Analysis of Algorithms

Chapter 3

CPTR 318

1

Algorithm

- An algorithm is a clearly specified set of instructions a computer follows to solve a problem
  - The number of instructions is finite
  - Each instruction must be executable in a finite amount of time
  - Each instruction must be unambigous

2

Algorithm Analysis: Technique #1

- Performance could be analyzed by using:
  - Actual Space requirements
    - Instruction and Data space

3

Algorithm Analysis: Technique #2

- Actual Time requirements
  - The above method depends on the particular compiler as well as the specific computer on which the program is run

4

Algorithm Analysis: Technique #3

- One way to analyze algorithms is to count all the instructions or steps in the algorithm
- Generally we discuss the algorithm's efficiency as a function of the number of elements to be processed. The general format that we will use is

$f(n)$ = efficiency

5

Counting Steps

- If the algorithm does not have loops that depend on the number of elements to be processed then the number of steps is a constant.
  - $f(n) = c$
  - c is a constant

6

Counting Steps

- A primitive execution consists of assignment, arithmetic, comparison, array access, function call, function return, etc.
- One C++ statement may contain several primitive executable steps
  - `x = a + 3;      // Two steps`
  - `x = a[i] + 3;   // Three steps`
  - `return x > 3;   // Two steps`
- `++`, `--`, `+=`, etc. count as two (arithmetic and assignment)
- We will not count non-executable statements, such as declarations

Counting Steps: Example #1

```
int f(int x) {
    int c, result;
    c = x + 5;          2
    if (c > 10)         1
        result = c;     0 or 1
    else
        result = x;     1 or 0
    return result;      1
}
```

$f(n) = 5$

Counting Steps

- If the algorithm has only sequential instructions and simple counting loops and at least one loop depends on the number of elements to be processed then
  - $f(n) = an + b$, where a and b are constants
  - Example: Sequential search

Counting Steps: Example #2

```
int f(int n) {
    int i = 1,              1
        s = 0;              1
    while (i <= n) {        n + 1
        s += i;             2n
        i++;                2n
    }
    return s;               1
}
```

$f(n) = 5n + 4$

Counting Steps

- If the algorithm contains in addition to the previous slide a nested counting loop where both loops depend on the number of elements to be processed then
  - $f(n) = an^2 + bn + c$
  - Example: Selection Sort
- In general a polynomial efficiency depends on the number of nested loops present:
  - $f(n) = a_m n^m + a_{m-1} n^{m-1} + a_{m-2} n^{m-2} + \dots a_1 n + a_0$

Counting Steps

- Logarithmic loops.
  - These are algorithms whose efficiency contain the log function
  - Example: Binary Search

```
while ( n > 0 ) {
    Application code …
    n = n / 2
}
```

  - $f(n) = a \log_2 n + c$

Best, Worst, Average

- When counting the steps for the efficiency function we have sometimes to consider the best, worst and average cases
  - Example: Sequential search

```
//  Returns the index of seek within vec
//  Returns -1 if seek is not an element of vec
int find(const vector<int>& vec, int seek) {
    int n = vec.size();
    for (int i = 0; i < n; i++)
        if (vec[i] == seek)
            return i;
    return -1;  //   Not found
}
```

13

Algorithm Analysis: Technique #4

- Big-O notation gives a general order of magnitude to compare algorithms.
  - It capture the most dominant term in a function
- It gives us an upper limit to compare the algorithms
- Classify algorithms as belonging to a family of algorithms

14

Growth Rates

n	$f(n) = n^2$	$f(n) = n^2 + 4n + 20$
10	100	160
100	10,000	10,420
10,000	100,000,000	100,040,020

15

Big-O Definition

$f(n) = O(g(n))$ iff positive constants c and n_0 exist such that:

$$f(n) \leq cg(n) \text{ for all } n \geq n_0$$

16

Big-O Definition

$f(n) = O(g(n))$ iff positive constants c and n_0 exist such that:

$$f(n) \leq cg(n) \text{ for all } n \geq n_0$$

f grows at about the rate as g

17

Big-O Definition

$f(n) = O(g(n))$ iff positive constants c and n_0 exist such that:

$$f(n) \leq cg(n) \text{ for all } n \geq n_0$$

f grows at about the rate as g

O bounds from above

18

Examples

Consider $f(n) = 3n + 2$.

Examples

Consider $f(n) = 3n + 2$.

$f(n)$

Examples

Consider $f(n) = 3n + 2$.

$f(n) = 3n + 2$

Examples

Consider $f(n) = 3n + 2$.

$f(n) = 3n + 2 \leq 3n + 2n$

Examples

Consider $f(n) = 3n + 2$.

$f(n) = 3n + 2 \leq 3n + 2n = 5n$, for all $n \geq 1$.

Examples

Consider $f(n) = 3n + 2$.

$f(n) = 3n + 2 \leq 3n + 2n = 5n$, for all $n \geq 1$.

Therefore $f(n) = O(n)$

Examples

Consider $f(n) = 3n + 2$.

$c = 5$

$f(n) = 3n + 2 \leq 3n + 2n = 5n$, for all $n \geq 1$.

$n_0 = 1$

Therefore $f(n) = O(n)$

25

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

27

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

28

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2$

29

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5$

30

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5$

31

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5 = 5(1)$

32

Inductive Proof

Show $\boxed{3n + 2} \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$\boxed{3(1) + 2} = 5 \leq 5 = 5(1)$

33

Inductive Proof

Show $\boxed{3n + 2} \leq \boxed{5n}$, for all $n \geq 1$

Basis: $n = 1$

$\boxed{3(1) + 2} = 5 \leq 5 = \boxed{5(1)}$

34

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$

35

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$

$3(n + 1) + 2$

36

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$

$3(n + 1) + 2 = (3n + 3) + 2$ (distributive property)

37

Inductive Proof

Show $3n + 2 \le 5n$, for all $n \ge 1$
Basis: $n = 1$
$3(1) + 2 = 5 \le 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$
$3(n + 1) + 2 = (3n + 3) + 2$

38

Inductive Proof

Show $3n + 2 \le 5n$, for all $n \ge 1$
Basis: $n = 1$
$3(1) + 2 = 5 \le 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$
$3(n + 1) + 2 = (3n + 3) + 2$
$= 3n + 2 + 3$ (commutative property)

39

Inductive Proof

Show $3n + 2 \le 5n$, for all $n \ge 1$
Basis: $n = 1$
$3(1) + 2 = 5 \le 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$
$3(n + 1) + 2 = (3n + 3) + 2$
$= 3n + 2 + 3$

40

Inductive Proof

Show $\boxed{3n + 2} \le 5n$, for all $n \ge 1$
Basis: $n = 1$
$3(1) + 2 = 5 \le 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$
$3(n + 1) + 2 = (3n + 3) + 2$
$= \boxed{3n + 2} + 3$

41

Inductive Proof

Show $\boxed{3n + 2 \le 5n}$, for all $n \ge 1$
Basis: $n = 1$
$3(1) + 2 = 5 \le 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$
$3(n + 1) + 2 = (3n + 3) + 2$
$= \boxed{3n + 2} + 3$
$\le \boxed{5n} + 3$ (inductive hypothesis)

42

Inductive Proof

Show $3n + 2 \le 5n$, for all $n \ge 1$
Basis: $n = 1$
$3(1) + 2 = 5 \le 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$
$3(n + 1) + 2 = (3n + 3) + 2$
$= 3n + 2 + 3$
$\le 5n + 3$

43

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$

$$\begin{aligned} 3(n + 1) + 2 &= (3n + 3) + 2 \\ &= 3n + 2 + 3 \\ &\leq 5n + 3 \\ &\leq 5n + 5 \qquad (3 \leq 5) \end{aligned}$$

