Time Complexity of Recursive Algorithms

We must understand how to analyze recursive algos in terms of their running time.

```
function factorial(n) - - - - - - - - - - T(n)
    if (N == 1) - - - - } Base - - - -     C_0
      return 1   C_5    } Case
    return n * factorial(n-1) } Recursive call - - - - C_6 + T(n-1)
```

$\underbrace{c_1 + c_2 + c_4}_{c_6}$ $\quad \underbrace{\text{factorial}(n-1)}_{T(N-1)}$

We need to define T(n) for the function above.

$\therefore$ We can say that $\quad T(n) = C_0 + C_6 + T(n-1)$

$\quad = C_7 + T(n-1)$

The expression T(n) is recursive. The running time of the algo is expressed in terms of T(n-1). This is know as a recurrence equation.

It descibes the running time, T(n), of a problem of size N, in terms of the running time of smaller inputs, until the input is small enough that T(n) calculation is straigthforward.

2.202 Time complexity of recursive algorithms

TOTAL POINTS 3

1. What is the recurrence equation that describes the running time of the algorithm below?

1 p

```
A: 1D array of N integer numbers
B: 1D array of N integer numbers
N: number of elements of arrays A and B
function ALG1(A,B,N):   T(N)
    if(N==0):  - - - - - C0
        return 0 - - - - C1
    return A[N-1]*B[N-1]+ALG1(A,B,N-1) - - - -
```

$C_2 \quad C_3 \quad C_4 \quad C_5 \quad C_6 \quad C_7 \quad C_8 \quad C_9 \quad C_{10} \quad \rightarrow T(N-1)$

C_{11}

- (x) T(N)=T(N-1)+C
- ( ) T(N)=T(N/2)+C
- ( ) None of the others
- ( ) T(N)=T(N+1)+C
- ( ) T(N)=2*T(N-1)+C

$$* \; T(N) = \underbrace{C_0 + C_{11}} + T(N-1)$$
$$= C_{12} + T(N-1)$$

2. What is the recurrence equation that describes the running time of the algorithm below?

1 point

```
function ALG3(A,N)
    if(N==0)
        return 0
    return A[N-1,N-1]+ALG3(A,N-1)
```

$T(n)$

C_0

C_1

$\underbrace{C_2 \quad C_3 \quad C_4 \quad C_5 \quad C_6 \quad C_7 \quad C_8}_{C_9}$

$T(n-1)$

$$T(n) = C_0 + C_9 + T(n-1)$$
$$= C_{10} + T(n-1)$$

- T(N)=T(N-1)+C ✓
- T(N)=T(N+1)+C
- T(N)=2*T(N-1)+C
- None of the others
- T(N)=T(N/2)+C

3. What is the recurrence equation that describes the running time of the algorithm below?

```
A: 1D array
low: lowest index
high: highest index

function ALG2(A,low,high):          T(n)
    if(high-low==1):                 C0
        if(A[low]<A[high])           C1
            return high              c
        else                         c
            return low               c
    if (low==high)                   c   C2
        return high                  c
    mid=low+floor((high-low)/2)      C3
    a=ALG2(A,low,mid)                a = c4 + T(n/2)
    b=ALG2(A,mid+1,high)             b = c5 + T(n/2)
    if(A[a]>A[b])                    C6
        return a
    return b                         C7
```

- [x] T(N)=2*T(N/2)+C
- [ ] None of the others
- [ ] T(N)=T(N/2)+C
- [ ] T(N)=T(N+1)+C
- [ ] T(N)=2*T(N-1)+C

$C_8 = C_0 + C_1 + C_2 \dots + C_7$

$T(n) = C_8 + T(n/2) + T(n/2)$

$= C_8 + 2T(n/2)$

arr = [1, 2, 3, 4, 5, 6]

mid = 0 + floor((5-0)/2) = 2

a = ALG2(A, low, mid) | b = ALG2(A, 3, 5)

mid = 0 + (2/2) = 1

⋮

Solving Recurrence Equations

The recurrence equations are running time expressions where the running time of N depends of the running time of N-1. ex. $T(n) = C + T(N-1)$

The problem with expression like these are that we do not have an explicit expresison for the running time of the algo. We must solve the recurrence equation in order to find its time complexity.

