- Complexity theory - 2
- Trees / Traversal
- C++

Complexity theory.

↳ Time taken by a given algorithm indep. of the underlying architecture / compiler / OS.

Bound.

Big-Oh → Upper bound.

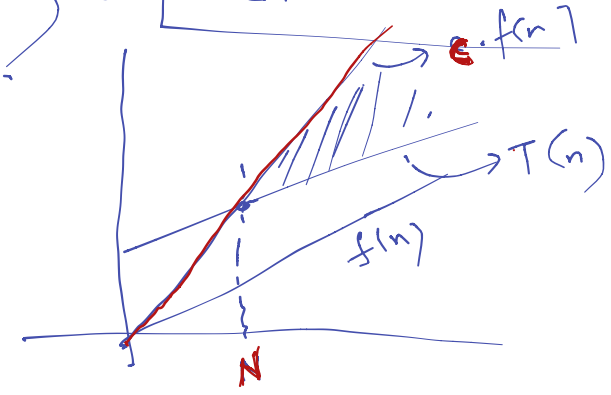
$$\log_a b = \frac{\log_{10} b}{\log_{10} a}$$

$$= \left(\frac{1}{\log_{10} a}\right) \cdot \log_{10} b$$

→ const.

$\bigcirc(\log_a b)$

$\bigcirc(\log_{10} b)$



Cautionary notes

* $n^2 \in O(n)$. Proof: Choose $c = n$ ~~×~~

$\therefore n^2 = c \cdot n = n$

WRONG: c must be a constant.

* Big-Oh notation does not always tell the whole story.

Eg $T_1(n) = n \log_2 n$ (Algo 1) → $O(n \log n)$

$T_2(n) = 100\,n$ (Algo 2) → $O(n)$

→ Algo2 is better?

$\log_2 n > 100$ only if $n > 2^{100}$.

$\log_2 n < 50$ in practice.

$\therefore$ Algo 1 is in fact better.

LOWER BOUND

$\Omega(f(n))$ is the set of all functions that satisfy: There exist positive constants c and N, such that for all $n \geq N$,

$$T(n) > c \cdot f(n)$$

$O(f(n)) \rightarrow T(n) \leq c \cdot f(n)$ UPPER BOUND

$\therefore \Omega$ is the reverse of O.

$\Rightarrow$ If $T(n) \in O(f(n))$ then

$f(n) \in \Omega(T(n))$ and vice versa.

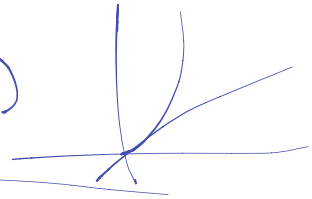
lg $\equiv \log_2$

$2n \in \Omega(n)$ because $n \in O(2n)$

$n^2 \in \Omega(n)$ because $n \in O(n^2)$

$n^2 \in \Omega(3n^2 + n \lg n)$,,

$3n^2 + n \lg n \in O(n^2)$



$\Omega \rightarrow$ At least this bad

$O \rightarrow$ At least this good.

If $T(n) \in O(f(n))$ and $\Omega(g(n))$

$T(n)$ is sandwitched between

$c \cdot f(n)$ and $d \cdot g(n)$

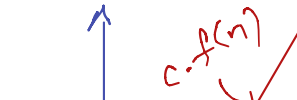
When $f(n) == g(n)$, $T(n) = \Theta(f(n))$

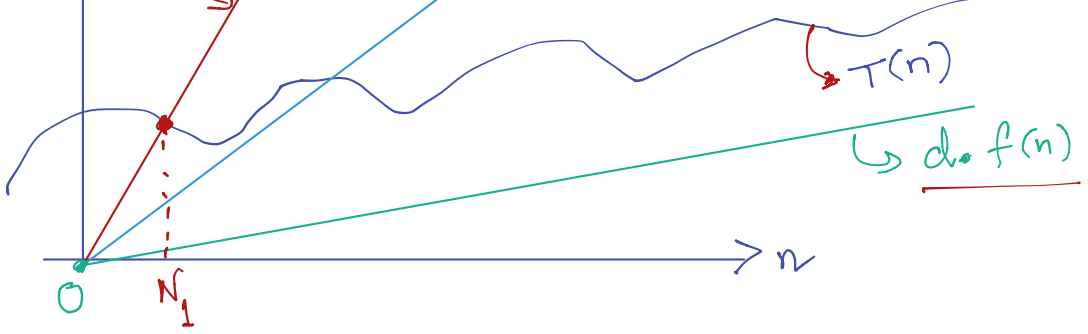
Big-Sigma $\equiv$ of the order of

Formally. $\Theta(f(n))$ is the SET of all functions $T(n)$ that are in BOTH $O(f(n))$ and $\Omega(f(n))$

How can you have matching upper and lower bounds? $\rightarrow$ Pick different values for c and d.

