SCC.121: ALGORITHMS AND COMPLEXITY
Linear Search: Time Complexity

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

Today’s Lecture

Aim: To analyse the time complexity of a linear search algorithm in terms of the best case, worst case and average case

Learning objectives:

- To be able to use the sigma notation for series summation and where needed apply to operation counting problems
- To know how to calculate the best, worst and average case time complexity of a linear search algorithm

Today’s Lecture

- Maths: Sigma summation – notation and evaluation
- Linear search example
    - Best case
    - Worst case
    - Average case

Today’s Lecture

- **Maths: Sigma summation – notation and evaluation**
- Linear search example
  - Best case
  - Worst case
  - Average case

Maths and Sigma Notation

- Sigma notation is a mathematical shorthand for expressing sums where each term is of the same form
- For example, if we want to write out the sum of all the integers from 1 to 10, we could write 1+2+3+4+5+6+7+8+9+10
- Equivalently, we could write 1+2+3+……+10
- Or, we can use the **Sigma** notation and write

$$\sum_{i=1}^{10} i$$

Maths and Sigma Notation

$$\sum_{i=1}^{10} i$$

which we may read as the sum of all i, for i taking on every integer value starting with 1 and going up to 10

$$\sum_{i=1}^{10} i = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$$

Maths and Sigma Notation

- Similarly, if we wanted to sum the squares of all integers from 1 to 10, we could write

$$1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 + 7^2 + 8^2 + 9^2 + 10^2$$

- Or,

$$1^2 + 2^2 + 3^2 + \cdots + 10^2$$

- Or

$$\sum_{i=1}^{10} i^2$$

Maths and Sigma Notation

- Note – the index does not have to be i and does not have to start from 1.

For example

$$\sum_{j=5}^{9} j = 5 + 6 + 7 + 8 + 9$$

- Rather than a given integer, we can also sum up to N terms

For example

$$\sum_{k=1}^{N} k = 1 + 2 + 3 + \cdots + N$$

Maths and Sigma Notation

- We can also have double (or more) summations,

$$\sum_{i=1}^{3}\sum_{j=2}^{4}(i+j)$$

- To evaluate – we first evaluate the inner (right sum)

$$\sum_{i=1}^{3}((i+2)+(i+3)+(i+4)) = \sum_{i=1}^{3}(3i+9)$$

- Then plug into the outer (left sum)

$$\sum_{i=1}^{3}\big((3(1)+9)+(3(2)+9)+(3(3)+9)\big) = 45$$

Summation Formula

$$\sum_{i=1}^{N} i = 1 + 2 + 3 + \cdots + N = \frac{N(N+1)}{2}$$

$$\sum_{i=1}^{N} 1 = 1 + 1 + 1 + \cdots + 1 = N$$

$$\sum_{i=a}^{b} 1 = b - a + 1$$

Summation Formula

$$\sum_{i=a}^{b} M = M \sum_{i=a}^{b} 1$$

$$\sum_{i=a}^{b} (A \times i) = A \times \sum_{i=a}^{b} i$$

$$\sum_{i=a}^{b} (A \times i + B) = \sum_{i=a}^{b} (A \times i) + \sum_{i=a}^{b} B$$

Question

- The Nth partial sum (the sum of the first N terms) is given by a simple formula

$$\sum_{i=1}^{N} i = 1 + 2 + 3 + \cdots + N = \frac{N(N+1)}{2}$$

- Question: What is the sum of the first N-1 terms?
  - $1 + 2 + 3 + 4 + \cdots + (N-1) = ?$

  - $\frac{N-1(N-1+1)}{2} = \frac{(N-1)N}{2}$

For Loop: General Rule

- The number of times the instruction within a for loop is executed is given using a general rule

General Rule:

```
for (i = a; i <= b; i++) {
        // instructions
}
```

$$\sum_{i=a}^{b} 1 = b - a + 1$$

Where a and b are positive integer numbers and $a < b$

Care: to use this general rule the for loop must be in the above format

For Loop: General rule – example question

General Rule:

```
for (i = a; i <= b; i++) {
        // instructions
}
```

$$\sum_{i=a}^{b} 1 = b - a + 1$$

How many times is the instruction in the following for loop executed?

```
for (j = N; j > 1; j--){
// instruction
}
```

First transform to correct format to use general rule

```
for (j = 2; j <= N; j++){
// instruction
}
```

For Loop: General rule – example question

General Rule:

```
for (i = a; i <= b; i++) {
        // instructions
}
```

$$\sum_{i=a}^{b} 1 = b - a + 1$$

```
for (j = 2; j <= N; j++){
// instruction
}
```

$$a = 2, b = N$$

$$\sum_{i=2}^{N} N - 2 + 1 = N - 1$$

slido

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.