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$

$$\begin{aligned} 3(n + 1) + 2 &= (3n + 3) + 2 \\ &= 3n + 2 + 3 \\ &\leq 5n + 3 \\ &\leq 5n + 5 \end{aligned}$$

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$

$$\begin{aligned} 3(n + 1) + 2 &= (3n + 3) + 2 \\ &= 3n + 2 + 3 \\ &\leq 5n + 3 \\ &\leq 5n + 5 \\ &= 5(n + 1) \qquad \text{(distributive property)} \end{aligned}$$

Inductive Proof

Show $3n + 2 \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$

$$\begin{aligned} 3(n + 1) + 2 &= (3n + 3) + 2 \\ &= 3n + 2 + 3 \\ &\leq 5n + 3 \\ &\leq 5n + 5 \\ &= 5(n + 1) \end{aligned}$$

Inductive Proof

Show $\boxed{3n + 2} \leq 5n$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$

$$\begin{aligned} \boxed{3(n + 1) + 2} &= (3n + 3) + 2 \\ &= 3n + 2 + 3 \\ &\leq 5n + 3 \\ &\leq 5n + 5 \\ &= 5(n + 1) \end{aligned}$$

Inductive Proof

Show $\boxed{3n + 2} \leq \boxed{5n}$, for all $n \geq 1$

Basis: $n = 1$

$3(1) + 2 = 5 \leq 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$

$$\begin{aligned} \boxed{3(n + 1) + 2} &= (3n + 3) + 2 \\ &= 3n + 2 + 3 \\ &\leq 5n + 3 \\ &\leq 5n + 5 \\ &= \boxed{5(n + 1)} \end{aligned}$$

Inductive Proof

Show $3n + 2 \le 5n$, for all $n \ge 1$
Basis: $n = 1$
$3(1) + 2 = 5 \le 5 = 5(1)$

Induction: Show $P(k) \rightarrow P(k + 1)$
$3(n + 1) + 2 = (3n + 3) + 2$
$= 3n + 2 + 3$
$\le 5n + 3$
$\le 5n + 5$
$= 5(n + 1)$

50

Examples

- Example 2: Is $2^{n+2} = O(2^n)$?
- Example 3: Is $3n + 2 = O(n^2)$?

51

Examples

Prove that $10n^2 + 4n + 2 \ne O(n)$.

52

Examples

Prove that $10n^2 + 4n + 2 \ne O(n)$.

Suppose $10n^2 + 4n + 2 = O(n)$ then there exists a positive c and a n_0 such that $10n^2 + 4n + 2 \le cn$, for all $n \ge n_0$.

53

Examples

Prove that $10n^2 + 4n + 2 \ne O(n)$.

Suppose $10n^2 + 4n + 2 = O(n)$ then there exists a positive c and a n_0 such that $10n^2 + 4n + 2 \le cn$, for all $n \ge n_0$. Dividing both sides by n we get $10n + 4 + 2/n \le c$ for all $n \ge n_0$ This is a false statement because as $n \rightarrow \infty$, $10n + 4 + 2/n \rightarrow \infty$ which cannot be less than c. Therefore $10n^2 + 4n + 2 \ne O(n)$.

54

Helpful Theorems

Theorem1: if $f(n) = a_m n^m + .. a_1 n + a_0$ and $a_m > 0$ then $f(n) = O(n^m)$

Theorem2 (Big O ratio theorem): Let $f(n)$ and $g(n)$ be such that $\lim_{n\rightarrow\infty} f(n) / g(n)$ exists.
$f(n) = O(g(n))$ iff $\lim_{n\rightarrow\infty} f(n) / g(n) \le c$ for some finite positive constant c.

55

Example

- Example 1: $3n + 2 = O(n)$ because as $n \to \infty$ $(3n + 2)/n \to 3$.
- Example 2: $3n^2 + 5 \neq O(n)$ because as $n \to \infty$ $(3n^2 + 5)/n \to \infty$.

56

Big-Omega Definition

$f(n) = \Omega(g(n))$ iff positive constants c and n_0 exist such that:

$$cg(n) \leq f(n) \quad \text{for all } n \geq n_0.$$

57

Big-Omega Definition

$f(n) = \Omega(g(n))$ iff positive constants c and n_0 exist such that:

$$cg(n) \leq f(n) \quad \text{for all } n \geq n_0.$$

g grows at about the rate as f

58

Big-Omega Definition

$f(n) = \Omega(g(n))$ iff positive constants c and n_0 exist such that:

$$cg(n) \leq f(n) \quad \text{for all } n \geq n_0.$$

g grows at about the rate as f

Ω bounds from below

59

Big Theta Definition

$f(n) = \Theta(g(n))$ iff positive constants c_1, c_2, and n_0 exist such that:

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \text{for all } n \geq n_0.$$

60

Big Theta Definition

$f(n) = \Theta(g(n))$ iff positive constants c_1, c_2, and n_0 exist such that:

$$c_1 g(n) \leq \overbrace{f(n) \leq c_2 g(n)}^{O} \quad \text{for all } n \geq n_0.$$

61

Big Theta Definition

$f(n) = \Theta(g(n))$ iff positive constants c_1, c_2, and n_0 exist such that:

$$\underbrace{c_1 g(n) \le \overbrace{f(n) \le c_2 g(n)}^{O}}_{\Omega} \quad \text{for all } n \ge n_0.$$

62

Big Theta Definition

$f(n) = \Theta(g(n))$ iff positive constants c_1, c_2, and n_0 exist such that:

$$\underbrace{\underbrace{c_1 g(n) \le \overbrace{f(n) \le c_2 g(n)}^{O}}_{\Omega}}_{\Theta} \quad \text{for all } n \ge n_0.$$

