# Cost of Algorithms

**Inputs:** parameterized by an integer $n$, called the size

$T(I)$ = runtimes on input $I$ → runtime of a particular instance

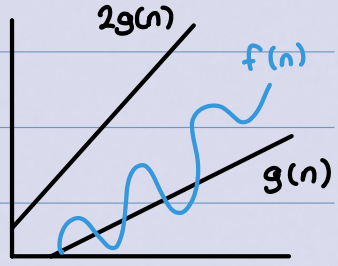
$T_{worst}(n) = \max_{I \text{ of size } n}(T(I))$ → worst-case runtime (default)

$T_{best}(n) = \min_{I \text{ of size } n}(T(I))$ → best-case runtime, not used much

## Asymptotic Notation

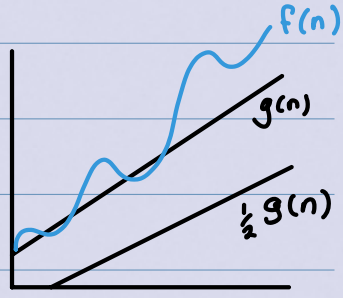
- consider 2 functions $f(n)$ and $g(n)$ with values in $\mathbb{R}_{>0}$

**big-O:** we say that $f(n) \in O(g(n))$ if there exists a $C > 0$ and $n_0$ such that for $n \geq n_0$, $f(n) \leq C\,g(n)$



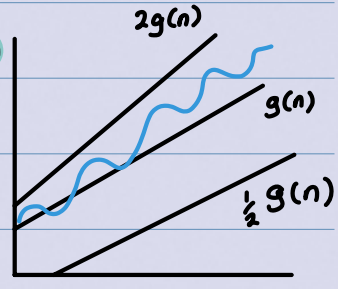

**big-Ω:** we say that $f(n) \in \Omega(g(n))$ if there exists a $C > 0$ and $n_0$ such that for $n \geq n_0$, $f(n) \geq C\,g(n)$

⇒ equivalent to $g(n) \in O(f(n))$



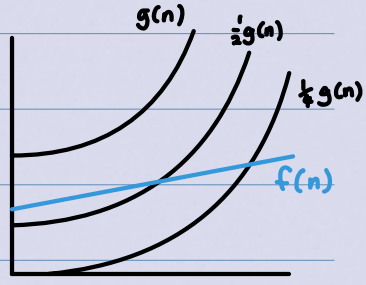

**Θ:** we say that $f(n) \in \Theta(g(n))$ if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.

⇒ in particular true if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C$ for some $0 < C < \infty$



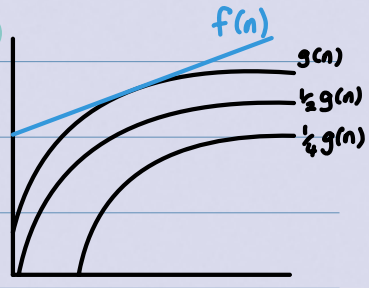

**little-o:** we say that $f(n) \in o(g(n))$ if for all $C > 0$, there exists an $n_0$ such that for $n \geq n_0$, $f(n) \leq C\,g(n)$

⇒ equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$



**little-ω:** we say that $f(n) \in \omega(g(n))$ if for all $c > 0$, there exists an $n_0$ such that for $n \geq n_0$, $f(n) \geq cg(n)$

⇒ equivalent to $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$




## Examples

a) $n^k + c_{k-1}n^{k-1} + \ldots + c_0 \in \Theta(n^k)$

b) $2^{n-1} \notin \Theta(2^n)$

c) $(n-1)! \in \Theta(n!)$

## Definitions for Several Parameters

consider two functions $f(n,m)$, $g(n,m)$ with values in $\mathbb{R}_{>0}$

$f(n,m)$ is in $O(g(n,m))$ if there exist $c, n_0, m_0$ such that $f(n,m) \leq Cg(n,m)$ for $n \geq n_0$ or $m \geq m_0$

## Case Study: Maximum Subarray

Given an array $A[0,\ldots,n]$, find a contiguos subarray $A[i,\ldots,j]$ that maximises the sum $A[i] + \ldots + A[j]$.

