SCC.121 ALGORITHMS AND COMPLEXITY
Operation Counting

Emma Wilson e.d.wilson1@lancaster.ac.uk

Today's Lecture

Aim: Introduce some important cases for time complexity, considering operation counts.

Learning objectives:

- To be able to use logarithmic functions for algorithms with logarithmic time complexity
- To be able to perform operation counting for a given algorithm
- To know what the key different types of time complexity based on operation counting are

Today's Lecture

- Examples of operation counting and T(N)
  - Case 1: Constant **T(N) = Constant**
  - Case 2: Linear **$T(N) = C_1 \times N + C_2$**
  - Case 3: Logarithmic **$C_1 \times \log_2 N + C_2$**
  - Case 4: Quadratic **$T(N) = C_1 \times N^2 + C_2 \times N + C_3$**

Today's Lecture

- Examples of operation counting and T(N)
  - **Case 1: Constant T(N) = Constant**
  - Case 2: Linear **$T(N) = C_1 \times N + C_2$**
  - Case 3: Logarithmic **$C_1 \times \log_2 N + C_2$**
  - Case 4: Quadratic **$T(N) = C_1 \times N^2 + C_2 \times N + C_3$**

Case 1: Averaging Function

```
double avg5(int theArray[])
{
      int total = 0;

      for (int i=0; i<5; i++)

            total += theArray[i];

      double avg = ((double) total) / 5.0;

      return avg;
}
```

How many operations does this piece of code execute (assume theArray has at least 5 values)?

Case 1: Operation Counting

```
double avg5(int theArray[])
{
      int total = 0; o1
              o2      o3     o4
      for (int i=0; i<5; i++)

            total += theArray[i]; o5

      double avg = ((double) total) / 5.0; o6

      return avg;  o7
}
```

Operations o3, o4, o5 executed every time around for loop

Case 1: Operation Counting

- How many times each operation is executed?

	o3	o5	o4
all cases	6	5	5

- Which are the best and worst cases?
  - **Best case:** 6+5+5
  - **Worst case:** 6+5+5
- Which will be the average case?
  - **Average Case:** 6+5+5

Case1: Working out T(N)

```
double avg5(int theArray[])
{
      int total = 0;o1 → 1
            o2 → 1  o3 → 6  o4 → 5
      for (int i=0; i<5; i++)

            total += theArray[i]; o5 → 5

      double avg = ((double) total) / 5.0; o6 → 1

      return avg; o7 → 1
}
```

What is the overall program time?

- **T(N) = 1+1+6+5+5+1+1 = 20**

Case1: T(N) is Constant

- What is the overall program time?
  - **Best Case:** T(N) = 20
    - **T(N) = Constant**
  - **Worst Case:** T(N) = 20
    - **T(N) = Constant**
- The time taken by this program is **independent** of the size of the input:
  - N is the size of array which can be more than 5, but since we only care about the first 5 values, increasing N does not change the time taken
- Case 1 is a bit contrived (and we assumed that ‘`theArray`’ has at least 5 values)!

Today's Lecture

- Examples of operation counting and T(N)
  - Case 1: Constant **T(N) = Constant**
  - **Case 2: Linear T(N) = C_1 x N + C_2**
  - Case 3: Logarithmic **C_1 x $\log_2$ N + C_2**
  - Case 4: Quadratic **T(N) = C_1 x N^2+ C_2 x N + C_3**

Case 2: Minimum Function

```
int findMin(int theArray[], int N)
{


        int smallest_i = 0; //Assume smallest value at index 0

        for (int i=1; i<N; i++)

                if (theArray[i] < theArray[smallest_i])

                        smallest_i = i;

        return smallest_i;
}
```

- How many operations does this piece of code execute?