63

Θ Example

$$n^2 + 4n + 20 = \Theta(?)$$

Θ Example

$$n^2 + 4n + 20 = \Theta(?)$$

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

Θ Example

$$n^2 + 4n + 20 = \Theta(?)$$

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

$$1n^2 \le n^2 + 4n + 20 \le n^2 + 4n^2 + 20n^2 = 25n^2$$

Θ Example

$$n^2 + 4n + 20 = \Theta(?)$$

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

Θ Example

$$n^2 + 4n + 20 = \Theta(?)$$

c_1

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

$c_1 = 1$

Θ Example

$$n^2 + 4n + 20 = \Theta(?)$$

c_1 c_2

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

$c_1 = 1$

$c_2 = 25$

Θ Example

$$n^2 + 4n + 20 = \Theta(?)$$

c_1 c_2 n_0

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

$c_1 = 1$

$c_2 = 25$

$n_0 = 1$

Θ Example

$$n^2 + 4n + 20 = \Theta(?)$$

c_1 c_2

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

$c_1 = 1$

$c_2 = 25$

$n_0 = 1$

Θ Example

$$n^2 + 4n + 20 = \Theta(?)$$

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

$c_1 = 1$

$c_2 = 25$

$n_0 = 1$

Θ Example

$$n^2 + 4n + 20 = \Theta(n^2)$$

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

$c_1 = 1$

$c_2 = 25$

$n_0 = 1$

Θ Example

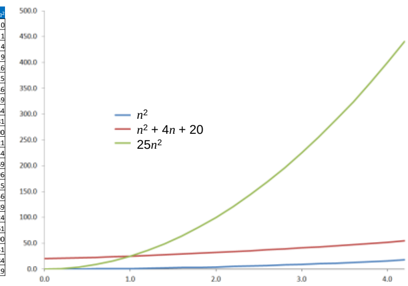
$$n^2 + 4n + 20 = \Theta(n^2)$$

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

$$c_1 = 1$$
$$c_2 = 25$$
$$n_0 = 1$$

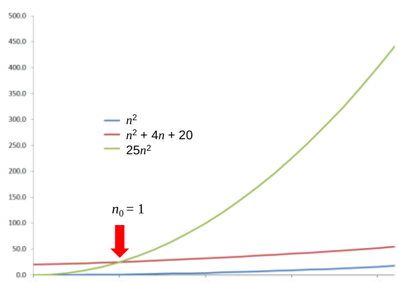
c_2 = 25 Range 0–4.6 0.2 increment

n	n^2	$n^2 + 4n + 20$	$25n^2$
0.0	0.0	20.0	0
0.2	0.0	20.8	1
0.4	0.2	21.8	4
0.6	0.4	22.8	9
0.8	0.6	23.8	16
1.0	1.0	25.0	25
1.2	1.4	26.2	36
1.4	2.0	27.6	49
1.6	2.6	29.0	64
1.8	3.2	30.4	81
2.0	4.0	32.0	100
2.2	4.8	33.6	121
2.4	5.8	35.4	144
2.6	6.8	37.2	169
2.8	7.8	39.0	196
3.0	9.0	41.0	225
3.2	10.2	43.0	256
3.4	11.6	45.2	289
3.6	13.0	47.4	324
3.8	14.4	49.6	361
4.0	16.0	52.0	400
4.2	17.6	54.4	441
4.4	19.4	57.0	484
4.6	21.2	59.6	529



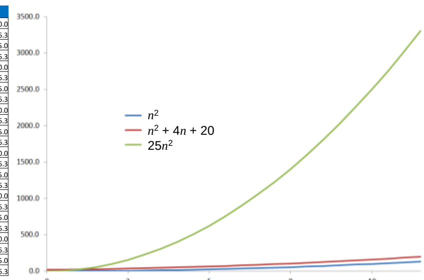
c_2 = 25 Range 0–4.6 0.2 increment

n	n^2	$n^2 + 4n + 20$	$25n^2$
0.0	0.0	20.0	0
0.2	0.0	20.8	1
0.4	0.2	21.8	4
0.6	0.4	22.8	9
0.8	0.6	23.8	16
1.0	1.0	25.0	25
1.2	1.4	26.2	36
1.4	2.0	27.6	49
1.6	2.6	29.0	64
1.8	3.2	30.4	81
2.0	4.0	32.0	100
2.2	4.8	33.6	121
2.4	5.8	35.4	144
2.6	6.8	37.2	169
2.8	7.8	39.0	196
3.0	9.0	41.0	225
3.2	10.2	43.0	256
3.4	11.6	45.2	289
3.6	13.0	47.4	324
3.8	14.4	49.6	361
4.0	16.0	52.0	400
4.2	17.6	54.4	441
4.4	19.4	57.0	484
4.6	21.2	59.6	529



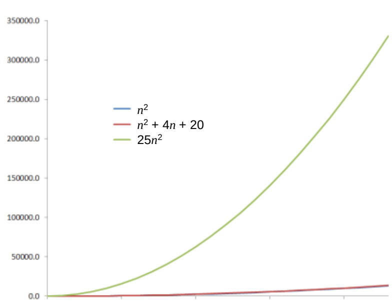
c_2 = 25 Range 0–12 0.5 increment

n	n^2	$n^2 + 4n + 20$	$25n^2$
0.0	0.0	20.0	0.0
0.5	0.3	22.3	6.3
1.0	1.0	25.0	25.0
1.5	2.3	28.3	56.3
2.0	4.0	32.0	100.0
2.5	6.3	36.3	156.3
3.0	9.0	41.0	225.0
3.5	12.3	46.3	306.3
4.0	16.0	52.0	400.0
4.5	20.3	58.3	506.3
5.0	25.0	65.0	625.0
5.5	30.3	72.3	756.3
6.0	36.0	80.0	900.0
6.5	42.3	88.3	1056.3
7.0	49.0	97.0	1225.0
7.5	56.3	106.3	1406.3
8.0	64.0	116.0	1600.0
8.5	72.3	126.3	1806.3
9.0	81.0	137.0	2025.0
9.5	90.3	148.3	2256.3
10.0	100.0	160.0	2500.0
10.5	110.3	172.3	2756.3
11.0	121.0	185.0	3025.0
11.5	132.3	198.3	3306.3

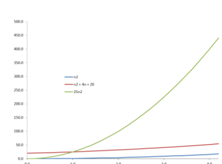


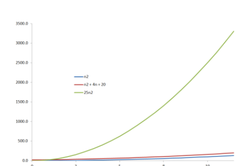
c_2 = 25 Range 0–115 5.0 increment

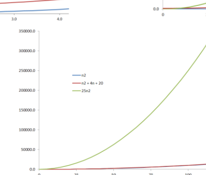
n	n^2	$n^2 + 4n + 20$	$25n^2$
0.0	0.0	20.0	0
5.0	25.0	65.0	625
10.0	100.0	160.0	2500
15.0	225.0	305.0	5625
20.0	400.0	500.0	10000
25.0	625.0	745.0	15625
30.0	900.0	1040.0	22500
35.0	1225.0	1385.0	30625
40.0	1600.0	1780.0	40000
45.0	2025.0	2225.0	50625
50.0	2500.0	2720.0	62500
55.0	3025.0	3265.0	75625
60.0	3600.0	3860.0	90000
65.0	4225.0	4505.0	105625
70.0	4900.0	5200.0	122500
75.0	5625.0	5945.0	140625
80.0	6400.0	6740.0	160000
85.0	7225.0	7585.0	180625
90.0	8100.0	8480.0	202500
95.0	9025.0	9425.0	225625
100.0	10000.0	10420.0	250000
105.0	11025.0	11465.0	275625
110.0	12100.0	12560.0	302500
115.0	13225.0	13705.0	330625



Summary, c_2 = 25

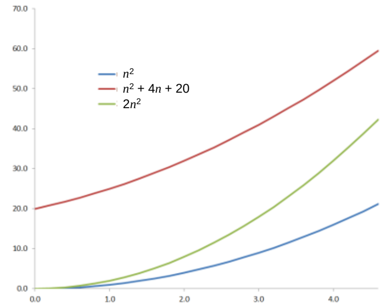






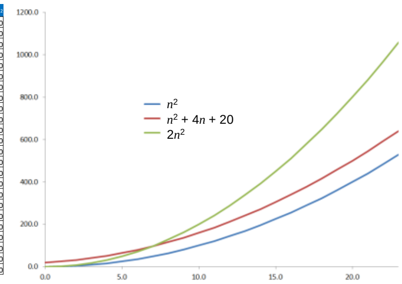
$c_2 = 2$ Range 0–4.6 0.2 increment

n	n^2	$n^2 + 4n + 20$	$2n^2$
0.0	0.0	20.0	0.0
0.2	0.0	20.8	0.1
0.4	0.2	21.8	0.3
0.6	0.4	22.8	0.7
0.8	0.6	23.8	1.3
1.0	1.0	25.0	2.0
1.2	1.4	26.2	2.9
1.4	2.0	27.6	3.9
1.6	2.6	29.0	5.1
1.8	3.2	30.4	6.5
2.0	4.0	32.0	8.0
2.2	4.8	33.6	9.7
2.4	5.8	35.4	11.5
2.6	6.8	37.2	13.5
2.8	7.8	39.0	15.7
3.0	9.0	41.0	18.0
3.2	10.2	43.0	20.5
3.4	11.6	45.2	23.1
3.6	13.0	47.4	25.9
3.8	14.4	49.6	28.9
4.0	16.0	52.0	32.0
4.2	17.6	54.4	35.3
4.4	19.4	57.0	38.7
4.6	21.2	59.6	42.3



