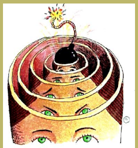


Lecture 26:
Algorithms with Logarithmic Complexity

Aims:

- To discuss logarithms;
- To look at an algorithm with logarithmic complexity.

Module Home Page
Title Page
Page 2 of 11
Back
Full Screen
Close
Quit

26.1. Logarithmic Functions

- Suppose you are told the following:

$$3^c = 81$$

  What is c? To what power must you raise 3 to get 81?

- Logarithmic functions are the inverse of exponential functions. If a is b to the power c, i.e. $b^c = a$, we also say that c is the logarithm of a to the base b (meaning c is the power to which we have to raise b in order to get a), and we write $\log_b a = c$.

- For example,

$$\begin{aligned} \log_{10} 100 &= 2 \quad (\text{since } 10^2 = 100) \\ \log_2 8 &= 3 \quad (\text{since } 2^3 = 8) \end{aligned}$$

- Note that $\log_b a$ is defined only when a is a positive real number and b is a positive real number other than 1. (Note: a cannot be 0; b cannot be 0 or 1.) We'll only be dealing with positive integer values for a and integers > 1 for b anyway. And mostly, for us, b will be 2.

- Some laws:

$$\begin{aligned} \log_b 1 &= 0 \\ \log_b b &= 1 \\ \log_b cd &= \log_b c + \log_b d \\ \log_b c/d &= \log_b c - \log_b d \\ \log_b a^c &= c \log_b a \\ \log_b a &= (\log_c a)/\log_c b \\ b^{\log_c a} &= a^{\log_c b} \end{aligned}$$

- E.g. simplify $\log_2 2^n$
- Outside Computer Science, it is common to compute logs to the base 10 and to compute so-called natural logs, which are logs to the base e (where $e = 2.71828\ldots$). But, in Computer Science logs to the base 2 are the most common.
- If your calculator has a button labelled *log* on it, then this almost certainly computes logs to the base 10. If your calculator has a button labelled *ln*, then this computes natural logs.
- You may then be wondering: if my calculator only offers logs to the base 10 and natural logs, how do I use my calculator to compute logs to the base 2? Well, you can use this law:

$$\log_b a = \frac{\log_c a}{\log_c b}$$

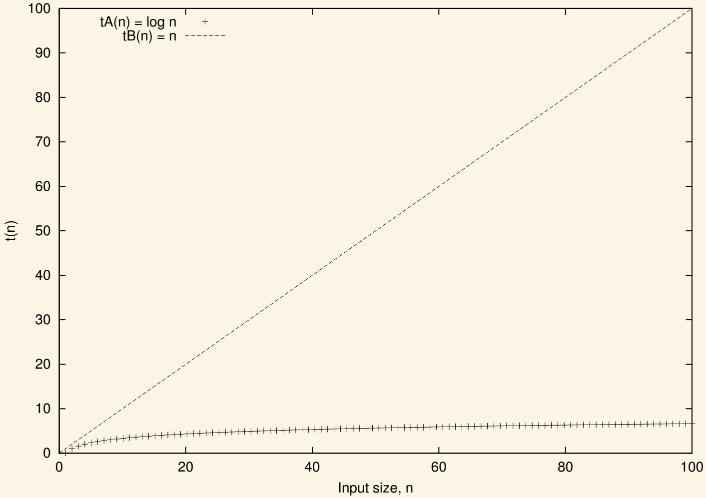
  This law allows us to change base. In particular,

$$\log_2 a = \frac{\ln a}{\ln 2}$$

- E.g. what is $\log_2 12$?
- What are logs used for?
  - They help you solve equations that involve exponentiation.
  - They reduce multiplication/division of large numbers to addition/subtraction (log tables & slide rules!)
  - In complexity theory, they are used to measure input sizes, especially when the input is numeric and we want to count the number of digits.

- In complexity theory, the complexity functions for algorithms that repeatedly split their input into two halves involve logs to the base 2.
- Logarithmic scale helps us to fit plots onto graph paper.
- They are used in the Richter scale for measuring the seismic energy released by earthquakes!

- Suppose algorithm A's worst-case time complexity $t_A(n) =_{\text{def}} \log n$ and algorithm B's worst-case time complexity $t_B(n) =_{\text{def}} n$. $\log_n$ grows much more slowly than n.



Module Home Page
Title Page
◀◀ ▶▶
◀ ▶

Back
Full Screen
Close
Quit

26.2. Binary Search

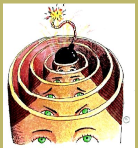
- Here is the binary search algorithm given in a previous lecture. Remember it assumes that the contents of a are stored in ascending order. Note also that here we have assumed that a contains distinct integers (no duplicates). This simplifies the analysis. But the algorithm works just as well when duplicates are allowed.

Algorithm: BINARYSEARCH$(x, a, lower, upper)$

Parameters: x is an integer; $a[lower \ldots upper]$ is an array of distinct integers stored in non-decreasing order; $0 < lower \leq upper$.

Returns: The position of x in a if found, otherwise **fail**.

