Time Complexity

Sandra Batista, Mark Redekopp, and David Kempe
Time Complexity Analysis

To find upper or lower bounds on the complexity, we must consider the set of all possible inputs, I, of size, n

Derive an expression, T(n), in terms of the input size, n, for the number of steps required to solve the problem on a given input, i, of size n.
Time Complexity Analysis

Case Analysis is when you determine which input must be used to define the runtime function, $T(n)$, for inputs of size $n$

**Best-case analysis**: Find the input of size $n$ that takes the **minimum** amount of time.

**Average-case analysis**: Find the time for all inputs of size $n$ and take the average of the times. (Assume a distribution over the inputs although uniform is a reasonable choice.)

**Worst-case analysis**: Find the input of size $n$ that takes the **maximum** amount of time.
Steps for Performing Runtime Analysis

When we perform **worst-case analysis** in determining the runtime on inputs of size n:

1. Find at least one input of size n that will require the **maximum** runtime of the algorithm.

2. Using that input, express the runtime of the algorithm (on that input case) as a function of n, $T(n)$.

3. Apply asymptotic notation to find the order of growth of the runtime function, $T(n)$.
Asymptotic Notation

\( T(n) \) is said to be \( O(f(n)) \) if…

- \( T(n) < a \cdot f(n) \) for \( n > n_0 \) (where \( a \) and \( n_0 \) are constants greater than 0)

\( T(n) \) is said to be \( \Omega(f(n)) \) if…

- \( T(n) > a \cdot f(n) \) for \( n > n_0 \) (where \( a \) and \( n_0 \) are constants greater than 0)

\( T(n) \) is said to be \( \Theta(f(n)) \) if…

- \( T(n) \) is both \( O(f(n)) \) AND \( \Omega(f(n)) \)
Worst Case and Big-Ω

Big-Ο for the **worst-case: no possible** inputs can exceed this runtime bound (at-most or upper bound)

Big-Ω for the **worst-case: there exists at least one input** requiring at least this bound for runtime (at-least or lower bound) for the worst case

To arrive at Ω(f(n)) for the **worst-case** requires you simply to find AN input case (i.e. the worst case) that requires at least f(n) steps

```c
int i; j;
for(i=0; i < n; i++){
    if(a[i][0] == 0){
        for(j=0; j<n; j++)
            {a[i][j] = i*j;
            }
    }
}
```
Steps for Deriving $T(n)$

Considering an input of size $n$ that requires the maximum runtime

1. Trace through each line of the algorithm or code. Assume elementary operations such as incrementing a variable occur in constant time.
2. If sequential blocks of code have runtime $T_1(n)$ and $T_2(n)$ respectively, then their total runtime will be their sum $T_1(n) + T_2(n)$.
3. For loops, sum the runtime for each iteration of the loop, $T_i(n)$, to get the total runtime for the loop.
Helpful Common Summations

- Arithmetic series: \( \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \theta(n^2) \)
- General form of the arithmetic series: \( \sum_{i=1}^{n} \theta(i^p) = \theta(n^{p+1}) \)
- Geometric series: \( \sum_{i=1}^{n} c^i = \frac{c^{n+1}-1}{c-1} = \theta(c^n) \)
- Harmonic series: \( \sum_{i=1}^{n} \frac{1}{i} = \theta(\log n) \)
Deriving $T(n)$

Derive an expression, $T(n)$, in terms of the input size for the number of operations/steps that are required to solve a problem.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int i = 0;
    x = 5;
    if(i < x){
        x--;  
    }
    else if(i > x){
        x += 2;
    }
    return 0;
}
```
Deriving $T(n)$

For loops, sum of the steps that get executed over all iterations

```cpp
#include <iostream>
using namespace std;

int main()
{
    for(int i=0; i < N; i++){
        x = 5;
        if(i < x){
            x--; 
        }
        else if(i > x){
            x += 2;
        }
    }
    return 0;
}
```
1. Setup the expression (or recurrence relationship) for the number of operations, $T(n)$

2. Solve to get a closed form for $T(n)$
   - Solve the recurrence relationship
   - Develop a series summation
   - Solve the series summation

3. Determine the asymptotic bound for $T(n)$
Derive an expression, \( T(n) \), in terms of the input size for the number of steps that are required.

```cpp
#include <iostream>

using namespace std;
const int n = 256;
unsigned char image[n][n]
int main()
{
    for(int i=0; i < n; i++){
        for(int j=0; j < n; j++){
            image[i][j] = 0;
        }
    }
    return 0;
}
```
Matrix Multiply

Derive an expression, $T(n)$, in terms of the input size for the number of steps that are required to solve a problem.

```
#include <iostream>
using namespace std;
const int n = 256;
int a[n][n], b[n][n], c[n][n];
int main()
{
  for(int i=0; i < n; i++){
    for(int j=0; j < n; j++){
      c[i][j] = 0;
      for(int k=0; k < n; k++){
        c[i][j] += a[i][k]*b[k][j];
      }
    }
  }
  return 0;
}
```
#include <iostream>
using namespace std;

const int n = /* large constant */;

unsigned char image[n][n]
int main()
{
    for(int i=0; i < n; i++){
        image[0][i] = 5;
    }

    for(int j=0; j < n; j++){
        image[1][j] = 5;
    }

    for(int k=0; k < n; k++){
        image[2][k] = 5;
    }

    return 0;
}
for(int i=0; i < n; i++){ 
    if (a[i][0] == 0){ 
        for (int j = 0; j < i; j++){ 
            a[i][j] = i * j; 
        } 
    } 
} 

Hint: Arithmetic series
for(int i=0; i < n; i++){
    if (i == 0){
        for (int j = 0; j < n; j++){
            a[i][j] = i*j;
        }
    }
}

for(int i=0; i < n; i++){
    if (i == 0){
        for (int j = 0; j < n; j++){
            a[i][j] = i*j;
        }
    }
}
for (int i = 0; i < n; i++)
{
    int m = sqrt(n);
    if( i % m == 0){
        for (int j=0; j < n; j++)
            cout << j << " ";
    }
    cout << endl;
}
int main()
{
    int data[4] = {1, 6, 7, 9};
    it_bsearch(3, data, 4);
}

int it_bsearch(int target, int data[], int len)
{
    int start = 0, end = len, mid;

    while (start < end) {
        mid = (start + end) / 2;
        if (data[mid] == target) {
            return mid;
        } else if (target < data[mid]) {
            end = mid - 1;
        } else {
            start = mid + 1;
        }
    }
    return -1;
}