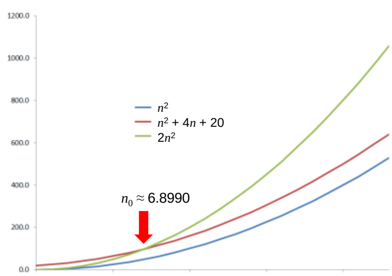
$c_2 = 2$ Range 0–23 1.0 increment

n	n^2	$n^2 + 4n + 20$	$2n^2$
0.0	0.0	20.0	0.0
1.0	1.0	25.0	2.0
2.0	4.0	32.0	8.0
3.0	9.0	41.0	18.0
4.0	16.0	52.0	32.0
5.0	25.0	65.0	50.0
6.0	36.0	80.0	72.0
7.0	49.0	97.0	98.0
8.0	64.0	116.0	128.0
9.0	81.0	137.0	162.0
10.0	100.0	160.0	200.0
11.0	121.0	185.0	242.0
12.0	144.0	212.0	288.0
13.0	169.0	241.0	338.0
14.0	196.0	272.0	392.0
15.0	225.0	305.0	450.0
16.0	256.0	340.0	512.0
17.0	289.0	377.0	578.0
18.0	324.0	416.0	648.0
19.0	361.0	457.0	722.0
20.0	400.0	500.0	800.0
21.0	441.0	545.0	882.0
22.0	484.0	592.0	968.0
23.0	529.0	641.0	1058.0



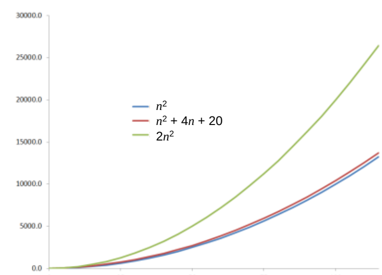
$c_2 = 2$ Range 0–23 1.0 increment

n	n^2	$n^2 + 4n + 20$	$2n^2$
0.0	0.0	20.0	0.0
1.0	1.0	25.0	2.0
2.0	4.0	32.0	8.0
3.0	9.0	41.0	18.0
4.0	16.0	52.0	32.0
5.0	25.0	65.0	50.0
6.0	36.0	80.0	72.0
7.0	49.0	97.0	98.0
8.0	64.0	116.0	128.0
9.0	81.0	137.0	162.0
10.0	100.0	160.0	200.0
11.0	121.0	185.0	242.0
12.0	144.0	212.0	288.0
13.0	169.0	241.0	338.0
14.0	196.0	272.0	392.0
15.0	225.0	305.0	450.0
16.0	256.0	340.0	512.0
17.0	289.0	377.0	578.0
18.0	324.0	416.0	648.0
19.0	361.0	457.0	722.0
20.0	400.0	500.0	800.0
21.0	441.0	545.0	882.0
22.0	484.0	592.0	968.0
23.0	529.0	641.0	1058.0

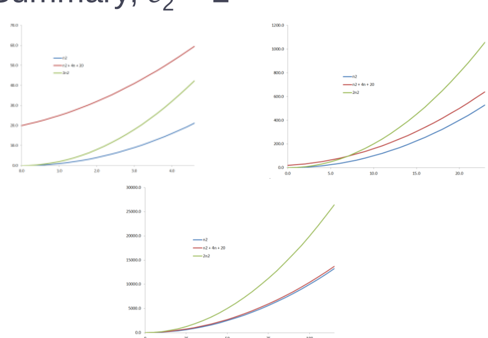


$c_2 = 2$ Range 0–115 5.0 increment

n	n^2	$n^2 + 4n + 20$	$2n^2$
0.0	0.0	20.0	0.0
5.0	25.0	65.0	50.0
10.0	100.0	160.0	200.0
15.0	225.0	305.0	450.0
20.0	400.0	500.0	800.0
25.0	625.0	745.0	1250.0
30.0	900.0	1040.0	1800.0
35.0	1225.0	1385.0	2450.0
40.0	1600.0	1780.0	3200.0
45.0	2025.0	2225.0	4050.0
50.0	2500.0	2720.0	5000.0
55.0	3025.0	3265.0	6050.0
60.0	3600.0	3860.0	7200.0
65.0	4225.0	4505.0	8450.0
70.0	4900.0	5200.0	9800.0
75.0	5625.0	5945.0	11250.0
80.0	6400.0	6740.0	12800.0
85.0	7225.0	7585.0	14450.0
90.0	8100.0	8480.0	16200.0
95.0	9025.0	9425.0	18050.0
100.0	10000.0	10420.0	20000.0
105.0	11025.0	11465.0	22050.0
110.0	12100.0	12560.0	24200.0
115.0	13225.0	13705.0	26450.0

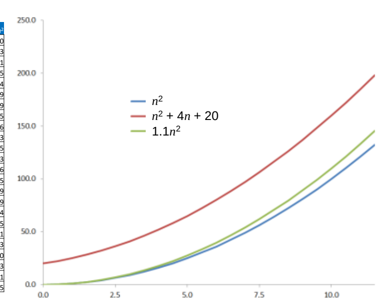


Summary, $c_2 = 2$



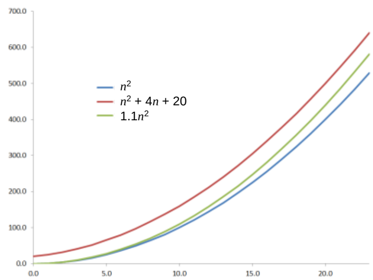
$c_2 = 1.1$ Range 0–11.5 0.5 increment

n	n^2	$n^2 + 4n + 20$	$1.1n^2$
0.0	0.0	20.0	0.0
0.5	0.3	22.3	0.3
1.0	1.0	25.0	1.1
1.5	2.3	28.3	2.5
2.0	4.0	32.0	4.4
2.5	6.3	36.3	6.9
3.0	9.0	41.0	9.9
3.5	12.3	46.3	13.5
4.0	16.0	52.0	17.6
4.5	20.3	58.3	22.3
5.0	25.0	65.0	27.5
5.5	30.3	72.3	33.3
6.0	36.0	80.0	39.6
6.5	42.3	88.3	46.5
7.0	49.0	97.0	53.9
7.5	56.3	106.3	61.9
8.0	64.0	116.0	70.4
8.5	72.3	126.3	79.5
9.0	81.0	137.0	89.1
9.5	90.3	148.3	99.3
10.0	100.0	160.0	110.0
10.5	110.3	172.3	121.3
11.0	121.0	185.0	133.1
11.5	132.3	198.3	145.5



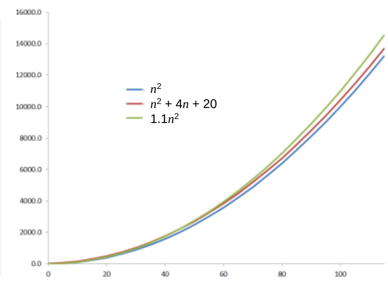
c_2 = 1.1 Range 0–23 1.0 increment

n	n^2	$n^2 + 4n + 20$	$1.1n^2$
0.0	0.0	20.0	0.0
1.0	1.0	25.0	1.1
2.0	4.0	32.0	4.4
3.0	9.0	41.0	9.9
4.0	16.0	52.0	17.6
5.0	25.0	65.0	27.5
6.0	36.0	80.0	39.6
7.0	49.0	97.0	53.9
8.0	64.0	116.0	70.4
9.0	81.0	137.0	89.1
10.0	100.0	160.0	110.0
11.0	121.0	185.0	133.1
12.0	144.0	212.0	158.4
13.0	169.0	241.0	185.9
14.0	196.0	272.0	215.6
15.0	225.0	305.0	247.5
16.0	256.0	340.0	281.6
17.0	289.0	377.0	317.9
18.0	324.0	416.0	356.4
19.0	361.0	457.0	397.1
20.0	400.0	500.0	440.0
21.0	441.0	545.0	485.1
22.0	484.0	592.0	532.4
23.0	529.0	641.0	581.9