{ $lo := lower$;
    $hi := higher$;
    while $lo \leq hi$
    { $mid := (lo + hi) \operatorname{div} 2$;
        if $a[mid] < x$
        { $lo := mid + 1$;
        }
        else if $a[mid] = x$
        { **return** mid;
        }
        else
        { $hi := mid - 1$;
        }
    }
    return fail;
}



Logarithmic Functions
Binary Search
Summary Graph

Module Home Page
Title Page
Page 6 of 11
Back
Full Screen
Close
Quit

- To carry out our analysis, let's make some assumptions:
  - The algorithm performs comparisons, some arithmetic and some assignments. We count only *element comparisons*, i.e. comparisons between x and the elements in a. The frequency of the other operations would be similar to that of the element comparisons.
  - What we want to do in this algorithm is carry out a three-way comparison. We want to find out whether $a[mid]$ is less than, equal to, or greater than x. But DECAFF, along with most programming languages, forces us to implement this as two two-way comparisons. For simplicity, in our frequency counts we will assume that only a single operation is needed to determine which of the three possibilities holds.
- Assume that a is an array of length 15 and that it is indexed from 1 to 15 (rather than 0 to 14 as it would be in Java). For concreteness, let's say that these are the values that a contains:

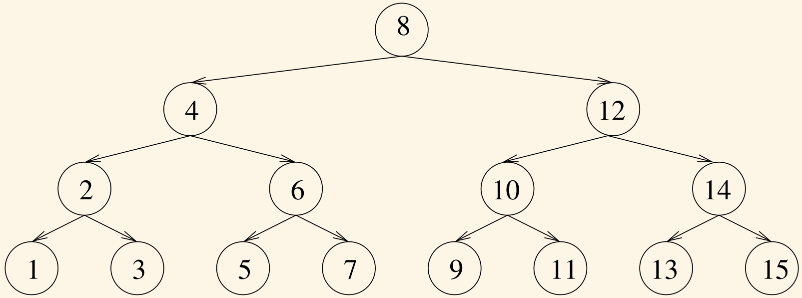
100	110	120	130	140	150	160	170	180	190	200	210	220	230	240
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

  In the lecture, we will search for
  - $x = 170$
  - $x = 150$
  - $x = 180$
  - $x = 185$

Logarithmic Functions
Binary Search
Summary Graph

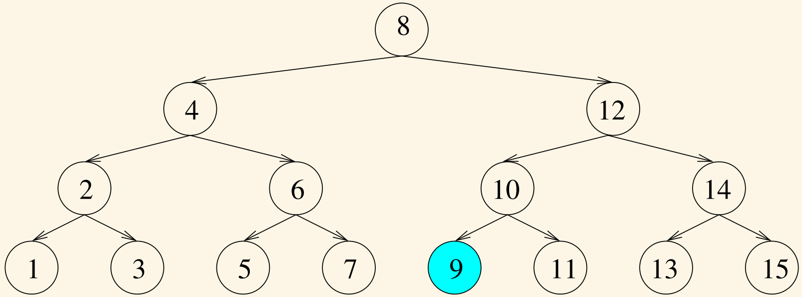
Module Home Page
Title Page
Page 7 of 11
Back
Full Screen
Close
Quit

100	110	120	130	140	150	160	170	180	190	200	210	220	230	240
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



- Searching for $x = 185$:

100	110	120	130	140	150	160	170	180	190	200	210	220	230	240
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



FAIL

– Initially, the number of candidates is n.
– If you have executed the loop body once, the number of candidates is at most $n \operatorname{div} 2$.
– If you have executed the loop body twice, the number of candidates is at most $n \operatorname{div} 4$.
– In general, if you have executed the loop body k times, the number of candidates is at most $n \operatorname{div} 2^k$.
– The worst case is unsuccessful search where we reduce the candidates to 1 and then do one more test. We have reduced the candidates to 1 when

$$1 = n \operatorname{div} 2^k$$

i.e.

$$k = \log_2 n$$

Module Home Page
Title Page
Page 9 of 11
Back
Full Screen
Close
Quit

- So, performing one more test, we get

$$t(n) =_{\text{def}} 1 + \log_2 n$$

  (For those of you who can handle a bit more precision, it is actually $1 + \lfloor \log_2 n \rfloor$.)
- E.g. with $n = 15$, $k = 4$

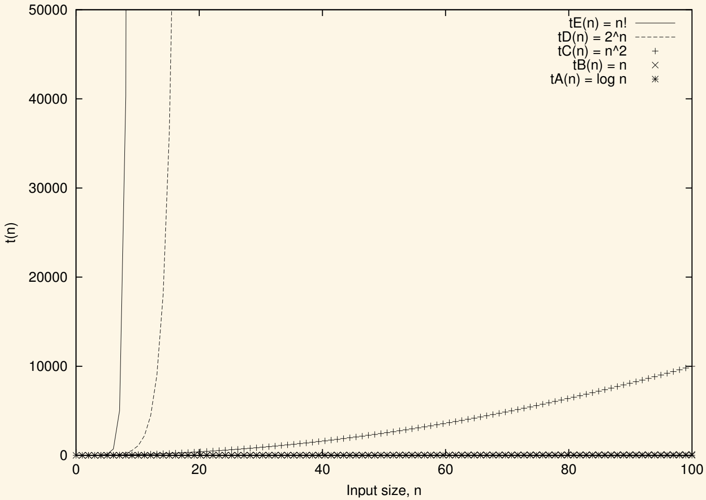
- Closing remarks:
  - Many algorithms involve repeatedly splitting a list or array into equal halves and then turning attention to one or both halves.
  - Base 2 logarithms tell us how many times we can split into halves.
  - Base 2 logarithms are therefore crucial in complexity analysis.
  - They are so useful that, henceforth, if we write $\log n$ we mean $\log_2 n$.

Logarithmic Functions
Binary Search
Summary Graph

Module Home Page
Title Page
Page 10 of 11
Back
Full Screen
Close
Quit

26.3. Summary Graph

- Let's put several plots on the same graph: something logarithmic, linear, quadratic, exponential and factorial.



Acknowledgements

I based my explanation of the behaviour and complexity of binary search on the treatments given in [GT02] and [HSR96].

Clip Art (of head with bomb) licensed from the Clip Art Gallery on DiscoverySchool.com.

References

[GT02] M. T. Goodrich and R. Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples*. Wiley, 2002.

[HSR96] E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms/C++*. W.H. Freeman, 1996.