Example: given $A = [10, -5, 4, 3, -5, 6, -1, -1]$, the subarray $A[0,\ldots,5] = [10, -5, 4, 3, -5, 6]$ has sum $10-5+4+3-5+6=13$ is the max.

**Brute Force algorithm:**

runtime is $\Theta(n^3)$

but, we can improve on this!
idea: we recompute the same sum many times in the j loop

```
BruteForce (A)
1. opt = 0
2. for (i=0→n) {
3.     for (j=i→n) {
4.         sum = 0
5.         for (k=i→j) {
6.             sum += A[k]
```

```
BetterBruteForce(A)
1. opt = 0
2. for (i=0→n) {
3.   sum = 0
4.   for (j=1→n) {
5.     sum += A[j]
6.     if (sum > opt) {
7.       opt = sum
8.     }
9.   }
10. }
11. return opt
```

```
7.     }
8.     if (sum > opt) {
9.       opt = sum
10.    }
11.  }
12. }
13. return opt
```

⇒ runtime $O(n^2)$

but, we can still do better using a **divide-and-conquer approach**:

idea: solve the problem twice in size $n/2$ (assuming $n$ is a power of 2). Then, the **optimal subarray** (if not empty):

1. is completely **in the left half** $A[0, \dots, n/2]$
2. or is completely **in the right half** $A[n/2+1, \dots, n]$
3. or **contains both** $A[n/2]$ **and** $A[n/2+1]$

⇒ the three cases are mutually exclusive

to find the optimal subarray in **case 3**, we have:

$$A[i] + \dots + A[n/2] + A[n/2+1] + \dots + A[j]$$

more abstractly, we have $F(i,j) = f(i) + g(j)$, for $i \in [0, \dots, n/2]$ and $j \in [n/2+1, \dots, n]$.

To maximise $F(i,j)$, we **maximise $f(i)$ and $g(j)$ independently!**

```
MaximiseLowerHalf(A)
1. opt = A[n/2]
```

```
MaximiseUpperHalf(A)
1. opt = A[n/2+1]
```

```
2. sum = A[n/2]
3. for (i = n/2 - 1 → 0) {
4.     sum += A[i]
5.     if (sum > opt) {
6.         opt = sum
7.     }
8. }
9. return opt
```

```
2. sum = A[n/2 + 1]
3. for (i = n/2 + 1 → n) {
4.     sum += A[i]
5.     if (sum > opt) {
6.         opt = sum
7.     }
8. }
9. return opt
```

↳ runtimes are $\Theta(n)$! ↵

So, final divide and conquer algorithm:

```
DivideAndConquerMaximumSubarray(A[0, ..., n])
1. if (n == 1) return max(A[0], 0)
2. opt_low = DivideAndConquerMaximumSubarray(A[0, ..., n/2])
3. opt_high = DivideAndConquerMaximumSubarray(A[n/2 + 1, ..., n])
4. opt_mid = MaximiseLowerHalf(A) + MaximiseUpperHalf(A)
5. return max(opt_low, opt_mid, opt_high)
```

⇒ runtime: $T(n) = 2T(n/2) + \Theta(n) \in \Theta(n \log n)$

## Solving Recurrences

Consider a recursive algorithm called Algo.

Assumption: for any input of size $n \geq n_0$, Algo does:

- $a$ recursive calls, in size either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$ ($a > 0$, $b > 1$, constant)
- between $c'n^y$ and $cn^y$ extra operations ($c, c' \neq 0$, $y$ constant)

Claim: Solving the sloppy recurrence $T(n) = aT(n/b) + cn^y$ for powers of $b$ gives a valid $\Theta$-bound for best and worst-

case runtimes.

- Remark: if we only know that we do at most $cn^3$ extra operations, we only get a big-O bound.

