Topic 1.1

Big-O notation

How much resource does an algorithm need?

There can be many algorithms to solve a problem.

Some are good and some are bad.

Good algorithms are efficient in

- time and
- space.

Our method of measuring time is cumbersome and machine-dependent.

We need approximate counting that is machine-independent.

Commentary: Sometimes there is a trade-off between time and space. For example, the inefficient linear search only needed one extra integer, but the binary search used three extra integers. The difference between two integers may be a minor issue, but it illustrates the trade-off.

Input size

An algorithm may have different running times for different inputs.

How do we think about comparing algorithms?

We define the rough size of the input, usually in terms of important parameters of input.

Example 1.8

In the problem of search, we say that the number of elements in the array is the input size.

Please note that the size of individual elements is not considered. *(Why?)*

Commentary: Ideally, the number of bits in the binary representation of the input is the size, which is too detailed and cumbersome to handle. In the case of search, we assume that elements are drawn from the space of size 2^{32} and can be represented using 32 bits. Therefore, the type of the element was `int`.

Best/Average/Worst case

For a given size of inputs, we may further make the following distinction.

1. Best case: Shortest running time for some input.
2. Worst case: Worst running time for some input.
3. Average case: Average running time on all the inputs of the given size.

Exercise 1.7

How can we modify almost any algorithm to have a good best-case running time?

Example: Best/Average/Worst case

Example 1.9

```
int BinarySearch(int* S, int n, int e){
  // S is a sorted array
  int first = 0, last = n;
  int mid = (first + last) / 2;
  while (first < last) {
    if (S[mid] == e) return mid;
    if (S[mid] > e) {
      last = mid;
    } else {
      first = mid + 1;
    }
    mid = (first + last) / 2;
  }
  return -1;
}
```

In BinarySearch, let $n = 2^{k-1}$.

1. Best case: $e == S[n/2]$
   $T_{Read} + 6T_{Arith} + T_{return}$,
2. Worst case: $e \notin S$
   We have seen the worst case.
3. The average case is roughly equal to the worst case because most often the loop will iterate k times. (Why?)

Commentary: Analyzing the average case is usually involved. For some important algorithms, we will do a detailed average time analysis.

Asymptotic behavior

For short inputs, an algorithm may use a shortcut for better running time.

To avoid such false comparisons, we look at the behavior of the algorithms in limit.

Ignore hardware-specific details

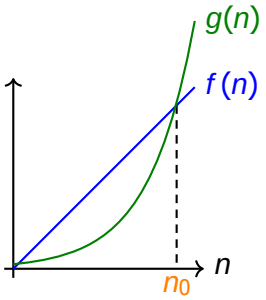
- Round numbers $100000000000001 \approx 100000000000000$
- Ignore coefficients $3kT_{Arith} \approx k$

Big-O notation: approximate measure

Definition 1.3

Let f and g be functions $\mathbb{N} \to \mathbb{N}$. We say $f(n) \in O(g(n))$ if there are c and n_0 such that

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



- In limit, $cg(n)$ will dominate $f(n)$
- We say $f(n)$ is $O(g(n))$

Exercise 1.8

Which of the following are the true statements?

- $5n + 8 \in O(n)$
- $5n + 8 \in O(n^2)$
- $5n^2 + 8 \in O(n)$
- $n^2 + n \in O(n^2)$
- $50000000000000000000000n^2 \in O(n^2)$
- $50n^2 \log n + 60n^2 \in O(n^2 \log n)$

Example: Big-O of the worst case of `BinarySearch`

Example 1.10

In `BinarySearch`, let $n = 2^{k-1}$.

1. Worst case: $e \notin S$

   $kT_{Read} + (6k + 5)T_{Arith} + (3k + 1)T_{jump} + T_{return} \in O(k)$

   Since $k = log\ n + 1$, therefore $k \in O(log\ n)$

   We may also say `BinarySearch` is $O(log\ n)$.

   Therefore, the worst-case running time of `BinarySearch` is $O(log\ n)$.

Exercise 1.9

Prove that $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.

What does Big O says?

Expresses the approximate number of operations executed by the program as a function of input size

Hierarchy of algorithms

- $O(log\ n)$ algorithm is better than $O(n)$
- We say $O(log\ n) < O(n) < O(n^2) < O(2^n)$

May hide large constants!!

Complexity of a problem

The complexity of a problem is the complexity of the best-known algorithm for the problem.

Exercise 1.10

What is the complexity of the following problem?

- *sorting an array* — $O(n^2)$ ✗
- *matrix multiplication* — $O(n^3)$ ✗

Best algorithm is still not known

Exercise 1.11

What is the best-known complexity for the above problems?

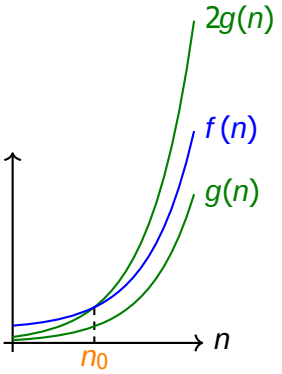
Commentary: A discussion on the latest developments in matrix multiplication algorithms. https://en.wikipedia.org/wiki/Computational_complexity_of_matrix_multiplication

Θ-Notation

Definition 1.4 (Tight bound)

Let f and g be functions $\mathbb{N} \to \mathbb{N}$. We say $f(n) \in \Theta(g(n))$ if there are c_1, c_2, and n_0 such that

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



There are more variations of the above definition. Please look at the end.

Exercise 1.12

a. Does the worst-case complexity of `BinarySearch` belong to $\Theta(\log n)$?

b. If yes, give c_1, c_2, and n_0 for the application of the above definition on `BinarySearch`.

Names of complexity classes

- Constant: $O(1)$
- Logarithmic: $O(logn)$
- Linear: $O(n)$
- Quadratic: $O(n^2)$
- Polynomial : $O(n^k)$ for some given k
- Exponential : $O(2^n)$