How to solve a recurrence equation

1. Find a value of N, for which T(n) is known (not a reccurence equation). Think of the best case.
2. Expand the right side of the equation until you can replace T(n) with known value.

$T(n) = C_0 + T(n-1)$

```
function FACT(N)
    if N = 1 then
        return 1
    end if
    return N· fact(N − 1)
end function
```

$\left.\begin{array}{l} C_0 \\ C_1 \end{array}\right\} C_2$ Base case execute

1) If $n=1$, the base case executes.

$T(1) = C_2$

2) Expand the right side and look for a pattern. Expand the expression using the def'n of $T(n)$, until we can replace the recursive term with the known value from step 1:

any #

$k=1$ $T(n) = C_0 + \underbrace{T(n-1)}$

$k=2$ $T(n) = C_0 + (C_0 + (T(n-1-1))) = C_0 + C_0 + T(n-2)$

$k=3$ $T(n) = C_0 + C_0 + (C_0 + T(n-3))$

We've found a pattern, where C can be multiplied by k, where T(n) gets decreased by k units

$T(\cancel{n}^{3}) = T(3) + T(n-1)$

$T(n^{3-1}) = T(3) + T(2)$

To do so, we expand the expression of T(N-1) using the definition of T(N).

$T(N-1)$ $T(\cancel{N}) = C_5 + (C_5 + T(N-2))$

Therefore, T(N-1) Is equal to C5 plus T(N-1-1), which is T(N-2)

$T(N-2)$ $T(N) = C_5 + (C_5 + (C_5 + T(N-3)))$

Now, we expand the expression T(N-2), applying the same method. You can start seeing a pattern here.

Every expansion of T(n-1) makes the argument of T(n) smaller and smaller.

When $*k = n-1$, we will find that, $T(n-(n-1))$ is $T(1)$, which is constant.

$$T(n) = k*C_0 + T(n-k)$$

$$T(n) = (n-1)*C_0 + T(n-(n-1))$$

$$= (n-1)C_0 + T(n-n+1)$$

$$= (n-1)*C_0 + T(1)$$

$$T(n) = (n-1)C_0 + C$$

$$= C_0 n - C_0 + C$$

$$= C_0 n + C$$

This is a linear function

This shows that T(n) is no longer described in terms of itself with smaller, rersive arguments.

* Now we can complete an asymptotic analysis.

$T(n)$ is $O(n), O(n^2)\ldots$ $\Theta(n)$

$\Omega(n), \Omega(\log n)\ldots$

1. The solution of the recurrence equation T(N)=T(N-1)+C (assuming that T(0) is equal to a constant value) is:

- ○ $T(N) = C_0$
- ○ None of the others
- ○ $T(N) = C_0 * N * logN + C_1$
- ○ $T(N) = C_0 * logN + C_1$
- ● $T(N) = C_0 * N + C_1$

