How fast is fast?

Order of magnitude analysis

Bruce Merry

Definition of big-O

Let $f(n)$ and $g(n)$ be two functions. We say that

$$f(n) = \mathrm{O}(g(n))$$

if and only if there exist constants C and N such that

$$f(n) \leq Cg(n) \text{ for all } n \geq N.$$

If an algorithm takes $f(n)$ seconds to run on an input of size n, then it is said to be $\mathrm{O}(g(n))$.

Examples

- Constant factors are irrelevant: $3n^2 = \mathrm{O}(n^2)$.

Examples

- Constant factors are irrelevant: $3n^2 = \mathrm{O}(n^2)$.
- In a sum, only the larger term is relevant: $n^2 + n \cdot \log n = \mathrm{O}(n^2)$.

Examples

- Constant factors are irrelevant: $3n^2 = O(n^2)$.
- In a sum, only the larger term is relevant: $n^2 + n \cdot \log n = O(n^2)$.
- Only the dominant term in a polynomial is relevant: $3n^2 + 4n + 2 = O(n^2)$.

Examples

- Constant factors are irrelevant: $3n^2 = \mathrm{O}(n^2)$.
- In a sum, only the larger term is relevant: $n^2 + n \cdot \log n = \mathrm{O}(n^2)$.
- Only the dominant term in a polynomial is relevant: $3n^2 + 4n + 2 = \mathrm{O}(n^2)$.
- It is only an upper bound: $n = \mathrm{O}(n^2)$, although one usually gives the tightest possible bound.

Examples

- Constant factors are irrelevant: $3n^2 = \mathrm{O}(n^2)$.
- In a sum, only the larger term is relevant: $n^2 + n \cdot \log n = \mathrm{O}(n^2)$.
- Only the dominant term in a polynomial is relevant: $3n^2 + 4n + 2 = \mathrm{O}(n^2)$.
- It is only an upper bound: $n = \mathrm{O}(n^2)$, although one usually gives the tightest possible bound.
- It is a worst case bound: linear search is $\mathrm{O}(n)$, even though it might only take one step.

Rule of thumb

To estimate the running time of your program

- Substitute the data set size into the big-O formula
- Divide by a scale factor for the machine (10–100 million per second)
- Estimating the scale factor is a bit of an art.

Examples

```
for i := 1 to N do
    for j := 1 to N − i do
        if a[j] > a[j + 1] then
            swap(a[j], a[j + 1]);
```

Examples

```
for i := 1 to N do
    for j := 1 to N − i do
        if a[j] > a[j + 1] then
            swap(a[j], a[j + 1]);
```

Answer: $O(n^2)$.

Examples

```
for i := 1 to N do
    for j := 1 to N − i do
        if a[j] > a[j + 1] then
            swap(a[j], a[j + 1]);
```

Answer: $O(n^2)$.

```
K := 0;
for i := 1 to N do
    while (K < i) and (bad[i][i − K]) do K := K + 1;
```

Examples

```
for i := 1 to N do
    for j := 1 to N − i do
        if a[j] > a[j + 1] then
            swap(a[j], a[j + 1]);
```

Answer: $O(n^2)$.

```
K := 0;
for i := 1 to N do
    while (K < i) and (bad[i][i − K]) do K := K + 1;
```

Answer: $O(n)$.

Divide and conquer

```
left  := 1;
right  := N;
while ( left  <  right )  do
begin
      middle  := ( left  +  right )  div 2;
      if  a[middle] < target  then  left  :=  middle + 1
      else  right  :=  middle;
end;
```

Divide and conquer

```
left := 1;
right := N;
while ( left < right ) do
begin
    middle := ( left + right ) div 2;
    if a[middle] < target then left := middle + 1
    else right := middle;
end;
```

Answer: $O(\log n)$.

Divide and conquer

```
procedure recurse(left, right : integer);
begin
    for i := left to right do do_something_constant(i);
    middle := ( left + right ) div 2;
    if ( left < middle) then recurse(left , middle);
end;
```

Divide and conquer

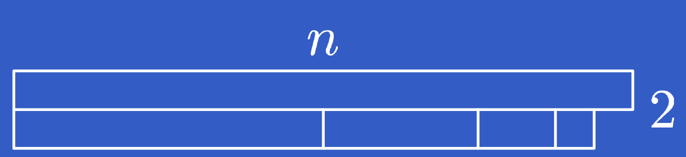
```
procedure recurse(left, right : integer);
begin
    for i := left to right do do_something_constant(i);
    middle := ( left + right ) div 2;
    if ( left < middle) then recurse(left , middle);
end;
```

Answer: $O(n)$.

Divide and conquer

```
procedure recurse(left, right : integer);
begin
    for i := left to right do do_something_constant(i);
    middle := ( left + right ) div 2;
    if ( left < middle) then recurse(left , middle);
end;
```

Answer: $O(n)$.



Divide and conquer

```
procedure recurse(left, right : integer);
begin
    for i := left to right do do_something_constant(i);
    middle := ( left + right ) div 2;
    if ( left < middle) then recurse(left , middle);
    if (middle + 1 < right ) then recurse(middle + 1, right );
end;
```

Divide and conquer

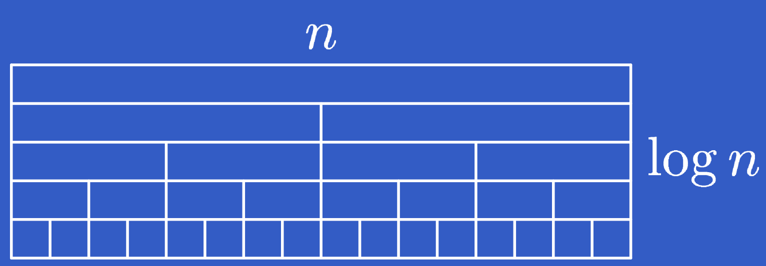
```
procedure recurse(left, right : integer);
begin
    for i := left to right do do_something_constant(i);
    middle := ( left + right ) div 2;
    if ( left < middle) then recurse(left , middle);
    if (middle + 1 < right ) then recurse(middle + 1, right );
end;
```

Answer: $O(n \cdot \log n)$.

Divide and conquer

```
procedure recurse(left, right : integer);
begin
    for i := left to right do do_something_constant(i);
    middle := ( left + right ) div 2;
    if ( left < middle) then recurse(left , middle);
    if (middle + 1 < right ) then recurse(middle + 1, right );
end;
```

Answer: $O(n \cdot \log n)$.



Average case

It is occasionally the case that the average case performance is much better than the worst case performance.

- Binary search tree insertion: $\mathrm{O}(\log n)$ on average, but $\mathrm{O}(n)$ in the worst case.
- Quicksort is $\mathrm{O}(n \cdot \log n)$ on average, but $\mathrm{O}(n^2)$ in the worst case.

However, this assumes a random distribution.

How to solve a problem

1. Look at the constraints
2. Cook up some algorithms and evaluate
3. Estimate your score from the constraints
4. Pick the **simplest** algorithm you can
5. Remember to leave time to code and debug

Other lessons

- Don't optimise unless you need to.
- Don't bother optimising the fast bits.
- Don't bother optimising the outer loops.
- Don't ignore an optimisation just because it doesn't affect the big O — it can still make a big difference.
- $O(\log n)$ is much closer to $O(1)$ than to $O(n)$

Questions

?