## Best and Worst Case Recurrence Relations

let $T_{worst}(n)$, $T_{best}(n)$ be the worst / best cases in size $n$.

worst-case recurrence: $T_{worst}(1) = d$, and

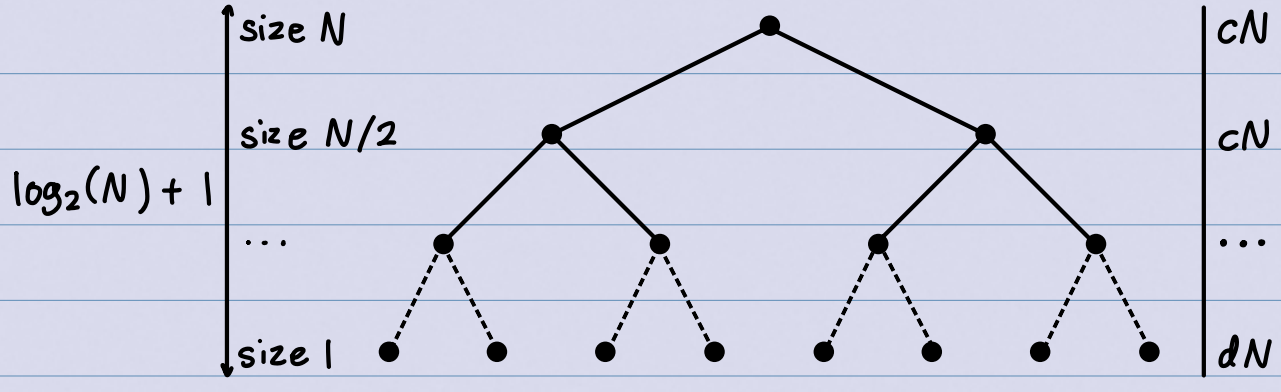
$$T_{worst}(n) \leq T_{worst}(\lceil n/2 \rceil) + T_{worst}(\lfloor n/2 \rfloor) + cn \quad \text{if } n>1$$

best-case recurrence: $T_{best}(1) = d'$, and

$$T_{best}(n) \geq T_{best}(\lceil n/2 \rceil) + T_{best}(\lfloor n/2 \rfloor) + cn \quad \text{if } n>1$$

we define $N$ as the next power of 2 of $n$, so $N < 2n$.

## the mergesort recursion tree




total: $t(N) = cN\log_2(N) + dN \leq KN\log_2(N)$ for $N$ a power of 2

consequence: $T_{worst}(n) \leq KN\log_2(N) \leq K(2n)\log_2(2n) \leq K' n \log_2 n$,

so, $T_{worst}(n) \in O(n \log n)$

remark: same approach proves $T_{best}(n) \in \Omega(n \log n)$, and so,

$T_{best}(n)$, $T_{worst}(n) \in \Theta(n \log n)$

## The Master Theorem

Suppose that $a \geq 1$ and $b > 1$. Consider the recurrence

$$T(n) = aT(n/b) + cn^y \quad (n \geq b), \qquad T(1) = d.$$

Let $x = \log_b a$ (so $a = b^x$).

