Algorithms: Complexity (Big Oh) of the codes

The RAM Model of Computation

Algorithms are an important and durable part of computer science because they can be studied in a machine/language independent way.
This is because we use the RAM model of computation for all our analysis.

- Each "simple" operation (+, -, =, if, call) takes 1 step.
- Loops and subroutine calls are *not* simple operations. They depend upon the size of the data and the contents of a subroutine. "Sort" is not a single step operation.
- Each memory access takes exactly 1 step.

We measure the run time of an algorithm by counting the number of steps.
This model is useful and accurate in the same sense as the flat-earth model (which *is* useful)!

Running Time/Cost of codes and Big Oh

- Assumption
  - Each operation will take same time

```
for(int i=0; i<5; i++)
    printf("%d", i);
```

$c1 = 1 + 6 + 5 = 12$

$c2 = 5*1 = 5$

Total $\mathbf{c = c1 + c2} = 12 + 5 = 17$

$= \mathbf{O(n^0)} = \mathbf{O(1)}$

Running Time/Cost of codes and Big Oh

- Assumption
  - Each operation will take same time

```
for(int i=0; i<n; i++)
    printf("%d", i);
```

$c1 = 1 + (\mathbf{n+1}) + n = 2n + 2$

$c2 = 1*\mathbf{n} = n$

Total $\mathbf{c = c1 + c2} = 3n + 2 = \mathbf{O(n)}$

Running Time/Cost of codes and Big Oh

- Assumption
  - Each operation will take same time

```
for(int i=0; i<6; i++)
    for(int k=0; k<5; k++)
        printf("%d", i+k);
```

c1 =

c2 =

c3 =

Running Time/Cost of codes and Big Oh

- Assumption
  - Each operation will take same time

```
for(int i=0; i<6; i++)
    for(int k=0; k<5; k++)
        printf("%d", i+k);
```

c1 = 1 + **(6+1)** + 6 = 12

c2 = **6***1 + **6***(5+1) + **6***5 = 72

c3 = 6*5*1 = 30

Total **c = c1 + c2 + c3**

= 12 + 72 + 30 = 114

= **O(n^0) = O(1)**

Running Time/Cost of codes and Big Oh

- Assumption
  - Each operation will take same time

```
for(int i=0; i<n; i++)
    for(int k=0; k<n; k++)
        printf("%d", i+k);
```

c1 = 1 + (n+1) + n = 2n + 2

c2 = n*1 + n*(n+1) + n*n = n*(2n + 2)

c3 = (n*n)*1 = n*n

Total **c = c1 + c2 + c3** = (2n+2) + n*(2n+2) + n*n

= 3*n*n + 3*n + 2

= **O(n*n)**

Running Time/Cost of codes and Big Oh

- Assumption
  - Each operation will take same time

```
for(int i=0; i<m; i++)
    for(int k=0; k<n; k++)
        printf("%d", i+k);
```

c1 = 1 + (m+1) + m = 2m + 2

c2 = m*1 + m*(n+1) + m*n = m*(2n + 2)

c3 = (m*n)*1 = m*n

Total **c = c1 + c2 + c3** = (2m+2) + m*(2n+2) + m*n

= 3*m*n + 3*m + 2

= **O(m*n)**

Asymptotic Analysis

High-level idea: Suppress constant factors and lower-order terms

too system-dependent

irrelevant for large inputs

Example: Equate $6n \log_2 n + 6$ with just $n \log n$.

Terminology: Running time is $O(n \log n)$

["big-Oh" of $n \log n$]

where n = input size (e.g. length of input array).

Example: One Loop

Problem: Does array A contain the integer t? Given A (array of length n) and t (an integer).

Algorithm 1

1: **for** $i = 1$ to n **do**
2:     **if** $A[i] == t$ **then**
3:         Return TRUE
4: Return FALSE

Question: What is the running time?

Example: One Loop

Problem: Does array A contain the integer t? Given A (array of length n) and t (an integer).

Algorithm 1

```
for i = 1 to n do
    if A[i] == t then
        Return TRUE
Return FALSE
```

Question: What is the running time?

A) $O(1)$ C) $O(n)$

B) $O(\log n)$ D) $O(n^2)$

Example: Two Loops

Given A, B (arrays of length n) and t (an integer). [Does A or B contain t?]

Algorithm 2

for $i = 1$ to n **do**
    if $A[i] == t$ **then**
        Return TRUE
for $i = 1$ to n **do**
    if $B[i] == t$ **then**
        Return TRUE
Return FALSE

Question: What is the running time?

Example: Two Loops

Given A, B (arrays of length n) and t (an integer). [Does A or B contain t?]

Algorithm 2

```
for i = 1 to n do
    if A[i] == t then
        Return TRUE
for i = 1 to n do
    if B[i] == t then
        Return TRUE
Return FALSE
```

Total **c** = **O(n)**

Total **c** = **O(n)**

Question: What is the running time?

A) $O(1)$
C) $O(n)$
B) $O(\log n)$
D) $O(n^2)$

= O(n) + O(n)

Example: Two Nested Loops

Problem: Do arrays A, B have a number in common? Given arrays A, B of length n.

Algorithm 3

for $i = 1$ to n **do**
    for $j = 1$ to n **do**
        if $A[i] == B[j]$ **then**
            Return TRUE
Return FALSE

Question: What is the running time?

Example: Two Nested Loops

Problem: Do arrays A, B have a number in common? Given arrays A, B of length n.

Algorithm 3

```
for i = 1 to n do
    for j = 1 to n do
        if A[i] == B[j] then
            Return TRUE
Return FALSE
```

Question: What is the running time?

A) $O(1)$ C) $O(n)$

B) $O(\log n)$ D) $O(n^2)$

Example: Two Nested Loops (II)

Problem: Does array A have duplicate entries? Given arrays A of length n.

Algorithm 4

for $i = 1$ to n **do**
    for $j =$ i+1 to n **do**
        if $A[i] ==$ A $[j]$ **then**
            Return TRUE
Return FALSE

Question: What is the running time?

Example: Two Nested Loops (II)

Problem: Does array A have duplicate entries? Given arrays A of length n.

Algorithm 4

for $i = 1$ to n **do**
  for $j =$ i+1 to n **do**
    if $A[i] ==$ A $[j]$ **then**
      Return TRUE
Return FALSE

Question: What is the running time?

A) $O(1)$ C) $O(n)$

B) $O(\log n)$ D) $O(n^2)$