Chapter 3

The Fundamentals: Algorithms

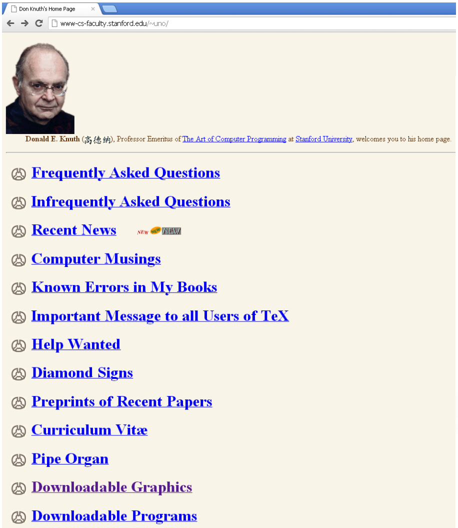
- This is a one-lecture introductory material on "Analysis of Algorithms".
- This introductory material is enough for a first course in "Data Structures" for sophomores.
- More "Analysis of Algorithms" is taught in "Algorithms" course.
- Good references for analysis of algorithms are:
  1. Rosen (2007), which we follow in this chapter.
  2. Knuth (1997), an indispensable resource.
  3. Mahmoud (1992, 2000), for probabilistic analysis of sorting and random search trees.

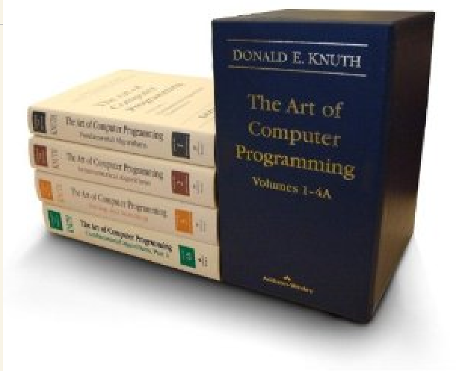
al-Khwārizmi, in full Muḥammad ibn Mūsā al-Khwārizmi (born c. 780, Baghdad, Iraq—died c. 850), Muslim mathematician and astronomer whose major works introduced Hindu-Arabic numerals and the concepts of algebra into European mathematics.

al-Khwārizmi's work on elementary algebra, al-Kitāb al-mukhtaṣar fi ḥisab al-jabr wa Í-muqābala ("The Compendious Book on Calculation by Completion and Balancing"), was translated into Latin in the 12th century, from which the title and term *Algebra* derives

... a second work by al-Khwārizmi introduced Hindu-Arabic numerals and their arithmetic to the West. It is preserved only in a Latin translation, *Algoritmi de numero Indorum* ("Al-Khwārizmi Concerning the Hindu Art of Reckoning"). From the name of the author, rendered in Latin as *algoritmi*, originated the term *algorithm*.[1]

[1] "al-Khwarizmi". Encyclopedia Britannica Online. Retrieved 30 October, 2012, from http://www.britannica.com/EBchecked/topic/317171/al-Khwarizmi





Donald Ervin Knuth, (born Jan. 10, 1938, Milwaukee, Wis., U.S.), American mathematician and computer scientist.

A pioneer in computer science, he took time out during the 1970s from writing his highly acclaimed multi-volume *The Art of Computer Programming* in order to develop TeX, a document-preparation system.[2]

[2]"Donald Ervin Knuth". Encyclopedia Britannica Online. Retrieved 30 October, 2012, from `http://www.britannica.com/EBchecked/topic/1380467/Donald-Ervin-Knuth`

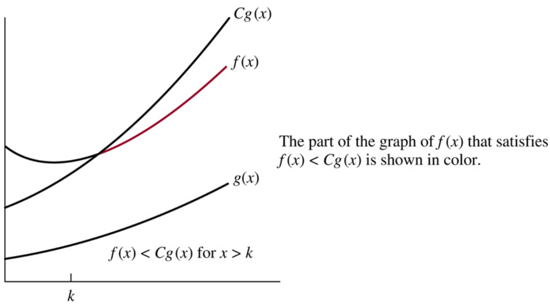
3.2 The Growth of Functions

- Used extensively to analyze algorithms.
- How algorithms behave with large size input?
- Algorithm A takes $100n^2 + 17n + 4$ steps; Algorithm B takes n^3; which is faster?
- Big-O (big-oh) notation.
- Big-O was used by mathematicians Edmund Landau and Paul Bachmann then introduced to Computer Science by Knuth.