Case 2: Operation Counting

```
int findMin(int theArray[], int N)
{


        int smallest_i = 0; //Assume smallest value at index 0  o1
                      o2     o3     o4
        for (int i=1; i<N; i++)

                if (theArray[i] < theArray[smallest_i]) o5

                        smallest_i = i; o6

        return smallest_i; o7
}
```

- Operations o3, o4, o5 executed every time around for loop
- Operation o6 executed if o5 is true

Case 2: Operation Counting

- How many times is each operation executed?

	o3	o4	o5	o6
Best case	N	N-1	N-1	0
Worst case	N	N-1	N-1	N-1

```
            o2    o3     o4
for (int i=1; i<N; i++)
        if (theArray[i] < theArray[smallest_i]) o5
                smallest_i = i; o6
```

Case 2: Working out T(N) Best Case

- What is the overall program time (Best case)?
  - **T(N) = 1+1+N+(N-1)+(N-1)+0+1=3N+1**

```
int findMin(int theArray[], int N)
{


        int smallest_i = 0; //Assume smallest value at index 0 o1 → 1
    o2 → 1  o3 → N  o4 → N-1
        for (int i=1; i<N; i++)

                if (theArray[i] < theArray[smallest_i]) o5 → N-1

                        smallest_i = i; o6 → 0

        return smallest_i; o7 → 1
}
```

Case 2: Working out T(N) Worst Case

- What is the overall program time (Worst case)?
  - **T(N) = 1+1+N+(N-1)+(N-1)+(N-1)+1=4N**

```
int findMin(int theArray[], int N)
{


      int smallest_i = 0; //Assume smallest value at index 0 o1 → 1
     o2 → 1 o3 → N  o4 → N-1
      for (int i=1; i<N; i++)

            if (theArray[i] < theArray[smallest_i]) o5 → N-1

                  smallest_i = i; o6 → N-1

      return smallest_i; o7 → 1
}
```

Case 2: T(N) is Linear

- What is the overall program time?
  - **Best Case:** T(N) = **3N+1**
    - **$T(N) = C_1 \times N + C_2$**
    - **$C_1 = 3$ and $C_2 = 1$ are constant**
  - **Worst Case:** T(N) = **4N**
    - **$T(N) = C_1 \times N + C_2$**
    - **$C_1 = 4$ and $C_2 = 0$ are constant**
- If we plot this, we get a line with slope C_1
- The time taken by this program is directly proportional to the size of the input 'N'
  - doubling N (approximately) doubles the time taken
  - tripling N (approximately) triples the time taken

Today's Lecture

- Examples of operation counting and T(N)
  - Case 1: Constant T(N) = Constant
  - Case 2: Linear **$T(N) = C_1 \times N + C_2$**
  - **Case 3: Logarithmic** **$C_1 \times log_2 N + C_2$**
  - Case 4: Quadratic **$T(N) = C_1 \times N^2 + C_2 \times N + C_3$**

Logarithmic Functions

- Logarithmic Functions are the inverse of Exponential Functions
- We know $\mathbf{2^3 = 2 \times 2 \times 2 = 8}$
- What power do we have to raise 2 by to get 8? $\boldsymbol{2^x = 8}$
- Can express by using logarithms! $\boldsymbol{x = \log_2(8) = 3}$
- $2^3 = 8$ **is equivalent to** $\log_2(8) = 3$

Exponential Form		Logarithmic Form
$2^4 = 16$	$\Longleftrightarrow$	$\log_2(16) = 4$
$10^2 = 100$	$\Longleftrightarrow$	$\log_{10}(100) = 2$
$4^3 = 64$	$\Longleftrightarrow$	$\log_4(64) = 3$

Logarithmic Functions

In general

- $\boldsymbol{b^c = a}$ is equivalent to $\mathbf{log}_{\boldsymbol{b}}(\boldsymbol{a}) = \boldsymbol{c}$
  for $a > 0$ and $b > 0$ and $b \neq 1$

- Both equations describe the same relationship between $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$
  - $\boldsymbol{b}$ is the base
  - $\boldsymbol{c}$ is the exponent
  - $\boldsymbol{a}$ is the argument

Case 3: Function

```
int logBaseTwoN(int N)
{
        int count = 0;

        while (N > 1) {

                count++;

                N = N/2;
        }
        return count;
}
```

- How many operations does this piece of code execute?

