Data Structure

A data structure is an organization of data in memory or on disk, done for reasons of efficiency.

Efficiency can refer to either time or space. We will mostly concern with time.

Q: How do we concern with time?

A: We usually try to avoid using wall-clock time so instead we count the number of simple operations or steps.

Ex.

For(i = 0; i < n; i ++)

a[i] = n;

Fortunately, we only care about the behavior as n gets large. So, consider the asymptotic behavior. Our example is linear in n. we actually have a notation to describe what we want: BIG-O notation. To compute the big-o runtime of a mathematical function, do the following.

1. Disregard all terms except the fastest growing term 3n+2🡪3n
2. Disregard any coefficient on the remaining term 3n🡪n

* We say that 3n+2 is O(n) [big-o of n]

Ex.

13+n 🡪 n

4n+6n^2+4 🡪 3n^2 🡪 n^2

3n^2-4logn+1000 🡪 3n^2 🡪 n^2

10nlogn+3n+60 🡪 10nlogn 🡪 nlogn

Common BIG-O complexity classes

Constant O(1)

Logarithmic O(n)

Linear O(logn)

Linear-logarithmic O(nlogn)

Quadratic O(n^2)

Cubic O(n^3)

Polynomial O(n^k)

Exponential O(2^n)

Factorial O(n!)

Applying big-o notation to node

for (i = 0; i < n; i ++) O(n)

x ++;

for (i = 0; i < n; i = i+2) O(n)

x ++;

for (i = 0; i < 3\*n; i ++) O(n)

x ++;

for (i = n; i > 0; i --) O(n)

x ++;

for (i = 1; i <= n; i = i\*2) O(logn) if n = 8, i = 1

x ++; i = 1,2,4,8,16

if n = 16, i = 1

i = 1,2,4,8,16,32

if n = 32, i = 1

i = 1,2,4,8,16,32,64

for (i = n; i > 0; i = i/3) O(logn) if n = 8 🡪 i = 8,2,0

x++; if n = 16 🡪 i = 16,5,1,0

if n = 32 🡪 i = 32,10,3,1,0

for ( i = 0; i < n; i ++) O(n^2)🡪independent loop

for( j = 0; j < n; j ++)🡪O(n)

x++;

for (i = 1; i <= n; i ++) O(n^2)🡪independent loop

for(j = 1; j < n/2; j ++)

x++;

for (i = 1; i < = n; i ++) O(n^2)🡪dependent loop

for (j = 1; j < 1; j ++) dependent quadratic

x++; loops: total # of steps = n(n+1)/2

for (i = 0; i < n; i ++) O(n^2)🡪dependent loops

for(j = i; j <n; j = j+2)

x++;