$f(n) = n$



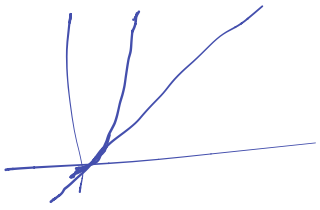


$$d \cdot f(n) \leq T(n) \leq c \cdot f(n)$$

↳ $n \geq N_1$

$$T(n) = \Theta(f(n)) = \Theta(n)$$

$T(n) = n$, $f(n) = n^2$



$T(n) \in O(n^2)$

$n \in \Omega(n^2)$ ✗

$f(n) = n^5$

If $f(n) \in \Theta(g(n))$ then $g(n) \in \Theta(f(n))$

$n^3 \in \Theta(3n^3 - n^2)$, $3n^3 - n^2 \in \Theta(n^3)$

$n^3 \notin \Theta(n)$, $n \notin \Theta(n^3)$

$n \in O(n^5)$ → Algo is slow. ✗

Big-theta → cannot be misleading.

$n \notin \Theta(n^5)$

Θ

Basic Rules for evaluating the efficiency of a given Algorithm.

1. FOR loops.

→ $T(n)$: at most the running time of

the statements inside the for loop TIMES the number of iterations.

```
for (int i=0; i<n; ++i) {
    S1
    S2
    :
}
```

→ Cost × n

→ $\Theta(1)$

$\Theta(n)$

(2) Nested loops

```
for (... n)
    for (... m)
        for (... p)
        {
        ...
        }
```

n × m × p.

n, n, n

$\Theta(n^3)$

Quadratic ≤ Above

(3) Consecutive statements → Additive

```
for (... n)
...
```

$\Theta(n)$

```
for (... n)
    for (... n)
    {
    }
```

$\Theta(n^2)$

$\Theta(n^2 + n) = \Theta(n^2)$

(4) If/Else. → Additive

```
if (...) {
```

(No recursion)

$\Theta(n^2 + n)$

$= \Theta(n^2)$

} else {

$S_2 \rightarrow \Theta(n)$

}.