Case 3: Operation Counting

```
int logBaseTwoN(int N)
{
      int count = 0; o1

      while (N > 1) { o2

            count++;  o3

            N = N/2;  o4
      }
      return count; o5
}
```

- Operations o2, o3, o4 executed every time around *while* loop

Case 3: Operation Counting

- How many times each operation is executed?
  - Try different N
    - N=$2^1 = 2$
    - N=$2^2 = 4$
    - N=$2^3 = 8$
    - N=$2^4 = 16$
    - N=$2^5 = 32$
    - …

```
int logBaseTwoN(int N)
{
        int count = 0;   o1

        while (N > 1) {  o2

                count++;  o3

                N = N/2;  o4
        }
        return count;    o5
}
```

Case 3: Operation Counting

- How many times each operation is executed?
  - **Try N=$2^1 = 2$**

```
int logBaseTwoN(int N)
{
        int count = 0;
        while (N > 1) {o2
                count++;  o3
                N = N/2;  o4
        }
        return count;
}
```

o2 → 2

o3 → 1

o4 → 1

Case 3: Operation Counting

- How many times each operation is executed?
  - **Try N=$2^2 = 4$**

```
int logBaseTwoN(int N)
{
        int count = 0;

        while (N > 1) {o2

                count++; o3

                N = N/2; o4

        }
        return count;
}
```

o2 → 3

o3 → 2

o4 → 2

Case 3: Operation Counting

- How many times each operation is executed?
  - **Try N=**$2^3 = 8$

```
int logBaseTwoN(int N)
{
        int count = 0;

        while (N > 1) {  o2

                count++;  o3

                N = N/2;  o4

        }
        return count;
}
```

o2 → 4

o3 → 3

o4 → 3

Case 3: Operation Counting

- How many times each operation is executed?
  - **Try N=$2^4 = 16$**

```
int logBaseTwoN(int N)
{
        int count = 0;

        while (N > 1) {  o2

                count++;  o3

                N = N/2;  o4

        }
        return count;
}
```

o2 → 5

o3 → 4

o4 → 4

Case 3: Operation Counting

- How many times each operation is executed?

b^c = a is equivalent to **log$_b$ a = c** for a > 0 and b > 0 and b != 1

	Instruction	o2	o3	o4
$x=1$	N=2^1 $= 2$	2	1	1
$x=2$	N=2^2 $= 4$	3	2	2
$x=3$	N=2^3 $= 8$	4	3	3
$x=4$	N=2^4 $= 16$	5	4	4
$x=5$	N=2^5 $= 32$	6	5	5
$x=6$	N=2^6 $= 64$	7	6	6
	N=2^x	$x+1$	x	x

$\log_2(N) = x$

Instruction	o2	o3	o4
N=2^1 $= 2$	$\log_2 2^1 + 1$	$\log_2 2^1$	$\log_2 2^1$
N=2^2 $= 4$	$\log_2 2^2 + 1$	$\log_2 2^2$	$\log_2 2^2$
N=2^3 $= 8$	$\log_2 2^3 + 1$	$\log_2 2^3$	$\log_2 2^3$
N=2^4 $= 16$	$\log_2 2^4 + 1$	$\log_2 2^4$	$\log_2 2^4$
N=2^5 $= 32$	$\log_2 2^5 + 1$	$\log_2 2^5$	$\log_2 2^5$
N=2^6 $= 64$	$\log_2 2^6 + 1$	$\log_2 2^6$	$\log_2 2^6$
	$\log_2 \mathrm{N} + 1$	$\log_2 \mathrm{N}$	$\log_2 \mathrm{N}$

