{ as the sum of } 1, 3, 4$$

Find the recurrence

1. Consider one possible solution $n = x_1 + x_2 + \cdot + x_m$
2. If $x_m = 1$, the rest of the terms must sum to $n - 1$
3. Thus, the number of sums that end with $x_m = 1$ is equal to D_{n-1}
4. Take other cases into account ($x_m = 3, x_m = 4$)

$$D_n = D_{n-1} + D_{n-3} + D_{n-4}$$

Base Cases

$$D_0 = 1 \qquad\qquad D_1 = 1$$