Big-O Notation

Definition 1 *Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $O\left(g(x)\right)$ or $f(x) = O\left(g(x)\right)$ (read as: "$f(x)$ is big-oh of $g(x)$") if there are constants C and k such that*

$$\left|f(x)\right| \leq C\left|g(x)\right|, \ \forall x > k.$$
$$\frac{\left|f(x)\right|}{\left|g(x)\right|} \leq C, \ \forall x > k.$$



You can think of $f(x)$ and $g(x)$ as the number of steps of two different sorting algorithms and x is the number of elements.

Example 2 *Show that* $f(x) = x^2 + 2x + 1$ *is* $O\left(x^2\right)$.

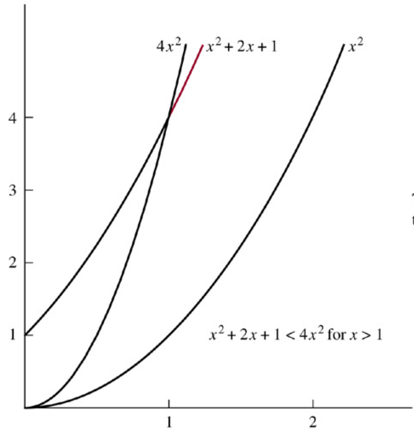
$$0 \le x^2 + 2x + 1 \le x^2 + 2x^2 + x^2 \qquad (\forall x \ge 1)$$
$$= 4x^2.$$

Then, $k = 1$ *and* $C = 4$.

Notice, it is also true that

$$x^2 < x^2 + 2x + 1, \ \forall x > 0.$$

Then, $x^2 = O\left(x^2 + 2x + 1\right)$; which said to have same order (coming later).



The part of the graph of $f(x) = x^2 + 2x + 1$ that satisfies $f(x) < 4x^2$ is shown in color.

Corollary 3 *If $f(x) = O\left(g(x)\right)$ and $h(x) > g(x)$ for sufficiently large x', it follows that $f(x) = O(h(x))$*

Proof. Since $f(x) = O\left(g(x)\right)$, then there exist C and k such that

$$
\begin{aligned}
\left|f(x)\right| &\leq C\left|g(x)\right|, \ \forall x > k, \\
&\leq C\left|h(x)\right|, \ \forall x > \max\left(k, x'\right) \\
&= C\left|h(x)\right|, \ \forall x > k' \\
f(x) &= O(h(x)).
\end{aligned}
$$

■

Notes:

- $x^2 + 2x + 1 = O\left(x^2\right)$; also it is $O\left(x^3\right)$.
- All functions used in the sequel are positive; hence we can drop $|\cdot|$.
- Shortly, x and $f(x)$ will denote problem size and algorithm complexity.

Example 4 *Show that n^2 is not $O(n)$.*

Suppose that $n^2 = O(n)$, then there exist n' and C such that

$$n^2 < Cn, \ \forall n > n',$$

which means

$$n < C\ , \ \forall n > n',$$

which is a contradiction.

Some Important Big-O Results

Theorem 5 *Let $f(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$, where a_i are real numbers. Then $f(x) = O(x^n)$.*

Proof. We consider $x > 1$, then

$$\begin{aligned}
\left|f(x)\right| &= \left|a_0 + a_1 x + \cdots + a_n x^n\right| \\
&\le |a_0| + |a_1 x| + \cdots + \left|a_n x^n\right| \quad \text{(triangle inequality)} \\
&= |a_0| + |a_1|\, x + \cdots + |a_n|\, x^n \\
&= x^n \left(|a_0| / x^n + |a_1| / x^{n-1} + \cdots + |a_n|\right) \\
&\le x^n \left(|a_0| + |a_1| + \cdots + |a_n|\right) \\
&= C x^n.
\end{aligned}$$