Case 3: Operation Counting

- How many times each operation is executed?

	o2	o3	o4
all cases	$\log_2 N + 1$	$\log_2 N$	$\log_2 N$

- Which are the best and worst cases (for o2,o3, o4)?
  - **Best case:** $(\log_2 N+1)+\log_2 N+\log_2 N = 3\log_2 N + 1$
  - **Worst case:** $3\log_2 N + 1$
  - **Average case:** $3\log_2 N + 1$

Case 3: Working out T(N)

- What is the overall program time?
  - **T(N) =** $1 + (3\log_2 \mathrm{N} + 1) + 1 = 3\log_2 \mathrm{N} + 3$

```
int logBaseTwoN(int N)
{
        int count = 0;     o1 → 1

        while (N > 1) {    o2 → log₂ N + 1

                count++;   o3 → log₂ N

                N = N/2;   o4 → log₂ N

        }
        return count;      o5 → 1
}
```

Case 3: T(N) is Logarithmic

- What is the overall program time?
  - **Best Case:** T(N) = $3\log_2 N + 3$
    - **$T(N) = C_1 \times \log_2 N + C_2$**
    - **$C_1 = 3$ and $C_2 = 3$ are constant**
  - **Worst Case:** T(N) = $3\log_2 N + 3$
    - **$T(N) = C_1 \times \log_2 N + C_2$**
    - **$C_1 = 3$ and $C_2 = 3$ are constant**
- The time taken by this program is logarithmically proportional to the size of the input
  - Our code/function '`logBaseTwoN`()' is accurate for N that is 2^x such that x is a natural number. It is for demonstration purposes only.

Today's Lecture

- Examples of operation counting and T(N)
  - Case 1: Constant **T(N) = Constant**
  - Case 2: Linear $T(N) = C_1 \times N + C_2$
  - Case 3: Logarithmic $\mathbf{C_1 \times \log_2 N + C_2}$
  - **Case 4: Quadratic** $\mathbf{T(N) = C_1 \times N^2 + C_2 \times N + C_3}$

Case 4: Function

```
void quadratic(int N)
{
     int count = 0;

      for (int i=0; i<N; i++)

           for (int j=0; j<N; j++)

                count++;
}
```

- How many operations does this piece of code execute?

Case 4: Operation Counting

```
void quadratic(int N)
{
      int count = 0; o1
                o2       o3    o4
       for (int i=0; i<N; i++)
                     o5      o6    o7
           for (int j=0; j<N; j++)

                count++;  o8
}
```

- Let’s start with the **inner** *for* loop

Case 4: Operation Counting

```
void quadratic(int N)
{
      int count = 0; o1
                  o2      o3   o4
       for (int i=0; i<N; i++)
                     o5      o6   o7
           for (int j=0; j<N; j++)

               count++; o8
}
```

- Operations o6, o7, o8 executed every time around the **innermost** *for* loop

Case 4: Operation Counting the inner *for* loop

- How many times each operation is executed?

	o5	o6	o8	o7
All cases	1	N+1	N	N

- Which are the best and worst cases?
  - **Best case:** 3N+2
  - **Worst case:** 3N+2

Case 4: Operation Counting

```
void quadratic(int N)
{
      int count = 0; o1
                  o2       o3    o4
       for (int i=0; i<N; i++)
      ┌─────────────────────────────┐
      │      o’ which has           │
      │   3N+2 instructions         │
      └─────────────────────────────┘
}
```

- Let’s continue with the **outer** *for* loop

Case 4: Operation Counting

```
void quadratic(int N)
{
      int count = 0;

       for (int i=0; i<N; i++)
```

o’ which has
3N+2 instructions

```
}
```

- Operations o3, o4, o’ executed every time around the **outer** *for* loop

Case 4: Operation Counting the outer *for* loop

- How many times each operation is executed?