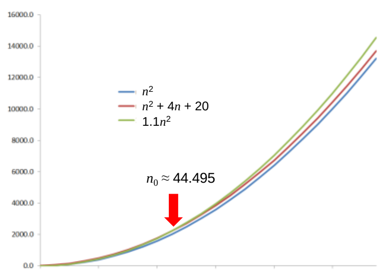
c_2 = 1.1 Range 0–115 5.0 increment

n	n^2	$n^2 + 4n + 20$	$1.1n^2$
0.0	0.0	20.0	0.0
5.0	25.0	65.0	27.5
10.0	100.0	160.0	110.0
15.0	225.0	305.0	247.5
20.0	400.0	500.0	440.0
25.0	625.0	745.0	687.5
30.0	900.0	1040.0	990.0
35.0	1225.0	1385.0	1347.5
40.0	1600.0	1780.0	1760.0
45.0	2025.0	2225.0	2227.5
50.0	2500.0	2720.0	2750.0
55.0	3025.0	3265.0	3327.5
60.0	3600.0	3860.0	3960.0
65.0	4225.0	4505.0	4647.5
70.0	4900.0	5200.0	5390.0
75.0	5625.0	5945.0	6187.5
80.0	6400.0	6740.0	7040.0
85.0	7225.0	7585.0	7947.5
90.0	8100.0	8480.0	8910.0
95.0	9025.0	9425.0	9927.5
100.0	10000.0	10420.0	11000.0
105.0	11025.0	11465.0	12127.5
110.0	12100.0	12560.0	13310.0
115.0	13225.0	13705.0	14547.5



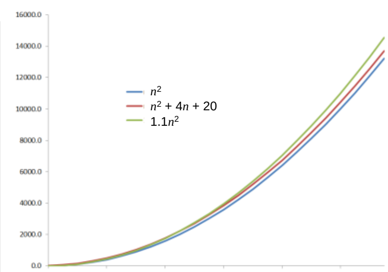
c_2 = 1.1 Range 0–115 5.0 increment

n	n^2	$n^2 + 4n + 20$	$1.1n^2$
0.0	0.0	20.0	0.0
5.0	25.0	65.0	27.5
10.0	100.0	160.0	110.0
15.0	225.0	305.0	247.5
20.0	400.0	500.0	440.0
25.0	625.0	745.0	687.5
30.0	900.0	1040.0	990.0
35.0	1225.0	1385.0	1347.5
40.0	1600.0	1780.0	1760.0
45.0	2025.0	2225.0	2227.5
50.0	2500.0	2720.0	2750.0
55.0	3025.0	3265.0	3327.5
60.0	3600.0	3860.0	3960.0
65.0	4225.0	4505.0	4647.5
70.0	4900.0	5200.0	5390.0
75.0	5625.0	5945.0	6187.5
80.0	6400.0	6740.0	7040.0
85.0	7225.0	7585.0	7947.5
90.0	8100.0	8480.0	8910.0
95.0	9025.0	9425.0	9927.5
100.0	10000.0	10420.0	11000.0
105.0	11025.0	11465.0	12127.5
110.0	12100.0	12560.0	13310.0
115.0	13225.0	13705.0	14547.5

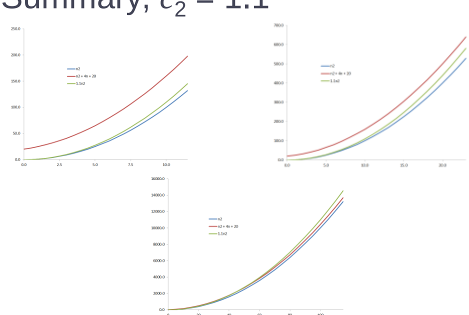


c_2 = 1.1 Range 0–115 5.0 increment

n	n^2	$n^2 + 4n + 20$	$1.1n^2$
0.0	0.0	20.0	0.0
5.0	25.0	65.0	27.5
10.0	100.0	160.0	110.0
15.0	225.0	305.0	247.5
20.0	400.0	500.0	440.0
25.0	625.0	745.0	687.5
30.0	900.0	1040.0	990.0
35.0	1225.0	1385.0	1347.5
40.0	1600.0	1780.0	1760.0
45.0	2025.0	2225.0	2227.5
50.0	2500.0	2720.0	2750.0
55.0	3025.0	3265.0	3327.5
60.0	3600.0	3860.0	3960.0
65.0	4225.0	4505.0	4647.5
70.0	4900.0	5200.0	5390.0
75.0	5625.0	5945.0	6187.5
80.0	6400.0	6740.0	7040.0
85.0	7225.0	7585.0	7947.5
90.0	8100.0	8480.0	8910.0
95.0	9025.0	9425.0	9927.5
100.0	10000.0	10420.0	11000.0
105.0	11025.0	11465.0	12127.5
110.0	12100.0	12560.0	13310.0
115.0	13225.0	13705.0	14547.5



Summary, c_2 = 1.1



Θ Example, revisited

$$n^2 + 4n + 20 = \Theta(n^2)$$

$$1n^2 \le n^2 + 4n + 20 \le 25n^2 \quad \text{for all } n \ge 1$$

$$c_1 = 1 \quad c_2 = 25 \quad n_0 = 1$$

Θ Example, revisited

$$n^2 + 4n + 20 = \Theta(n^2)$$

$1n^2 \le n^2 + 4n + 20 \le 25n^2$ for all $n \ge 1$

$c_1 = 1 \quad c_2 = 25 \quad n_0 = 1$

$1n^2 \le n^2 + 4n + 20 \le 1.1n^2$ for all $n \ge 44.5$

$c_1 = 1 \quad c_2 = 1.1 \quad n_0 = 44.5$

Θ Example, revisited

$$n^2 + 4n + 20 = \Theta(n^2)$$

$c_1 = 1 \quad c_2 = 25 \quad n_0 = 1$

$c_1 = 1 \quad c_2 = 1.1 \quad n_0 = 44.5$

$c_1 = 1 \quad c_2 = 1.0001 \quad n_0 = 40{,}005$

Example

- Example 1: Prove that $f(n) = 3n + 2 = \Theta(n)$ We have already shown that $f(n) = O(n)$.
- We just need to prove that $f(n)$ is $\Omega(n)$. That is to show that $cg(n) \le f(n)$ $n \ge n_0$.
- This is easy because $n \le 3n + 2$ for all $n \ge 0$
- Example 2: Prove that $3n + 3 \ne \Theta(n^2)$

95

More Helpful Theorems

Theorem: if $f(n) = a_m n^m + .. a_1 n + a_0$
and $a_m > 0$
then $f(n) = \Theta(n^m)$

Theorem (Ratio for Θ) : Let $f(n)$ and $g(n)$ be such that $\lim_{n\to\infty} f(n)/g(n)$ and $\lim_{n\to\infty} g(n)/f(n)$ exist then $f(n) = \Theta(g(n))$
iff $\lim_{n\to\infty} f(n)/g(n) \le c$ and
$\lim_{n\to\infty} g(n)/f(n) \le c$ for some finite positive constant c.

96

Polynomial Functions and Θ

Theorem:

If $f(n) = a_m n^m + .. a_1 n + a_0$ and $a_m > 0$,
then $f(n) = \Theta(n^m)$.

Ratio Theorem for Θ

Let

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

where c is a constant or ∞.

- If $0 \le c < \infty$, then $f(n) = O(g(n))$
- If $0 < c \le \infty$, then $f(n) = \Omega(g(n))$
- If $0 < c < \infty$, then $f(n) = \Theta(g(n))$

98

Example

- $3n + 2 = \Theta(n)$
  because as $n \to\infty$ $(3n + 2)/ n = 3$
  and as $n \to\infty$ $n/(3n + 2) = 1/3 \le 3$.