1) $T(0) = C$

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

$k = 1$

$k = 2$

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

$= C + C + (C + T(n-3)) = 2C + C + T(n-3)$ $k=3$

$= k*C + T(n-k)$

When $n = 0$

$T(0) = C$

$T(n-k) = C$

$n - k = 0$

$n = k$

$\therefore T(n) = T(n-n) + Cn$

$= T(0) + Cn$

$= C + Cn$

II) Method 2

$T(n) = kC + T(n-k)$

$= nC + T(n-n)$

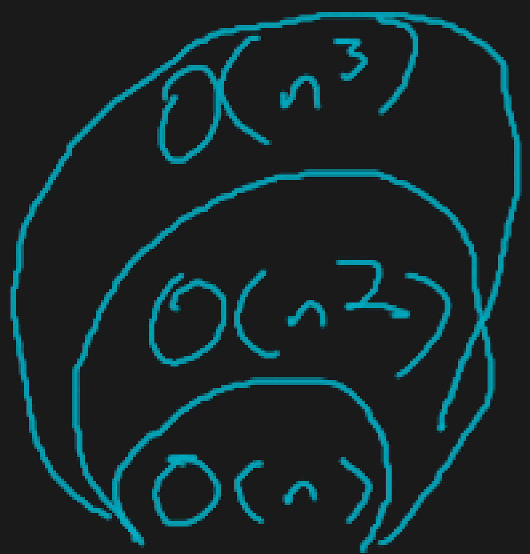
$= nC + T(0)$

$= Cn + C$

* We know we want to arrive at T(0).

2. An algorithm with a running time described by the recurrence equation T(N)=T(N-1)+C is:

- [x] O(N)
- [ ] None of the others
- [ ] O(logN)
- [ ] O(1)
- [x] O(N^2)



3. An algorithm with a running time described by the recurrence equation T(N)=T(N-1)+C is:

- [ ] Theta(N^2)
- [ ] Theta(1)
- [ ] Theta(logN)
- [x] None of the others
- [ ] Theta(N^3)

$T(n)$ is $\Theta(n)$

4. Assuming that T(0) is equal to a constant value, the solution of the recurrence equation T(N)=2*T(N-1)+C is:

- ( ) $T(N) = C_0$*N!+C_1
- ( ) $T(N) = C_0 * 2^N + C_1$
- (x) $T(N) = 2 * C_0 * N + C_1$
- ( ) $T(N) = C_0 * logN + C_1$
- ( ) None of the others

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

$= 3C + 2T(n-2)$

$= 5C + 2T(n-3)$

$= 7C + 2T(n-4)$

$= (2^k) - 1 +$

5. An algorithm with a running time described by the recurrence equation T(N)=2*T(N-1)+C is:

- [ ] $O(2^N)$
- [ ] $O(N^N)$
- [ ] O(logN)
- [ ] None of the others
- [ ] O(N^2)

6. An algorithm with a running time described by the recurrence equation T(N)=T(N-1)+N is

- [ ] $Theta(1)$
- [x] $Theta(N^2)$
- [ ] $Theta(N^3)$
- [ ] $Theta(N)$
- [ ] None of the others

$T(0) = C$

6) $T(n) = T(n-1) + N$

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

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

$= T(n-3) + kn$

$= T(n-k) + kn$

We want a constant where we have $T(n-k)$.

$k = n$

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

assume it's a constant $= T(0) + n^2$

$= C + n^2$

so $\Theta(n^2)$

7. An algorithm with a running time described by the recurrence equation $T(N) = T(N-1) - N^2$ is

- ☐ None of the others
- ☐ $Theta(N^3)$
- ☐ $Theta(N^2)$
- ☐ $Theta(N)$
- ☐ $Theta(1)$

Master Theorem

It's a method for solving recurrence equations, however it can only be applied to functions in the form of: T(n) = aT(n/b) + f(n), where a >= 1, and b > 1.

Example that can be solved using master theorem: T(n) = T(n/2) + n.

Can be applied because a = 1, and b > 1

Example that cannot be solved using master theorem: T(n) = 2T(n) + n

Cannot be applied because b=1, it's not striclty greater than 1.

How to use the Master Theorem

The master theorem classifies recurrence equations into one of three cases:

The form of the equation must be : $T(n) = aT(\frac{n}{b}) + f(n)$

In order to apply the Master Theorem, the recurrence equation must be of the form $T(n) = aT(n/b) + f(n)$ where $a \geq 1$ and $b > 1$.