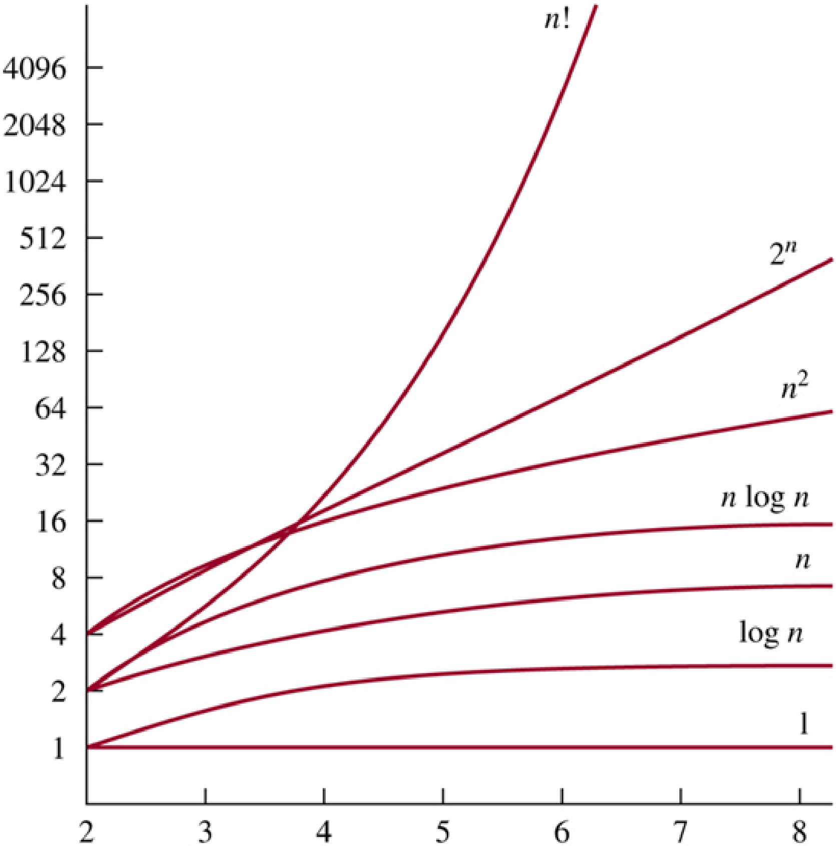
■

Example 6 *Big-O can be used, even, to approximate mathematical expressions.*

$$\begin{aligned}
S_n &= 1 + 2 + \cdots + n \\
&\le n + n + \cdots + n \\
&= n^2 \\
S_n &= O\left(n^2\right).
\end{aligned}$$

However, no guarantee that this is the lower bound!

Let's proof the following trends:



Notice that this is a log-scale.

We know that

$$\binom{n}{0}+\binom{n}{1}+\cdots+\binom{n}{n}=(1+1)^n=2^n$$

$$n<2^n$$

$$\log n<n \qquad \text{(base 2)}$$

Particularly, for $n>2$

$$1<\log n<n$$

$$n<n\log n<n^2$$

For $n\geq 4$,

$$2\cdot 2\cdots 2<1\cdot 2\cdots n<n\cdot n\cdots n$$

$$2^n<n!<n^n$$

$$n<\log n!<n\log n.$$

All the above are O relationships with $C=1$. What is very interesting is that we can show that (at the end) $n\log n\leq 2\log n!$, as well. Therefore, they are of the same order:

$$\log n!=O\left(n\log n\right),$$

$$n\log n=O\left(\log n!\right).$$

Proof of $n \log n = O\left(\log n!\right)$ **.**

$$
\begin{aligned}
n! &= n \cdot (n-1) \cdots \cdot 1 \\
&= 1 \cdot 2 \cdot \cdots \cdot n \\
(n!)^2 &= (n \cdot 1)\left((n-1) \cdot 2\right) \cdots (1 \cdot n) \\
&= \prod_{i=0}^{n-1} (n-i)(i+1) \, . \\
&\geq \prod_{i=0}^{n-1} n && \text{(easy to show)} \\
&= n^n \\
2 \log n! &\geq n \log n.
\end{aligned}
$$

$$
\begin{aligned}
& n \leq (n-i)(i+1) && \equiv \\
& n \leq ni + n - i^2 - i && \equiv \\
& 0 \leq (n-i-1)\, i && \Longrightarrow \\
& 0 \leq i \leq n-1.
\end{aligned}
$$