Today’s Lecture

- Maths: Sigma summation – notation and evaluation
- **Linear search example**
    - Best case
    - Worst case
    - Average case

Worst, Best and Average case

Worst Case Analysis (Mostly used)

- In the worst-case analysis, we calculate the upper bound on the running time of an algorithm. We must know the case that causes a maximum number of operations to be executed.

Best Case Analysis (Rarely used)

- In the best-case analysis, we calculate the lower bound on the running time of an algorithm. We must know the case that causes a minimum number of operations to be executed.

Average Case Analysis (Rarely used)

- In average case analysis, we take all possible inputs and calculate the computing time for all of the inputs. Sum all the calculated values and divide the sum by the total number of inputs.

Linear Search

- Example of array of integers, and one integer to search for in array
- Exercise
  - Write a function '*isInArray*' which takes the three arguments: An array of integers (theArray), the length of the array (N) and an integer to search the array for (iSearch). The function '*isInArray*' returns true if the number is in the array, false otherwise

```
boolean isInArray(int theArray[], int N, int iSearch)
{
//Write your code here
}
```

Linear Search

```
int isInArray(int theArray[], int N, int iSearch)
{

        for (int i = 0; i < N; i++)
                if (theArray[i] == iSearch)
                        return 1;

        return 0;
}
```

- How many operations does this piece of code execute?

Linear Search

```
int isInArray(int theArray[], int N, int iSearch)
{
                 o1     o2     o3
        for (int i = 0; i < N; i++)
                if (theArray[i] == iSearch) o4
                        return 1; o5

        return 0; o6
}
```

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

Operation Counting

How many times each operation is executed?

	o2	o4	o3
‘iSearch’ is the first element	1	1	0
‘iSearch’ is the last element	N	N	N-1
‘iSearch’ is not in ‘theArray’	N+1	N	N

```
                o1      o2    o3
    for (int i = 0; i < N; i++)

        if (theArray[i] == iSearch) o4
```

Operation Counting

How many times each operation is executed?

	o2	o4	o3	
‘iSearch’ is the first element	1	1	0	Best case
‘iSearch’ is the last element	N	N	N-1	
‘iSearch’ is not in ‘theArray’	N+1	N	N	Worst case

- What are the best and worst cases?
  - **Best case:** ‘iSearch’ is the first element in the array;
  - **Worst case:** ‘iSearch’ is not in the array
  - **Average Case:** Between best and worst case

Working out T(N) in Best Case

- What is the overall program time (Best case)?
  - **T(N) = 1+1+0+1+1+0 = 4**

```
int isInArray(int theArray[], int N, int iSearch)
{
                  o1 → 1   o2 → 1  o3 → 0
          for (int i = 0; i < N; i++)

                    if (theArray[i] == iSearch) o4 → 1

                              return 1; o5 → 1

          return 0; o6 → 0
}
```

Working out T(N) in Worst Case

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

```
int isInArray(int theArray[], int N, int iSearch)
{

                o2 → 1  o3 → N+1 o4 → N
      for (int i = 0; i < N; i++)

            if (theArray[i] == iSearch) o5 → N

                  return 1; o6 → 0

      return 0;  o7 → 1
}
```

Working out T(N)

- What is the overall program time?
  - **Best Case:** T(N) = 4
    - **T(N) = Constant**
  - **Worst Case:** T(N) = 3N+3
    - **$T(N) = C_1 \times N + C_2$**
    - **C_1 and C_2 are constant**
- In the worst case, If we plot this, we get a line with slope C_1
- In the worst case, 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

Working out T(N) Average Case

- What is the overall program time (Average case)?
  - Suppose '`iSearch`' is in the array

```
int isInArray(int theArray[], int N, int iSearch)
{
                   o1→ 1    o2 →?   o3→ ?
        for (int i = 0; i < N; i++)

                if (theArray[i] == iSearch) o4 → ?

                        return 1; o6 → 1

        return 0; o7 → 0
}
```

Working out T(N) Average Case

- What is the overall program time (Average case)?
  - Suppose ‘`iSearch`’ is in the array
  - In average, how many times o4 is executed?