99

Little o Definition

$$f(n) = o(g(n)) \text{ iff } f(n) = O(g(n)) \text{ and } f(n) \neq \Theta(g(n))$$

100

Little o Definition

$$f(n) = o(g(n)) \text{ iff } f(n) = O(g(n)) \text{ and } f(n) \neq \Theta(g(n))$$

$n^2 + 4n + 20 = O(n^2)$ and
$n^2 + 4n + 20 = \Theta(n^2)$, so
$n^2 + 4n + 20 \neq o(n^2)$

101

Little o Definition

$$f(n) = o(g(n)) \text{ iff } f(n) = O(g(n)) \text{ and } f(n) \neq \Theta(g(n))$$

$5n + 3 = O(n^2)$ but
$5n + 3 \neq \Theta(n^2)$, so
$5n + 3 = o(n^2)$

102

Meaning of the various growth functions

Mathematical Expression	Relative Rates of Growth
$f(n) = O(g(n))$	$f(n) \le g(n)$
$f(n) = \Omega(g(n))$	$f(n) \ge g(n)$
$f(n) = \Theta(g(n))$	$f(n) = g(n)$
$f(n) = o(g(n))$	$f(n) < g(n)$

104

Common asymptotic functions

- In order of magnitude
  1. 1
  2. $\log n$
  3. n
  4. $n \log n$
  5. n^2
  6. n^3
  7. 2^n
  8. $n!$

105

Common asymptotic functions

- In order of magnitude
  1. 1 *Array or vector access*
  2. $\log n$
  3. n
  4. $n \log n$
  5. n^2
  6. n^3
  7. 2^n
  8. $n!$

106

Common asymptotic functions

- In order of magnitude
  1. 1 *Array or vector access*
  2. $\log n$ *Binary search*
  3. n
  4. $n \log n$
  5. n^2
  6. n^3
  7. 2^n
  8. $n!$

107

Common asymptotic functions

- In order of magnitude
  1. 1 *Array or vector access*
  2. $\log n$ *Binary search*
  3. n *Sequential search, verifying ordering*
  4. $n \log n$
  5. n^2
  6. n^3
  7. 2^n
  8. $n!$

108

Common asymptotic functions

- In order of magnitude
  1. 1 *Array or vector access*
  2. $\log n$ *Binary search*
  3. n *Sequential search, verifying ordering*
  4. $n \log n$ *Fast sorting*
  5. n^2
  6. n^3
  7. 2^n
  8. $n!$

109

Common asymptotic functions

- In order of magnitude
  1. 1 *Array or vector access*
  2. $\log n$ *Binary search*
  3. n *Sequential search, verifying ordering*
  4. $n \log n$ *Fast sorting*
  5. n^2 *Simple sorting, shortest path*
  6. n^3
  7. 2^n
  8. $n!$

110

Common asymptotic functions

- In order of magnitude
  1. 1 *Array or vector access*
  2. $\log n$ *Binary search*
  3. n *Sequential search, verifying ordering*
  4. $n \log n$ *Fast sorting*
  5. n^2 *Simple sorting, shortest path*
  6. n^3 *Matrix multiplication*
  7. 2^n
  8. $n!$

111

Common asymptotic functions

- In order of magnitude
  1. 1 *Array or vector access*
  2. $\log n$ *Binary search*
  3. n *Sequential search, verifying ordering*
  4. $n \log n$ *Fast sorting*
  5. n^2 *Simple sorting, shortest path*
  6. n^3 *Matrix multiplication*
  7. 2^n *Hamiltonian circuit, longest path*
  8. $n!$

112

Common asymptotic functions

- In order of magnitude
  1. 1 *Array or vector access*
  2. $\log n$ *Binary search*
  3. n *Sequential search, verifying ordering*
  4. $n \log n$ *Fast sorting*
  5. n^2 *Simple sorting, shortest path*
  6. n^3 *Matrix multiplication*
  7. 2^n *Hamiltonian circuit, longest path*
  8. $n!$ *List permutations*

113

Common asymptotic functions

- In order of magnitude
  1. 1
  2. $\log n$
  3. n
  4. $n \log n$
  5. n^2
  6. n^3
  7. 2^n
  8. $n!$

114

Common asymptotic functions

- In order of magnitude
  1. 1
  2. $\log n$
  3. n
  4. $n \log n$
  5. n^2
  6. n^3
  7. 2^n
  8. $n!$

115

Common asymptotic functions

- In order of magnitude
  1. 1
  2. $\log n$
  3. n
  4. $n \log n$
  5. n^2
  6. n^3
  7. 2^n
  8. $n!$

Tractable

116

Common asymptotic functions

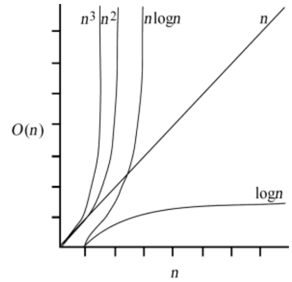
- In order of magnitude
  1. 1
  2. $\log n$
  3. n
  4. $n \log n$
  5. n^2
  6. n^3
  7. 2^n
  8. $n!$

Tractable

Intractable

117

Graph of Asymptotic functions



118

Example

Consider $f(n) = 6 \cdot 2^n + n^2$.

119

Example

Consider $f(n) = 6 \cdot 2^n + n^2$.

$f(n) = 6 \cdot 2^n + n^2 \leq 6 \cdot 2^n + 2^n = 7 \cdot 2^n$, for all $n \geq 4$

therefore $f(n) = O(2^n)$

120

Example

Consider $f(n) = 6 \cdot 2^n + n^2$.

$f(n) = 6 \cdot 2^n + n^2 \leq 6 \cdot 2^n + 2^n = 7 \cdot 2^n$, for all $n \geq 4$

therefore $f(n) = O(2^n)$

$1 \cdot 2^n \leq 6 \cdot 2^n + n^2 \leq 6 \cdot 2^n + 2^n = 7 \cdot 2^n$, for all $n \geq 4$

therefore $f(n) = \Theta(2^n)$

121

Example

Consider $f(n) = 3n^2 + 5n + 10$.

$f(n) = 3n^2 + 5n + 10 \leq 3 \cdot 2^n + 5 \cdot 2^n + 10 \cdot 2^n = 18 \cdot 2^n$,

for all $n \geq 4$, therefore $f(n) = O(2^n)$

Suppose $f(n) = \Theta(2^n)$. Then there exists positive constant c such that

$$c \cdot 2^n \leq 3n^2 + 5n + 10, \text{ for all } n \geq n_0$$

So $c \cdot 2^n - 3n^2 - 5n \leq 10$

122

Execution Time Comparison

- A particular algorithm can solve a problem of input size 1,000 in 20 milliseconds. Estimate the size of the problem the algorithm can solve in 1 minute if the algorithm's asymptotic complexity is
  - $\Theta(n)$
  - $\Theta(n^2)$

- Input size 1,000 → 20 msec
- 1 minute = 60,000 msec
- $\Theta(n)$:
  $$\frac{1{,}000}{20} = \frac{n}{60{,}000}$$

- Input size 1,000 → 20 msec
- 1 minute = 60,000 msec
- $\Theta(n)$:
  $$\frac{1{,}000}{20} = \frac{n}{60{,}000} \rightarrow 20n = 60{,}000{,}000$$

- Input size 1,000 → 20 msec
- 1 minute = 60,000 msec
- $\Theta(n)$:
  $$\frac{1{,}000}{20} = \frac{n}{60{,}000} \rightarrow 20n = 60{,}000{,}000$$
  $$\rightarrow n = 3{,}000{,}000$$