$\blacksquare$

Proof of $n^2 < 2^n$. base: for $n = 5,\ 5^2 < 2^5$.
induction: Suppose the statement is true for some $n \geq 5$; i.e.,

$$\begin{aligned} n^2 &< 2^n, \\ (n+1)^2 &= n^2 + 2n + 1 \\ &< n^2 + n^2 \qquad\qquad (n^2 > 2n + 1\ \forall n \geq 3) \\ &= 2n^2 \\ &< 2 \cdot 2^n \\ &= 2^{(n+1)}. \end{aligned}$$

Hence, it is true for all $n \geq 5$. ■
A more general proof for n^m is given next:

Proof of $n^m < 2^n$. We show that $\forall m, \exists n_0(m)$ such that $n^m < 2^n \; \forall n > n_0$. Let's see Mathematica Notebook.
base: We need to get a base value n_0, such that $n_0^m < 2^{n_0}$. Take $n_0 = 2^m$; indeed

$$\left(2^m\right)^m = 2^{m^2} < 2^{2^m},$$

if $m^2 < 2^m$; which was proven for $m \geq 5$.
induction: Suppose it is true for **some** $n \geq n_0$; i.e.,

$$\begin{aligned}
n^m &< 2^n, \; n_0 \leq n. \\
(n+1)^m &= n^m + \binom{m}{1} n^{m-1} + \cdots + \binom{m}{m} \\
&= n^m + n^{m-1} \left(\binom{m}{1} + \binom{m}{2} / n \cdots + \binom{m}{m} / n^{m-1} \right) \\
&< n^m + n^{m-1} \left(\binom{m}{0} + \binom{m}{1} + \binom{m}{2} \cdots + \binom{m}{m} \right) \\
&= n^m + n^{m-1} 2^m \\
&\leq n^m + n^{m-1} n && (2^m = n_0 \leq n \text{ is used}) \\
&= 2n^m \\
&< 2 \cdot 2^n \\
&= 2^{n+1}.
\end{aligned}$$

Hence, for $m \geq 5$, we proved that $n^m < 2^n \; \forall n \geq 2^m$; and since, $\forall n > 1$, $n^{m_1} < n^{m_2}$ when $m_1 < m_2$, then the statement is true for every m. ■

The Growth of Combinations of Functions

Theorem 7 *Suppose* $f_1(x) = O\left(g_1(x)\right)$ *and* $f_2(x) = O\left(g_2(x)\right)$*. Then* $\left(f_1 + f_2\right)(x) = O\left(\max\left(\left|g_1(x)\right|, \left|g_2(x)\right|\right)\right)$.

Proof.

$$\begin{aligned}
\left|f_1(x)\right| &\leq C_1\left|g_1(x)\right|,\ \forall x > k_1\\
\left|f_2(x)\right| &\leq C_2\left|g_2(x)\right|,\ \forall x > k_2\\
\left|f_1(x) + f_2(x)\right| &\leq \left|f_1(x)\right| + \left|f_2(x)\right| && \text{(triangle inequality)}\\
&\leq C_1\left|g_1(x)\right| + C_2\left|g_2(x)\right|,\ \forall x > \max(k_1, k_2).\\
&\leq (C_1 + C_2)\max\left(\left|g_1(x)\right|, \left|g_2(x)\right|\right),\ \forall x > k\\
&= C\max\left(\left|g_1(x)\right|, \left|g_2(x)\right|\right),\ \forall x > k.
\end{aligned}$$

■

Theorem 8 *Suppose that* $f_1(x) = O\left(g_1(x)\right)$ *and* $f_2(x) = O\left(g_2(x)\right)$*. Then* $\left(f_1 f_2\right)(x) = O\left(g_1(x)\, g_2(x)\right)$.

Proof.

$$\begin{aligned}
\left|f_1(x)\right| &\leq C_1\left|g_1(x)\right|,\ \forall x > k_1\\
\left|f_2(x)\right| &\leq C_2\left|g_2(x)\right|,\ \forall x > k_2\\
\left|f_1(x)\, f_2(x)\right| &= \left|f_1(x)\right|\left|f_2(x)\right|\\
&\leq C_1\left|g_1(x)\right| C_2\left|g_2(x)\right|,\ \forall x > \max(k_1, k_2)\\
&= C\left|g_1(x)\, g_2(x)\right|,\ \forall x > k.
\end{aligned}$$

■

Motivation:

- E.g., One program is composed of two sequential algorithms.
- Estimates using O, frequently is easier than exact calculations.