Position of `iSearch`	1st	2nd	3rd	4th	5th	6th	7th	…	Nth
Number of executions (o4)	1	2	3	4	5	6	7	…	N

Average number of operations (o4): $\frac{1}{N} \times (1 + 2 + 3 + 4 + 5 + 6 + \cdots + N) = \cdots$

Working out T(N) Average Case

- What is the overall program time (Average case)?
  - Suppose ‘`iSearch`’ is in the array
  - In average, how many times o4 is executed?

$$\frac{1}{N} \times (1+2+3+4+5+6+\cdots+N) = \frac{1}{N}\sum_{i=1}^{N} i$$

$$\boxed{\sum_{i=1}^{N} i = \frac{N(N+1)}{2}}$$

$$\frac{1}{N} \times \left(\frac{N(N+1)}{2}\right) = \cdots$$

$$= \frac{(N+1)}{2}$$

Working out T(N) Average Case

- What is the overall program time (Average case)?
  - Suppose ‘`iSearch`’ is in the array
  - What about o2 and o3?

```
int isInArray(int theArray[], int N, int iSearch)
{
                            o1→ 1     o2 →?    o3→ ?
        for (int i = 0; i < N; i++)
            if (theArray[i] == iSearch) o4 → (N+1)/2
                return 1;     o5 → 1
        return 0;     o6 → 0
}
```

o4 → $\frac{(N+1)}{2}$

Working out T(N) Average Case

- What is the overall program time (Average case)?
  - Suppose ‘iSearch’ is in the array
  - In average, how many times o3 is executed?

Position of iSearch	1st	2nd	3rd	4th	5th	6th	7th	…	Nth
Number of executions (o3)	0	1	2	3	4	5	6	…	N-1

Average number of operations (o3): $\frac{1}{N} \times (0 + 1 + 2 + 3 + 4 + 5 + 6 + \cdots + (N-1)) = \cdots$

$$\frac{1}{N} \times \left(\frac{N(N-1)}{2}\right) = \frac{N-1}{2}$$

$$\boxed{\sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}}$$

Working out T(N) Average Case

- What is the overall program time (Average case)?
  - Suppose ‘`iSearch`’ is in the array
  - In average, how many times o2 is executed?

Position of `iSearch`	1st	2nd	3rd	4th	5th	6th	7th	…	Nth
Number of executions (o2)	1	2	3	4	5	6	7	…	N

Average number of operations (o2): $\frac{1}{N} \times \left(\frac{N(N+1)}{2}\right) = \frac{N+1}{2}$

Working out T(N) Average Case

- What is the overall program time (Average case)?
  - Suppose '`iSearch`' is in the array
  - **T(N)** = $1+\frac{N+1}{2}+\frac{N-1}{2}+\frac{N+1}{2}+1+0=\frac{3}{2}N+\frac{5}{2}$

```
int isInArray(int theArray[], int N, int iSearch)
{
```

o1 $\rightarrow$ 1 o2 $\rightarrow \frac{(N+1)}{2}$ o3 $\rightarrow \frac{(N-1)}{2}$

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

                if (theArray[i] == iSearch)
```

o4 $\rightarrow \frac{(N+1)}{2}$

```
                        return 1;
```

o5 $\rightarrow$ 1

```
        return 0;
```

o6 $\rightarrow$ 0

```
}
```

Working out T(N)
Average Case

- What is the overall program time (Average case)?
  - if ‘`iSearch`’ is in the array **T(N)** = $\frac{3}{2}N + \frac{5}{2}$
  - if ‘`iSearch`’ is not in the array **T(N)** = $3N + 3$

```
int isInArray(int theArray[], int N,  int iSearch)
{
                o1→ 1   o2 →N + 1  o3→ N
        for (int i = 0; i < N; i++)

                if (theArray[i] == iSearch) o4 → N

                        return 1; o5 → 0

        return 0; o6 → 1
}
```

Working out T(N) Average Case

- What is the overall program time (Average case)?
  - if '`iSearch`' is in the array **T(N) =** $\mathbf{\frac{3}{2}N + \frac{5}{2}}$
  - if '`iSearch`' is **NOT** in the array **T(N) =** $\mathbf{3N + 3}$
- If the probability that '`iSearch`' is in the array is P $(0 \leq P \leq 1)$
  - Probability that '`iSearch`' is **NOT** in the array is $1 - P$
- So, the overall program time in the Average case is:
  - **T(N) =** $\mathrm{P} \times \left(\frac{3}{2}N + \frac{5}{2}\right) + (1 - P) \times (3N + 3)$
  - **T(N) =** $\left(\frac{3}{2}P + 3 - 3\mathrm{P}\right)\mathrm{N} + \left(\frac{5}{2}P + 3 - 3P\right)$
  - **T(N) = C_1 x N + C_2 Where C_1 and C_2 are constant**

slido

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

- Introduced the sigma notation and shown how it can be useful for calculating the time complexity
- We have analysed algorithms with different time complexity T(N)
- Time complexity of Linear Search algorithm in best case, worst case and average case (generally not easy to evaluate)

Next Lecture: Other searching algorithms that reduce time complexity from linear search