- Input size 1,000 → 20 msec
- 1 minute = 60,000 msec
- $\Theta(n)$:
  $$\frac{1{,}000}{20} = \frac{n}{60{,}000} \rightarrow 20n = 60{,}000{,}000$$
  $$\rightarrow n = 3{,}000{,}000$$
- $\Theta(n^2)$:
  $$\frac{1{,}000^2}{20} = \frac{n^2}{60{,}000}$$

- Input size 1,000 → 20 msec
- 1 minute = 60,000 msec
- $\Theta(n)$:
  $$\frac{1{,}000}{20} = \frac{n}{60{,}000} \rightarrow 20n = 60{,}000{,}000$$
  $$\rightarrow n = 3{,}000{,}000$$
- $\Theta(n^2)$:
  $$\frac{1{,}000^2}{20} = \frac{n^2}{60{,}000} \rightarrow 20n^2 = 60{,}000{,}000{,}000$$

- Input size 1,000 → 20 msec
- 1 minute = 60,000 msec
- $\Theta(n)$:
  $$\frac{1{,}000}{20} = \frac{n}{60{,}000} \rightarrow 20n = 60{,}000{,}000$$
  $$\rightarrow n = 3{,}000{,}000$$
- $\Theta(n^2)$:
  $$\frac{1{,}000^2}{20} = \frac{n^2}{60{,}000} \rightarrow 20n^2 = 60{,}000{,}000{,}000$$
  $$\rightarrow n^2 = 3{,}000{,}000{,}000$$

- Input size 1,000 → 20 msec
- 1 minute = 60,000 msec
- $\Theta(n)$:

$$\frac{1{,}000}{20} = \frac{n}{60{,}000} \rightarrow 20n = 60{,}000{,}000$$
$$\rightarrow n = 3{,}000{,}000$$

- $\Theta(n^2)$:

$$\frac{1{,}000^2}{20} = \frac{n^2}{60{,}000} \rightarrow 20n^2 = 60{,}000{,}000{,}000$$
$$\rightarrow n^2 = 3{,}000{,}000{,}000$$
$$\rightarrow n = \sqrt{3{,}000{,}000{,}000}$$

- Input size 1,000 → 20 msec
- 1 minute = 60,000 msec
- $\Theta(n)$:

Over 50 times larger

$$\frac{1{,}000}{20} = \frac{n}{60{,}000} \rightarrow 20n = 60{,}000{,}000$$
$$\rightarrow n = 3{,}000{,}000$$

- $\Theta(n^2)$:

$$\frac{1{,}000^2}{20} = \frac{n^2}{60{,}000} \rightarrow 20n^2 = 60{,}000{,}000{,}000$$
$$\rightarrow n^2 = 3{,}000{,}000{,}000$$
$$\rightarrow n = \sqrt{3{,}000{,}000{,}000}$$
$$\rightarrow n \approx 54{,}772$$

Execution Time Comparison

- A particular algorithm can solve a problem of input size 1,000 in 20 milliseconds. Estimate the size of the problem the algorithm can solve in 1 minute if the algorithm's asymptotic complexity is
  - $\Theta(n)$ 3,000,000
  - $\Theta(n^2)$ 54,772
  - $\Theta(\log_2 n)$

- Input size 1,000 → 20 msec
- 1 minute = 60,000 msec
- $\Theta(\log_2 n)$:

$$\frac{\log_2 1{,}000}{20} = \frac{\log_2 n}{60{,}000} \rightarrow 20 \log_2 n \approx 597{,}947$$
$$\rightarrow \log_2 n \approx 29{,}897$$
$$\rightarrow n \approx 2^{29{,}897} \approx 7.8 \times 10^{8{,}999}$$

Execution Time Comparison

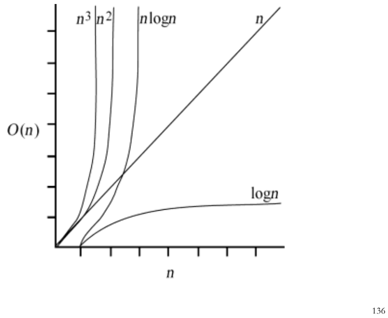
- A particular algorithm can solve a problem of input size 1,000 in 20 milliseconds. Estimate the size of the problem the algorithm can solve in 1 minute if the algorithm's asymptotic complexity is
  - $\Theta(n)$ 3,000,000
  - $\Theta(n^2)$ 54,772
  - $\Theta(\log_2 n)$ $7.8 \times 10^{8{,}999}$

Execution Time Comparison

- A particular algorithm can solve a problem of input size 1,000 in 20 milliseconds. Estimate the size of the problem the algorithm can solve in 1 minute if the algorithm's asymptotic complexity is
  - $\Theta(n)$ 3,000,000
  - $\Theta(n^2)$ 54,772
  - $\Theta(\log_2 n)$ $7.8 \times 10^{8{,}999}$

Estimated number of atoms in the observable universe: 10^{78} to 10^{82}

Graph of Asymptotic functions



Logarithms

- All logarithms are asymptotically equivalent
  - $\log_{10} n = \Theta(\log_2 n)$
  - $\log_2 n = \Theta(\log_{10} n)$

$$\log_2 1024 = 10$$

$$\log_{10} 1024 \approx 3.01$$

Logarithms

- All logarithms are asymptotically equivalent
  - $\log_{10} n = \Theta(\log_2 n)$
  - $\log_2 n = \Theta(\log_{10} n)$

$$\log_2 1024 = 10$$

$$\log_{10} 1024 \approx 3.01$$

$$\log_{10} x \leq \log_2 x$$

Logarithms

- All logarithms are asymptotically equivalent
  - $\log_{10} n = \Theta(\log_2 n)$
  - $\log_2 n = \Theta(\log_{10} n)$

$$c_1 \cdot \log_{10} n \leq \log_2 n \leq c_2 \cdot \log_{10} n$$
for all $n \geq 2$

Logarithms

- All logarithms are asymptotically equivalent
  - $\log_{10} n = \Theta(\log_2 n)$
  - $\log_2 n = \Theta(\log_{10} n)$

$$c_1 \cdot \log_{10} n \leq \log_2 n \leq c_2 \cdot \log_{10} n$$
for all $n \geq 2$

$c_1 = 1$
$c_2 = ?$

Logarithms

All logarithms are asymptotically equivalent

$$1 \cdot \log_{10} n \leq \log_2 n \leq c_2 \cdot \log_{10} n$$
for all $n \geq 2$

Logarithms

All logarithms are asymptotically equivalent

$$1 \cdot \log_{10} n \le \log_2 n \le c_2 \cdot \log_{10} n$$

for all $n \ge 2$

- $\log_b x = \frac{\log_a x}{\log_a b}$

Logarithms

All logarithms are asymptotically equivalent

$$1 \cdot \log_{10} n \le \log_2 n \le c_2 \cdot \log_{10} n$$

for all $n \ge 2$

- $\log_b x = \frac{\log_a x}{\log_a b}$

$$\log_b x = \frac{1}{\log_a b} \log_a x$$

Logarithms

All logarithms are asymptotically equivalent

$$1 \cdot \log_{10} n \le \log_2 n \le c_2 \cdot \log_{10} n$$

for all $n \ge 2$

- $\log_b x = \frac{\log_a x}{\log_a b}$ *a constant factor*

$$\log_b x = \boxed{\frac{1}{\log_a b}} \log_a x$$

Logarithms

All logarithms are asymptotically equivalent

$$1 \cdot \log_{10} n \le \log_2 n \le c_2 \cdot \log_{10} n$$

for all $n \ge 2$

- $\log_b x = \frac{\log_a x}{\log_a b}$ *a constant factor*

$$\log_b x = \boxed{\frac{1}{\log_a b}} \log_a x \qquad c_2 = \frac{1}{\log_{10} 2}$$