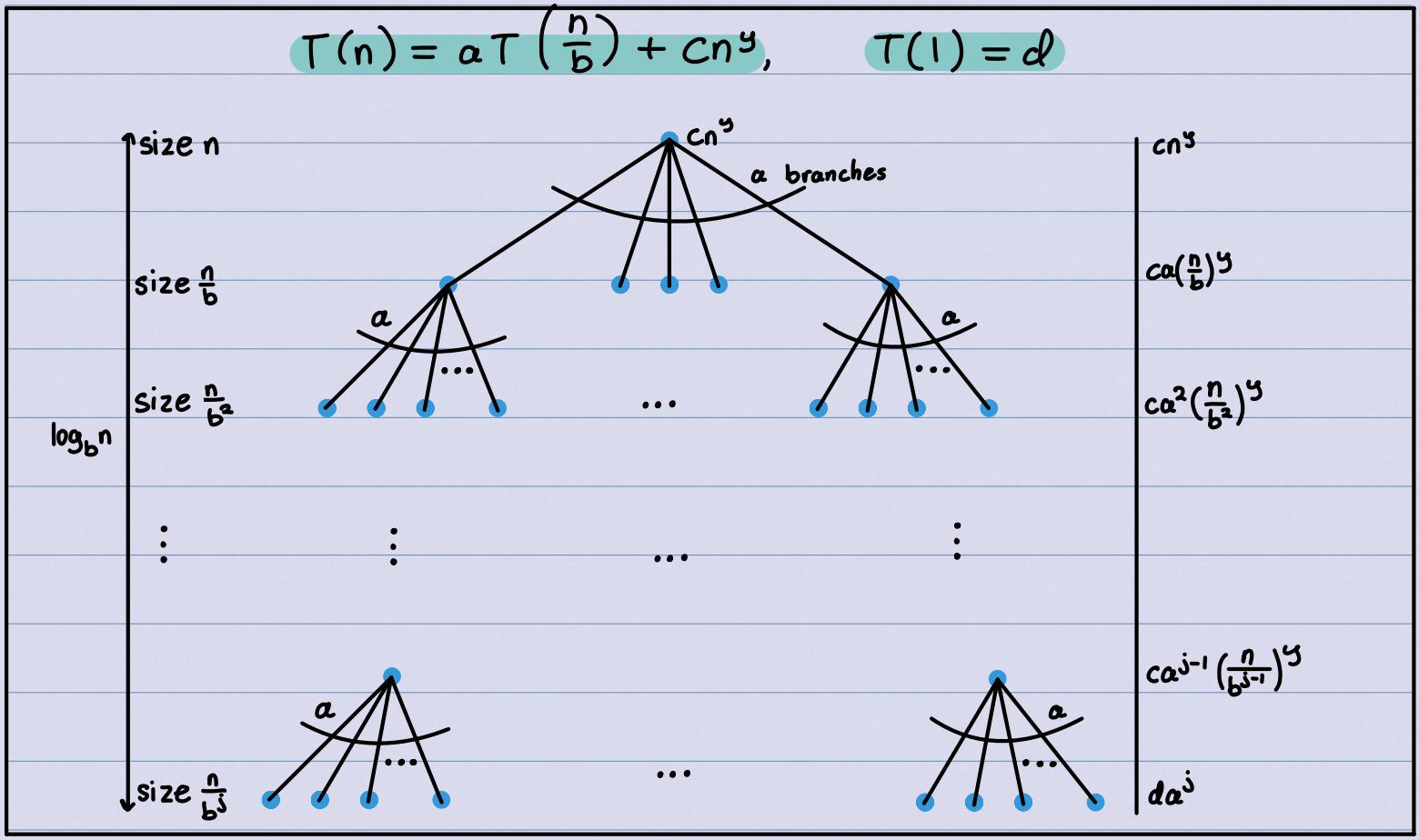
Then, for $n$ a power of $b$,

$$T(n) \in \begin{cases} \Theta(n^y) & \text{if } y > x \quad \text{root heavy} \\ \Theta(n^y \log n) & \text{if } y = x \quad \text{balanced} \\ \Theta(n^x) & \text{if } y < x \quad \text{leaf heavy} \end{cases}$$

## Recursion Tree

Suppose that $n = b^j$, $a \geq 1$, $b \geq 2$ are integers and

$$T(n) = aT(n/b) + cn^y, \qquad T(1) = d$$



## Breakdown of the Cost

Suppose that $a \geq 1$ and $b \geq 2$ are integers and

$$T(n) = aT(n/b) + cn^y, \quad T(1) = d.$$

Let $n = b^j$.

| Size of subproblem | # nodes | cost/node | total cost |
|---|---|---|---|
| $n = b^j$ | $1$ | $cn^y$ | $cn^y$ |
| $n/b = b^{j-1}$ | $a$ | $c(n/b)^y$ | $ca(n/b)^y$ |
| $n/b^2 = b^{j-2}$ | $a^2$ | $c(n/b^2)^y$ | $ca^2(n/b^2)^y$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $n/b^{j-1} = b$ | $a^{j-1}$ | $c(n/b^{j-1})^y$ | $ca^{j-1}(n/b^{j-1})^y$ |
| $n/b^j = 1$ | $a^j$ | $d$ | $da^j$ |
| generally: $n/b^i = b^{j-i}$ | $a^i$ | $c(n/b^i)^y$ | $ca^i(n/b^i)^y$ |

## Computing $T(n)$:

①: $x = \log_b a$, so $a = b^x$. we know $n = b^j$. $\therefore$, $a^j = (b^x)^j = (b^j)^x = n^x$.

total: $T(n) = da^j + cn^y \sum_{i=0}^{j-1} \left(\frac{a}{b^y}\right)^i \overset{\text{using ①}}{=} dn^x + cn^y \sum_{i=0}^{j-1} \left(\frac{a}{b^y}\right)^i$

observation: geometric sum with ratio $r = \frac{a}{b^y} = \frac{b^x}{b^y} = b^{x-y}$!