Example 9 *A program has two sequential pieces of code, one is* $3n\log(n!)$ *steps followed by* $(n^2+3)\log n$ *steps. Approximately, what is the order of number of steps of the whole program.?*

$$\begin{aligned}
f(n) &= \underline{3n}\log(n!) + \underline{\left(n^2+3\right)}\log n && (\text{“=”} \equiv \text{“equal”})\\
&= \underline{O(n)}O(n\log n) + \underline{O\left(n^2\right)}O(\log n) && (\text{“=”} \equiv \text{“is”})\\
&= O\left(n^2\log n\right) + O\left(n^2\log n\right)\\
&= O\left(n^2\log n\right).
\end{aligned}$$

Example 10

$$\begin{aligned}
f(x) &= (x+1)\log\left(x^2+1\right) + 3x^2\\
&\le (x+1)\log\left(2x^2\right) + 3x^2 && (\forall x \ge 1)\\
&= (x+1)\left(\log 2 + 2\log x\right) + 3x^2\\
&= O(x\log x) + O\left(x^2\right)\\
&= O\left(x^2\right).
\end{aligned}$$

Big-Ω (Big-Omega) and Big-Θ (Big-Theta) Notation

- If O gives an upper bound, Ω gives a lower bound, and Θ gives both.
- This is a remedy for abusing O; e.g., $x = O\left(x^{10}\right)$ of course :)

Definition 11 *Let f and g be functions from $\mathbb{N} \to \mathbb{R}$ or $\mathbb{R} \to \mathbb{R}$. We say that $f(x) = \Omega\left(g(x)\right)$ if $\exists C, k$ such that*

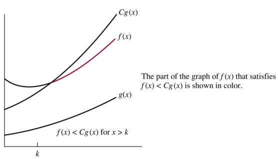
$$\left|f(x)\right| \geq C\left|g(x)\right|, \ \forall x > k.$$

Corollary 12 *Let f and g be functions from $\mathbb{N} \to \mathbb{R}$ or $\mathbb{R} \to \mathbb{R}$. Then, $f(x) = \Omega\left(g(x)\right)$ if and only if $g(x) = O\left(f(x)\right)$.*

Proof. The proof is trivial

$$\left|f(x)\right| \geq C\left|g(x)\right|, \ \forall x > k \iff \left|g(x)\right| \leq \frac{1}{C}\left|f(x)\right|, \ \forall x > k$$
$$f(x) = \Omega\left(g(x)\right) \iff g(x) = O\left(f(x)\right)$$

■



Definition 13 *Let f and g be functions from $\mathbb{N} \to \mathbb{R}$ or $\mathbb{R} \to \mathbb{R}$. We say that $f(x) = \Theta(g(x))$ (or, $f(x)$ is of* order *$g(x)$) if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.*

Notice that (using the corollary above)**:**

$$
\begin{aligned}
f(x) = \Theta(g(x)) &\equiv f(x) = O(g(x)) \;\&\; f(x) = \Omega(g(x)) \\
&\iff g(x) = \Omega(f(x)) \;\&\; g(x) = O(f(x)) \\
&\equiv g(x) = \Theta(f(x))
\end{aligned}
$$

$$
\begin{aligned}
f(x) = \Theta(g(x)) &\equiv f(x) = O(g(x)) \;\&\; f(x) = \Omega(g(x)) \\
&\iff f(x) = O(g(x)) \;\&\; g(x) = O(f(x))
\end{aligned}
$$

Example 14 *We showed above that* $S_n\,(=1+2+\cdots+n) = O\left(n^2\right)$. *Is this bound exaggerated? What is its lower bound? What is its exact order?*

$$\begin{aligned}
S_n &\geq \lceil n/2 \rceil + (\lceil n/2 \rceil + 1) + \cdots + n \\
&\geq \lceil n/2 \rceil + \lceil n/2 \rceil + \cdots + \lceil n/2 \rceil \\
&= (n - \lceil n/2 \rceil + 1)\lceil n/2 \rceil \\
&\geq (n/2)(n/2) \\
&= n^2/4 \\
&= \Omega\left(n^2\right).
\end{aligned}$$