(5) Recursive algorithm → Recurrence relationship in $T(n)$

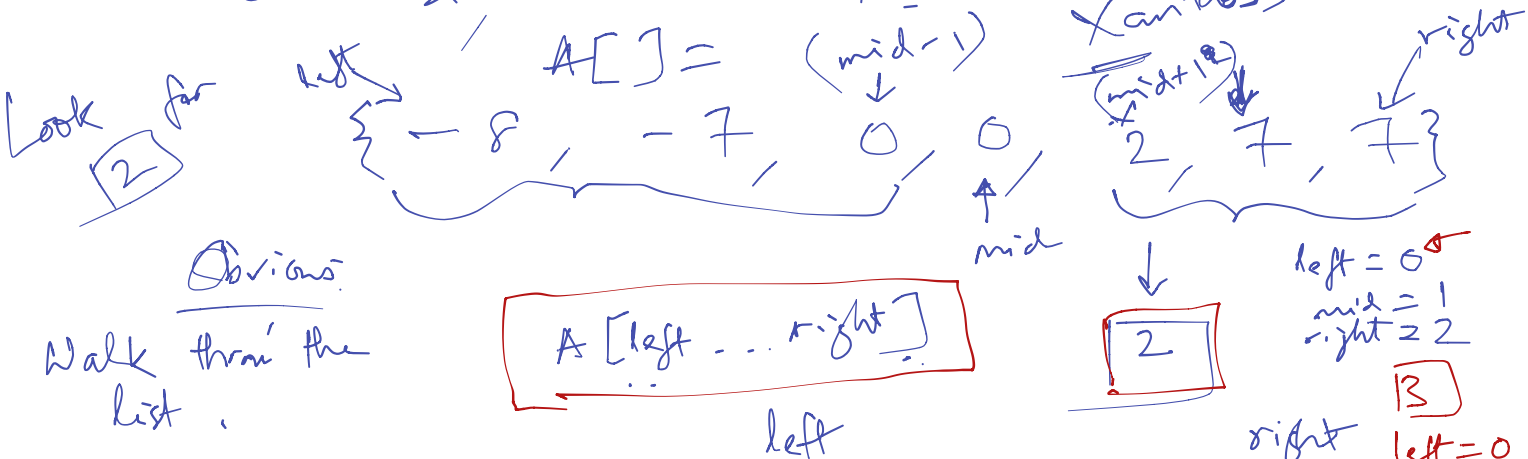
Binary Search

↳ Given a SORTED array (eg. Dictionary)

SEARCH for a value aVal

If we find aVal, return the array index

Otherwise return −1



Binary search.

Arguments: bsearch(A, left, right, aVal)

Base: Fail → (left > right) ⇒ out = −1

left = right + 1, right = 0

Pass → mid = (left + right)/2

(aVal == A[mid]) ⇒ out = mid

Recursion:

(aVal < A[mid]) ⇒

→ out = bsearch(A, left, mid−1, aVal)

(aVal > A[mid]) ⇒

$T(n)$ → out = bsearch(A, mid+1, right, aVal)

```
int bsearch (int * A, int left, int right, int aVal)
{
    if (left > right){
        return -1;          // Base 1        → Θ(1)
    }
    mid = (left + right) / 2;                 → Θ(1)
    if (aVal == A[mid]){
        return mid;         // Base 2         Θ(1)
    }
    if (aVal < A[mid]){
        return bsearch(A, left, mid-1, aVal);
    } else {
        return bsearch(A, mid+1, right, aVal);
    }
}
```

$T(\frac{n}{2})$ $T(\frac{n}{2})$ aVal = 2

Index: 0, 1, 2, 3, 4, 5, 6

A[]: -8, -7, 0, 0, 2, 7, 7

bsearch(A, 0, 6, 2)

↓

bsearch(A, 4, 6, 2)

↓

bsearch(A, 4, 4, 2) → FOUND (4)

↓

bsearch(A, 5, 4, 2) → NOT FOUND (-1)

aVal = 3

recursive

$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + \Theta(1)$

Loop → [illegible]

non-recursive

$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(1)$

$T(1) = \Theta(1)$

$= 2 \cdot \left(2 \cdot T\left(\frac{n}{2^2}\right) + \Theta(1)\right) + \Theta(1)$

$= 2^2 \cdot T\left(\frac{n}{2^2}\right) + \Theta(1) + \Theta(1)$

$= 2^2 \cdot \left(2 \cdot T\left(\frac{n}{2^3}\right) + \Theta(1)\right) + \Theta(1) + \Theta(1)$

$= 2^3 \cdot T\left(\frac{n}{2^3}\right) + \underbrace{\Theta(1) + \Theta(1) + \Theta(1)}_{3 \text{ times}}$

$= 2^k \cdot T\left(\frac{n}{2^k}\right) + \underbrace{\Theta(1) + \Theta(1) + \dots + \Theta(1)}_{k \text{ times}}$

$= 2^k \cdot T\left(\frac{n}{2^k}\right) + \Theta(k)$

smaller & smaller

$T(1)$ if $\frac{n}{2^k} = 1$

$\Rightarrow n = 2^k$

$\Rightarrow k = \log_2 n$

$T(1) = \Theta(1)$

$= n \cdot T(1) + \Theta(\log_2 n)$

$= n + \Theta(\log n)$

$T(n) = \Theta(n)$ → Sequential search

$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$

$$= \left(T\left(\frac{n}{2^2}\right) + \Theta(1)\right) + \Theta(1)$$

$$= T\left(\frac{n}{2^3}\right) + \underbrace{\Theta(1) + \Theta(1) + \Theta(1)}_{3 \text{ times}}$$

$$= T\left(\frac{n}{2^k}\right) + \Theta(k)$$

$$= T(1) + \Theta(\log_2 n) \quad \rightarrow 1$$

$$n = 2^k$$

$$\Rightarrow k = \log_2 n$$

$$= \Theta(1) + \Theta(\log_2 n)$$

$$\boxed{T(n) = \Theta(\log_2 n)} \rightarrow \text{Binary search.}$$