- if $r < 1$, then $b^{x-y} < 1$, so $x < y$.

  we know that $\sum_{i=0}^{j-1} r^i \in \Theta(1)$ if $r < 1$.

  $\therefore$, $T(n) = dn^x + cn^y \times \Theta(1) \in \Theta(n^y)$. $\therefore$, $x < y \Rightarrow T(n) \in \Theta(n^y)$

- if $r=0$, then $b^{x-y}=1$, so $x=y$.
  we know that $\sum_{i=0}^{j-1} r^i \in \mathcal{O}(j) = \mathcal{O}(\log n)$ if $r=1$
  $\therefore$, $T(n) = dn^x + cn^y \times \mathcal{O}(\log n) \in \mathcal{O}(n^y \log n)$ $\therefore$, $x=y \Rightarrow T(n) \in \mathcal{O}(n^y \log n)$

- if $r>1$, then $b^{x-y} > 1$, so $x>y$
  we know that $\sum_{i=0}^{j-1} r^i \in \mathcal{O}(r^j)$ if $r>1$
  $\therefore$, $T(n) = dn^x + cn^y \times \mathcal{O}(r^j)$. $r^j = a^j/b^{yj} = n^x/n^y$.
  $T(n) = dn^x + cn^y \times \mathcal{O}(n^x/n^y) \in \mathcal{O}(n^x)$. $\therefore$, $x>y \Rightarrow T(n) \in \mathcal{O}(n^x)$.

Examples: find $\mathcal{O}$-bounds for the following:

1) $T(n) = 4T(n/2) + n$ (multiplying polynomials)
   $a=4$, $b=2$, $y=1$, so $x = \log_b a = \log_2 4 = 2$.

   $\therefore$, $x=2 > y=1$, so $T(n) \in \mathcal{O}(n^2)$

2) $T(n) = 2T(n/2) + n^2$ (kd-trees)
   $a=2$, $b=2$, $y=2$, so $x = \log_b a = \log_2 2 = 1$

   $\therefore$, $x=1 < y=2$, so $T(n) \in \mathcal{O}(n^2)$

3) $T(n) = 2T(n/4) + 1$ (kd-trees)
   $a=2$, $b=4$, $y=0$, so $x = \log_b a = \log_4 2 = 1/2$

   $\therefore$, $x=1/2 < y=0$, so $T(n) \in \mathcal{O}(\sqrt{n})$

4) $T(n) = T(n/2) + 1$ (binary search)
   $a=1$, $b=2$, $y=0$, so $x = \log_b a = \log_2 1 = 0$

$\therefore$, $x = 0 = y$, so $T(n) \in \Theta(n^0 \log n) = \Theta(\log n)$

5) $T(n) = T(n/2) + n$ (amortized analysis of dynamic arrays)

$a = 1$, $b = 2$, $y = 1$, so $x = \log_b a = \log_2 1 = 0$

$\therefore$, $x = 0 < y = 1$, so $T(n) \in \Theta(n)$

Alternative: guess and prove: $T(n) = 2T(n/2) + n$, $T(1) = 0$.

guess $T(n) \leq n$. Then $T(n/2) \leq n/2$.

$T(n) = 2T(n/2) + n \leq 2(n/2) + n = 2n \not\leq n$.

guess $T(n) \leq kn$ for some $k$. Then $T(n/2) \leq kn/2$.

$T(n) = 2T(n/2) + n \leq 2(kn/2) + n = kn + n \not\leq kn$

guess $T(n) \leq kn\log_2 n$ for some $k$. Then $T(n/2) \leq kn\log_2(n/2)$

$T(n) = 2T(n/2) + n \leq 2\left(\frac{kn\log_2(n/2)}{2}\right) + n = kn\log_2 n - kn + n \leq n$ for $k \geq 1$.

$\therefore$, $T(n) \leq kn\log n$ for $k \geq 1$, so $T(n) \in O(n\log n)$.

(proving $T(n) = \ldots$ is harder)