This proves that $S_n = \Theta\left(n^2\right)$. *Compare to the exact formula:*

$$S_n = \frac{1}{2} n (n+1).$$

Theorem 15 *Suppose* $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$*, i.e., a polynomial of degree n, where* $a_i \neq 0$*. Then*

$$f(x) = \Theta\left(x^n\right).$$

Proof. We proved $f(x) = O(x^n)$. Let's show $f(x) = \Omega(x)$.

$$\left|f(x)\right| = \left|\sum_{i=1}^{n} a_i x^i\right|,$$

OLOG, $a_n > 0$; otherwise $\left|-\sum_{i=1}^{n} -a_i x^i\right| = \left|\sum_{i=1}^{n} -a_i x^i\right|$.

$$\begin{aligned}\left|f(x)\right| &= \left|\sum_{a_i>0} a_i x^i - \sum_{a_i<0} |a_i| x^i\right| \\ &\geq \sum_{a_i>0} a_i x^i - \sum_{a_i<0} |a_i| x^i \\ &\geq a_n x^n - \sum_{a_i<0} |a_i| x^{n-1} && (\forall x > 1) \\ &= a_n x^n - s x^{n-1}.\end{aligned}$$

We can say $a_n x^n - s x^{n-1} > c x^n$ iff

$$\begin{aligned}(a_n - c) x^n &> s x^{n-1} \\ x &> \frac{s}{a_n - c} && (c < a_n)\end{aligned}$$

$$\begin{aligned}&\left|f(x)\right| > c x^n \quad \forall 0 < c < a_n,\ x > \frac{s}{a_n - c} \\ &\left|f(x)\right| = \Omega\left(x^n\right).\end{aligned}$$

■

Example 16 *Show that $H_n = O(\log n)$, where H_n is the n^{th} harmonic number*

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}.$$

By drawing the curve $y = 1/x,\ x \geq 1$, we observe that

$$\sum_{j=2}^{n} \frac{1}{j} < \int_1^n \frac{1}{x} dx.$$

$$\begin{aligned} H_n &= 1 + \sum_{j=2}^{n} \frac{1}{j} \\ &< 1 + \int_1^n \frac{1}{x} dx \\ &= 1 + \ln n \\ &< 2 \ln n = \left(\frac{2}{\log e}\right) \log n \qquad \text{(ln for base } e\text{.)} \\ H_n &= O(\log n). \end{aligned}$$

3.3 Complexity of Algorithms (Analysis of Algorithms)

- Time complexity: time required for solving the problem.
- Space complexity: memory required for solving the problem.

Let's stick now to the Analysis of Algorithms concerning Time Complexity.

Time Complexity

Definition 17 (Time Complexity) *of an algorithm is expressed in terms of the number of* basic *operations used by the algorithm when the input has a particular* size.

Very Important Comments:

- We defined *Time Complexity* in number of steps, not absolute time, not to be machine dependent.
- Not all operations are *basic*; e.g.,
  - `x = MatrixMultiplcation(A, B);`
  - `x = 0;`
- Not all basic operations take same execution time.
  - `x = 3 * 4;`
  - `x = 3 + 4;`
- Therefore, we have to **define step(s)**.

General Framework of Analysis of Algorithms:

- The best is to obtain an exact expression for complexity T (number of steps) as a function of n (problem size): $T = T(n)$.
- $T = \Theta\big(f(n)\big)$ or $T = O\big(f(n)\big)$.
- Sometimes T is a r.v. (not deterministic)
  - Worst-case complexity: $\max(T)$.
  - Best-case complexity: $\min(T)$.
  - Average-case complexity: $\mathrm{E}[T]$

Example 18 (a max. of a list) :

```
xmax = list[0];
for(i=1; i<n; i++)
    if (list[i]>xmax)
        xmax = list[i];
```

What is the step(s)?

- *If a step is **any** comparison: then we have* $2(n-1)$ *for both i<n, list[i]>xmax, and another* 1 *for i<n when exiting the loop.*

$$\begin{aligned} T_1 &= 2(n-1)+1 = 2n-1. \\ &= \Theta(n). \end{aligned}$$

- *If a step is an **element** comparison*

$$\begin{aligned} T_2 &= n-1 \\ &= \Theta(n). \end{aligned}$$

For both definitions, the worst-, best-, and average-case are the same since the algorithm is deterministic. All are $\Theta(n)$.