When the Master Theorem can be applied, there are three cases to take into account:

1. $f(n) < n^{\log_b a}$

   In this case, $T(n) = \Theta(n^{\log_b a})$

2. $f(n) = n^{\log_b a}$

   In this case, $T(n) = \Theta(n^{\log_b a} \log n)$

3. $f(n) > n^{\log_b a}$

   For this case to be applicable, there is one extra requirement to be met: $a \cdot f(\frac{n}{b}) \leq c$, where $c < 1$ and n is large. In this case, $T(n) = \Theta(f(n))$

Applying the master theorem

Given: $T(n) = 2T(n/2) + n$

1) step 1: Identify values of a and b, check that they meet the restriction $a \geq 1$, $b > 1$.

$a = 2$, $b = 2$

2) Identify which case the equation satisfies:

a) Calculate $\log_b a = \log_2 2 = 1$

b) Now compare $f(n)$ to $n^{\log_b a}$

$f(n) = n$

$\log_b a = 1$

$n == n^1$

$\therefore T(n)$ is $\Theta(n^{\log_b a} \log n) = \Theta(n^1 \log n)$

$= \Theta(n \log n)$

Ex: $T(n) = \overset{a}{9}\, T(n/\underset{b}{3}) + n$

1) $a = 9$, $b = 3$, $f(n) = n$

2) $\log_b a = \log_3 9 = \log_3 3^2 = 2 \log_3 3 = 2(1) = 2$

$n^{\log_b a} = n^2$

$$\left.\begin{matrix} f(n) & < & n^{\log_b a} \\ n & < & n^2 \end{matrix}\right\} \quad T(n) = \Theta(n^2)$$

1. The solution to the recurrence equation T(N)=T(N/2)+1 is:

- [ ] $\Theta(N \log(N))$
- [ ] $Theta(log(N))$
- [ ] $\Theta(N^2)$
- [ ] None of the others
- [ ] $\Theta(N \log^2(N))$
- [ ] $\Theta(N^2 \log(N))$
- [ ] $\Theta(N)$

2. The solution to the recurrence equation T(N)=2*T(N/2)+N is:

- [ ] $Theta(N * log^2(N))$
- [ ] $Theta(N^2)$
- [ ] $Theta(N)$
- [ ] $Theta(log(N))$
- [ ] None of the others
- [ ] $\Theta(N * log(N))$
- [ ] $\Theta(N^2 \log(N))$

Ex: $T(n) = T\left(\underbrace{\frac{2n}{3}}\right) + 1$

1) $a = 1$, $b = 3/2$, $f(n) = 1$

$$\frac{n}{\underbrace{(3/2)}_{b}} = \frac{n}{1} \times \frac{2}{3} = \frac{2n}{3}$$

$\log_b a = \log_{3/2} 1 = 0 \quad \therefore n^0$

$f(n) = 1 \qquad n^{\log_b a} = n^0$

compare

$1 == 1 \quad \therefore T(n)$ is $\Theta(\log n)$

3. The solution to the recurrence equation T(N)=T(N/3)+1 is:

- [ ] $Theta(N^2)$
- [ ] $Theta(log(N))$
- [ ] $Theta(N)$
- [ ] $Theta(N * log^3(N))$
- [ ] $Theta(N^3)$
- [ ] None of the others
- [ ] $Theta(N * log(N))$

4. The solution to the recurrence equation T(N)=3*T(N/2)+N is:

- [ ] $Theta(N^{1.58})$
- [ ] None of the others
- [ ] $Theta(N^2)$
- [ ] $Theta(N^3)$
- [ ] $Theta(N)$
- [ ] $Theta(N^{2.32})$

5. The solution to the recurrence equation T(N)=5*T(N/2)+1 is:

- [ ] $Theta(N^3)$
- [ ] $Theta(N)$
- [ ] $Theta(N^{2.32})$
- [ ] None of the others
- [ ] $Theta(N^{1.58})$
- [ ] $Theta(N^2)$