Logarithms

All logarithms are asymptotically equivalent

$$1 \cdot \log_{10} n \le \log_2 n \le c_2 \cdot \log_{10} n$$

for all $n \ge 2$

- $\log_b x = \frac{\log_a x}{\log_a b}$ *a constant factor*

$$\log_b x = \boxed{\frac{1}{\log_a b}} \log_a x \qquad c_2 = \frac{1}{\log_{10} 2} \approx 0.301$$

Execution on a computer that executes 1 billion instructions per second

n	$f(n) = n$	$f(n) = \log_2 n$	$f(n) = n \log_2 n$	$f(n) = n^2$	$f(n) = 2^n$
10	0.01 μs	0.003 μs	0.033 μs	0.1 μs	1 μs
50	0.05 μs	0.006 μs	0.282 μs	2.5 μs	13 days
100	0.10 μs	0.007 μs	0.664 μs	10 μs	4×10^{13} years

147

Binary Search

Example from book

Sec. 3.5 Calculating the Running Time for a Program 75

```
// Return the position of an element in sorted array "A" of
// size "n" with value "K".  If "K" is not in "A", return
// the value "n".
int binary(int A[], int n, int K) {
  int l = -1;
  int r = n;            // l and r are beyond array bounds
  while (l+1 != r) {  // Stop when l and r meet
    int i = (l+r)/2;  // Check middle of remaining subarray
    if (K < A[i]) r = i;     // In left half
    if (K == A[i]) return i; // Found it
    if (K > A[i]) l = i;     // In right half
  }
  return n; // Search value not in A
}
```

Figure 3.5 Implementation for binary search.

Binary Search

- *Spacing*

Sec. 3.5 Calculating the Running Time for a Program 75

```
// Return the position of an element in sorted array "A" of
// size "n" with value "K".  If "K" is not in "A", return
// the value "n".
int binary(int A[], int n, int K) {
  int l = -1;
  int r = n;            // l and r are beyond array bounds
  while (l+1 != r) {  // Stop when l and r meet
    int i = (l+r)/2;  // Check middle of remaining subarray
    if (K < A[i]) r = i;     // In left half
    if (K == A[i]) return i; // Found it
    if (K > A[i]) l = i;     // In right half
  }
  return n; // Search value not in A
}
```

Figure 3.5 Implementation for binary search.

Binary Search

- *Spacing*

Sec. 3.5 Calculating the Running Time for a Program 75

```
// Return the position of an element in sorted array "A" of
// size "n" with value "K".  If "K" is not in "A", return
// the value "n".
int binary(int A[], int n, int K) {
  int l = -1;
  int r = n;            // l and r are beyond array bounds
  while (l+1 != r) {  // Stop when l and r meet
    int i = (l+r)/2;  // Check middle of remaining subarray
    if (K < A[i]) r = i;     // In left half
    if (K == A[i]) return i; // Found it
    if (K > A[i]) l = i;     // In right half
  }
  return n; // Search value not in A
}
```

Figure 3.5 Implementation for binary search.

Binary Search

- *Spacing*
- *Indentation*

Sec. 3.5 Calculating the Running Time for a Program 75

```
// Return the position of an element in sorted array "A" of
// size "n" with value "K".  If "K" is not in "A", return
// the value "n".
int binary(int A[], int n, int K) {
  int l = -1;
  int r = n;            // l and r are beyond array bounds
  while (l+1 != r) {  // Stop when l and r meet
    int i = (l+r)/2;  // Check middle of remaining subarray
    if (K < A[i]) r = i;     // In left half
    if (K == A[i]) return i; // Found it
    if (K > A[i]) l = i;     // In right half
  }
  return n; // Search value not in A
}
```

Figure 3.5 Implementation for binary search.

Binary Search

- *Spacing*
- *Indentation*
- *"Hidden" logic*

Sec. 3.5 Calculating the Running Time for a Program 75

```
// Return the position of an element in sorted array "A" of
// size "n" with value "K".  If "K" is not in "A", return
// the value "n".
int binary(int A[], int n, int K) {
  int l = -1;
  int r = n;            // l and r are beyond array bounds
  while (l+1 != r) {  // Stop when l and r meet
    int i = (l+r)/2;  // Check middle of remaining subarray
    if (K < A[i]) r = i;     // In left half
    if (K == A[i]) return i; // Found it
    if (K > A[i]) l = i;     // In right half
  }
  return n; // Search value not in A
}
```

Figure 3.5 Implementation for binary search.

Binary Search

- *Spacing*
- *Indentation*
- *"Hidden" logic*
- *Variable name*

Sec. 3.5 Calculating the Running Time for a Program 75

```
// Return the position of an element in sorted array "A" of
// size "n" with value "K".  If "K" is not in "A", return
// the value "n".
int binary(int A[], int n, int K) {
  int l = -1;
  int r = n;            // l and r are beyond array bounds
  while (l+1 != r) {  // Stop when l and r meet
    int i = (l+r)/2;  // Check middle of remaining subarray
    if (K < A[i]) r = i;     // In left half
    if (K == A[i]) return i; // Found it
    if (K > A[i]) l = i;     // In right half
  }
  return n; // Search value not in A
}
```

Figure 3.5 Implementation for binary search.

Binary Search

- *Spacing*
- *Indentation*
- *"Hidden" logic*
- *Variable name*
- *Math*

Sec. 3.5 Calculating the Running Time for a Program 75

```
// Return the position of an element in sorted array "A" of
// size "n" with value "K". If "K" is not in "A", return
// the value "n".
int binary(int A[], int n, int K) {
  int l = -1;
  int r = n;            // l and r are beyond array bounds
  while (l+1 != r) {    // Stop when l and r meet
    int i = (l+r)/2;    // Check middle of remaining subarray
    if (K < A[i]) r = i;      // In left half
    if (K == A[i]) return i;  // Found it
    if (K > A[i]) l = i;      // In right half
  }
  return n; // Search value not in A
}
```

Figure 3.5 Implementation for binary search.

Binary Search

- *Spacing*
- *Indentation*
- *"Hidden" logic*
- *Variable name*
- *Math*

Sec. 3.5 Calculating the Running Time for a Program 75

```
// Return the position of an element in sorted array "A" of
// size "n" with value "K". If "K" is not in "A", return
// the value "n".
int binary(int A[], int n, int K) {
  int l = -1;
  int r = n;            // l and r are beyond array bounds
  while (l+1 != r) {    // Stop when l and r meet
    int i = (l+r)/2;    // Check middle of remaining subarray
    if (K < A[i]) r = i;      // In left half
    if (K == A[i]) return i;  // Found it
    if (K > A[i]) l = i;      // In right half
  }
  return n; // Search value not in A
}
```

Figure 3.5 Implementation for binary search.


```
l + (r − l)/2
```

```
// Return the position of an element in sorted array "A"
// of size "n" with value "K". If "K" is not in "A",
// return the value "n".
int binary(int A[], int n, int K) {
    int lf = -1;
    int rt = n;              // lf and rt are beyond array bounds
    while (lf + 1 != rt) {      // Stop when lf and rt meet
        int mid = lf + (rt - lf)/2; // Compute middle index
        if (K < A[mid])
            rt = mid;           // In left half
        if (K == A[mid])
            return mid;         // Found it
        if (K > A[mid])
            lf = mid;           // In right half
    }
    return n;                   // Search value not in A
}
```

Limitations of Asymptotic Analysis

- Its use is not appropriate for small amounts of input
  - For small amounts of input, use the simplest algorithm
- The constants involved with asymptotic analysis may be too large to be practical
- Average-case analysis is almost always much more difficult than worst-case or best analysis to compute

157

Exponential Growth?


5 Southern Accent NEWS Thursday, January 22, 2015

Enrollment Services implements new strategies

Overall enrollment at Southern declined last semester after years of exponential growth in the student body population.

Southern Accent

The student voice since 1926