Example 19 (linear search) :

```
for (i=1; i<=n; i++)
    if (x == list[i])
        return i;
```

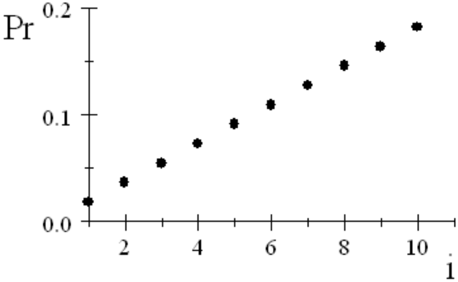
What is the step(s)?

- *a step is an **element** comparison*

$$
\begin{aligned}
T &= i,\ 1 \le i \le n. \\
&\le n \\
&= O(n) \\
\min T &= 1 = \Theta(1). \\
\max T &= n = \Theta(n). \\
\mathrm{E}[T] &= \sum_{i=1}^{n} i \Pr(i) \\
&= \sum_{i=1}^{n} i \frac{1}{n} \qquad \text{(only for } i \sim Uniform(1,n)\text{)} \\
&= \frac{1}{n} \sum_{i=1}^{n} i \\
&= \frac{1}{n} \frac{1}{2} n(n+1) \\
&= \frac{1}{2}(n+1) = \Theta(n).
\end{aligned}
$$

What about if the element exists with, e.g., a linear pmf

$$\Pr(i) = \frac{2}{n(n+1)} i, \ 1 \le i \le n.$$



$$\begin{aligned}
\mathrm{E}[T] &= \sum_{i=1}^{n} i \Pr(i) \\
&= \sum_{i=1}^{n} i \frac{2}{n(n+1)} i \\
&= \frac{2}{3} n + \frac{1}{3} && \text{(homework)} \\
&> \frac{1}{2}(n+1) \ \forall n > 1.
\end{aligned}$$

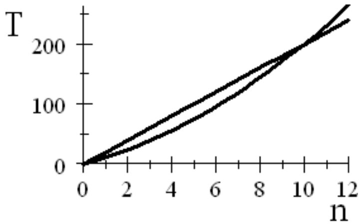
Notice: This is a field by itself: “Probabilistic Analysis of Algorithms”

Understanding the Complexity of Algorithms

- To compare two algorithms you must use the same *step* definition.

- Absolute vs. asymptotic complexity; e.g., which is better, A1 and A2:

$$T_1(n) = n^2 + 10n \qquad = \Theta\left(n^2\right)$$
$$T_2(n) = 20n \qquad = \Theta(n).$$



- The following algorithm is the best:

```
If n<n0 then A1 else A2
```

- Do not Use O where it is Θ

- If $\max(T) = O\left(n^b\right)$, the problem is called **tractable.**

- However, $\max(T) = O\left(n^{100}\right)$, will take unrealistic time

- If $\max(T) \neq O\left(n^b\right)$, the problem is called **intractable.**

TABLE 1 Commonly Used Terminology for the Complexity of Algorithms.

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	$n \log n$ complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

TABLE 2 The Computer Time Used by Algorithms.

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-9} s	10^{-8} s	3×10^{-8} s	10^{-7} s	10^{-6} s	3×10^{-3} s
10^2	7×10^{-9} s	10^{-7} s	7×10^{-7} s	10^{-5} s	4×10^{13} yr	*
10^3	$1(0 \times 10^{-8}$ s	10^{-6} s	1×10^{-5} s	10^{-3} s	*	*
10^4	$1(3 \times 10^{-8}$ s	10^{-5} s	1×10^{-4} s	10^{-1} s	*	*
10^5	$1(7 \times 10^{-8}$ s	10^{-4} s	2×10^{-3} s	10 s	*	*
10^6	2×10^{-8} s	10^{-3} s	2×10^{-2} s	17 min	*	*

Bibliography

Knuth, D.E., 1997. The art of computer programming. 3rd ed., Addison-Wesley, Reading, Mass.

Mahmoud, H.M., 1992. Evolution of random search trees. Wiley, New York.

Mahmoud, H.M., 2000. Sorting : a distribution theory. John Wiley \& Sons, New York.

Rosen, K.H., 2007. Discrete mathematics and its applications. 6th ed., McGraw-Hill Higher Education, Boston.