CSCI 241

Scott Wehrwein

Runtime Analysis:
Constant-time Operations

Goals

Know the motivations for using asymptotic runtime analysis.

Know how to identify constant time operations in simple algorithms.

Which algorithm is better?

Suppose you have two different algorithms that solve the same problem. For example, *search a sorted array*.

```
int linearSearch(int[] A, int x) {
  for (int i = 0; i < A.length; i++){
    if (A[i] == x) {
      return i;
    }
  }
  return -1;
}
```

```
int binarySearch(int[] A, int x) {
  int start = 0;
  int end = A.length;
  while (start < end) {
    int mid = (start + end) / 2;
    if (x == A[mid]) {
      return mid;
    if (x < A[mid]) {
      end = mid;
    } else {
      start = mid + 1;
    }
  }
  return -1;
}
```

A consequential question:

Which is better?

What *is* "better"?

How should we compare algorithms?

- Which one finishes faster?
- Which one uses less memory?
- Which one has more lines of code?
- Which one executes more lines of code?
- How many *operations* does each perform *as a function of the input data size*?

Properties of a good measurement system

- Explicitly depends on input size
- Doesn't sweat the details:
  - Doesn't depend on hardware specifics
  - Assigns same number to algorithms that are 'close enough'

What do we mean by "operations"?

A constant time (or primitive) operation is any operation whose runtime does not depend on the size of the input.

Here, size of the input is `A.length`:

```
int linearSearch(int[] A, int x) {
   for (int i = 0; i < A.length; i++){
      if (A[i] == x) {
         return i;
      }
   }
   return -1;
}
```

Examples:

- Read from memory
- Write to memory
- Evaluate arithmetic
- Return from a method

What do we mean by "operations"?

A primitive (or constant time) operation is any operation whose runtime does not depend on the size of the input.

Here, size of the input is `A.length`:

```
int linearSearch(int[] A, int x) {
   for (int i = 0; i < A.length; i++){
      if (A[i] == x) {
         return i;
      }
   }
   return -1;
}
```

Examples:

- Read from memory
- Write to memory
- Evaluate arithmetic
- Return from a method

What do we mean by "operations"?

A primitive (or constant time) operation is any operation whose runtime does not depend on the size of the input.

Here, size of the input is `A.length`:

```
int linearSearch(int[] A, int x) {
   for (int i = 0; i < A.length; i++){
      if (A[i] == x) {
         return i;
      }
   }
   return -1;
}
```

Examples:

- Read from memory
- Write to memory
- Evaluate arithmetic
- Return from a method

What do we mean by "operations"?

A primitive (or constant time) operation is any operation whose runtime does not depend on the size of the input.

Here, size of the input is `A.length`:

```
int linearSearch(int[] A, int x) {
   for (int i = 0; i < A.length; i++){
      if (A[i] == x) {
         return i;
      }
   }
   return -1;
}
```

Examples:

- Read from memory
- Write to memory
- Evaluate arithmetic
- Return from a method

What do we mean by "operations"?

Key insight: a fixed number of primitive operations is itself a primitive operation.

Example:

```
i++
```

is shorthand for

```
i = i + 1
```

...none of this depends on the input size!

How should we compare algorithms?

- Which one finishes faster?
- Which one uses less memory?
- Which one has more lines of code?
- Which one executes more lines of code?
- How many *operations* does each perform *as a function of the input data size*?

constant-time
^