	o3	o'	o4
all cases	N+1	N	N

- Which are the best and worst cases?
  - **Best case:** $(N+1)+\mathbf{N\times(3N+2)}+N=3N^2+4N+1$
  - **Worst case:** $3N^2+4N+1$

Case 4: Working out T(N)

- What is the overall program time?
  - **T(N) =** 1+1+(N+1)+N+N(3N+2) = …
  - **T(N) = 3N^2+4N+3**

o3 N+1 | o' N | o4 N

```
void quadratic(int N)
{
      int count = 0;
       for (int i=0; i<N; i++)
}
```

o1 → 1

o2 → 1 o3 → N+1 o4 → N

o' which has
3N+2 instructions
→ N

Case 4: T(N) is Quadratic

- What is the overall program time?
  - **Best Case:** $T(N) = 3N^2 + 4N + 3$
    - **$T(N) = C_1 \times N^2 + C_2 \times N + C_3$**
    - **$C_1 = 3$ and $C_2 = 4$ and $C_3 = 3$ are constant**
  - **Worst Case:** $T(N) = 3N^2 + 4N + 3$
    - **$T(N) = C_1 \times N^2 + C_2 \times N + C_3$**
    - **$C_1 = 3$ and $C_2 = 4$ and $C_3 = 3$ are constant**
- The time taken by this program is quadratic with respect to input size N
  - because the highest order term is N^2

Question

How many operations does this piece of code execute?

```
void cubic(int N)
{
    int count = 0;
    for (int i=0; i<N; i++)
            for (int j=0; j<N; j++)
                    for (int k=0; k<N; k++)
                            count++;
}
```

slido

Please download and install the Slido app on all computers you use

How many operations does this piece of code execute?

ⓘ Start presenting to display the poll results on this slide.

Solution

How many operations does this piece of code execute?

```
void cubic(int N)
{
    int count = 0;  o1
         o2   o3   o4
    for (int i=0; i<N; i++)
                         o5     o6     o7
          for (int j=0; j<N; j++)
                             o8     o9     o10
               for (int k=0; k<N; k++)
               count++;          o11
}
```

Start with the inner loop

```
 o8       o9       o10
for (int k=0; k<N; k++)
count++;  o11
```

o8=1, o9=N+1, o10=N, o11=N

o'= 3N + 2

Solution

How many operations does this piece of code execute?

```
void cubic(int N)
{
    int count = 0; o1
           o2      o3   o4
    for (int i=0; i<N; i++)
                  o5       o6    o7
        for (int j=0; j<N; j++)

                 o'=3N+2

}
```

Now consider the middle loop

```
      o5     o6      o7
for (int j=0; j<N; j++)
        o';o12
```

- o5=1, o6=N+1, o7=N
- o' has 3N+2 operations and is executed N times
- o12=N(2+3N)
- o''= 1+N+1+N+2N + $3N^2$
- o''=$3N^2$ + 4N + 2

Solution

How many operations does this piece of code execute?

```
void cubic(int N)
{
    int count = 0;  o1
          o2      o3      o4
    for (int i=0; i<N; i++)

                o''=3N²+4N+2

}
```

Finally consider the outer loop

```
     o2     o3       o4
for (int i=0; i<N; i++)
        o'';o13
```

- o2=1, o3=N+1, o4=N
- o'' has $3N^2 + 4N + 2$ operations and is executed N times
- o13=N($3N^2 + 4N + 2$)
- o'''=$1+N+1+N+3N^3+4N^2+2N$
- o'''= $3N^3+4N^2+4N+2$
- Overall: add o1 to o''' giving

$3N^3+4N^2+4N+3$

Please download and install the Slido app on all computers you use

Audience Q&A

ⓘ Start presenting to display the audience questions on this slide.

Summary

Today's lecture: Introduce some important cases for time complexity, considering operation counts.

- **Next Lecture:** Linear search: Best, worst and **average case**