

CSE 332: Data Structures and Parallelism

Spring 2022
 Richard Anderson
 Lecture 2: Stacks, Queues and Algorithm Analysis

Announcements

- Section on Thursday
- Project #1: Partner request by 6 pm today
 - Pair programming: work together!
- Exercise #1: Due Friday
 - Computer set up and some programming
- Reading, Weiss
 - Algorithm Analysis: 2.1-2.4
 - Stacks and Queues: 3.1-3.7

First Example: Queue ADT

- FIFO: First In First Out
- Queue operations
 - create
 - enqueue
 - dequeue
 - is_empty

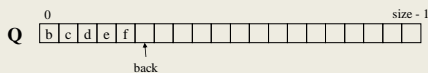


Queues in practice

- Print jobs
- File serving
- Phone calls and operators

(Later, we will consider “priority queues.”)

Array Queue Data Structure



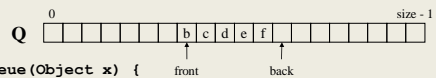
```
enqueue(Object x) {
    Q[back] = x
    back = back + 1
}
```

What's missing in these functions?

```
dequeue() {
    x = Q[0]
    shiftLeftOne()
    back = back - 1
    return x
}
```

How to find K-th element in the queue?

Circular Array Queue Data Structure



```
enqueue(Object x) {
    assert(!is_full())
    Q[back] = x
    back = (back + 1) % Q.size
}
```

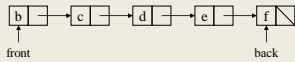
How test for empty/full list?

```
dequeue() {
    assert(!is_empty())
    x = Q[front]
    front = (front + 1) % Q.size
    return x
}
```

How to find K-th element in the queue?

What to do when full?

Linked List Queue Data Structure



```

void enqueue(Object x) {
    if (is_empty())
        front = back = new Node(x)
    else {
        back.next = new Node(x)
        back = back.next
    }
}

bool is_empty() {
    return front == null
}

Object dequeue() {
    assert(!is_empty())
    return_data = front.data
    temp = front
    front = front.next
    delete temp
    return return_data
}
    
```

3/30/2022

CSE 332

7

Circular Array vs. Linked List

- Advantages of circular array?
- Advantages of linked list?

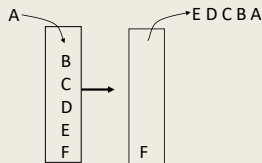
3/30/2022

CSE 332

8

Second Example: Stack ADT

- LIFO: Last In First Out
- Stack operations
 - create
 - push
 - pop
 - top
 - is_empty



3/30/2022

CSE 332

9

Stacks in Practice

- Function call stack
- Removing recursion
- Balancing symbols (parentheses)
- Evaluating postfix or “reverse Polish” notation

3/30/2022

CSE 332

10

Algorithm Analysis

Algorithm Analysis

- Correctness:
 - Does the algorithm do what is intended.
- Performance:
 - Speed: **time complexity**
 - Memory: **space complexity**
- Why analyze?
 - To make good design decisions
 - Enable you to look at an algorithm (or code) and identify the bottlenecks, etc.

3/30/2022

CSE 332

11

3/30/2022

CSE 332

12

How to measure performance?

3/30/2022

CSE 332

13

Analyzing Performance

We will focus on analyzing time complexity. First, we have some “rules” to help measure how long it takes to do things:

- Basic operations** Constant time
 - Consecutive statements** Sum of times
 - Conditionals** Test, plus larger branch cost
 - Loops** Sum of iterations
 - Function calls** Cost of function body
 - Recursive functions** Solve recurrence relation...
- Second, we will be interested in **Worse** performance (average and best case sometimes).

3/30/2022

CSE 332

14

Complexity cases

We’ll start by focusing on two cases.

Problem size **N**

- **Worst-case complexity:** **max** # steps algorithm takes on “most challenging” input of size **N**
- **Best-case complexity:** **min** # steps algorithm takes on “easiest” input of size **N**

3/30/2022

CSE 332

15

Exercise - Searching

2	3	5	16	37	50	73	75
---	---	---	----	----	----	----	----

```
bool ArrayContains(int array[], int n, int key){  
  
  
  
  
  
  
  
  
  
}
```

What algorithm would you choose to implement this code snippet?

3/30/2022

CSE 332

16

Linear Search Analysis

```
bool LinearArrayContains(int array[], int n, int key ) {  
    for( int i = 0; i < n; i++ ) {  
        if( array[i] == key )  
            // Found it!  
            return true;  
    }  
    return false;  
}
```

Best Case:

Worst Case:

3/30/2022

CSE 332

17

Binary Search Analysis

2	3	5	16	37	50	73	75
---	---	---	----	----	----	----	----

```
bool BinArrayContains( int array[], int low, int high, int key ) {  
    //The subarray is empty  
    if( low > high ) return false;  
  
    //Search this subarray recursively  
    int mid = (high + low) / 2;  
    if( key == array[mid] ) {  
        return true;  
    } else if( key < array[mid] ) {  
        return BinArrayFind( array, low, mid-1, key );  
    } else {  
        return BinArrayFind( array, mid+1, high, key );  
    }  
}
```

Best case:

Worst case:

3/30/2022

CSE 332

18

Solving Recurrence Relations

1. Determine the recurrence relation and base case(s).
2. "Expand" the original relation to find an equivalent expression *in terms of the number of expansions (k)*.
3. Find a closed-form expression by setting *k* to a value which reduces the problem to a base case

3/30/2022

CSE 332

19

Linear Search vs Binary Search

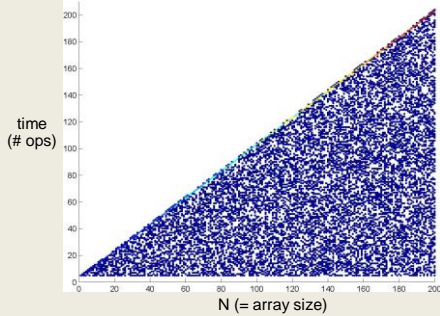
	Linear Search	Binary Search
Best Case	4	5 at [middle]
Worst Case	$3n+3$	$7 \lfloor \log n \rfloor + 9$

3/30/2022

CSE 332

20

Linear search—empirical analysis

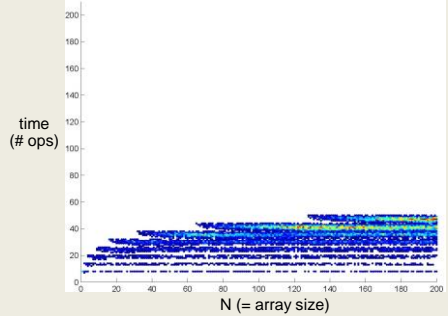


3/30/2022

Each search produces a dot in above graph.
Blue = less frequently occurring, Red = more frequent

21

Binary search—empirical analysis

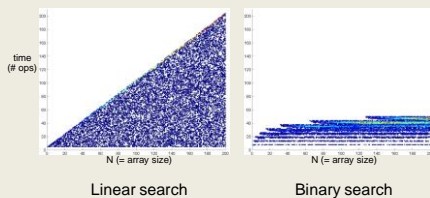


3/30/2022

Each search produces a dot in above graph.
Blue = less frequently occurring, Red = more frequent

22

Empirical comparison



Gives additional information

3/30/2022

CSE 332

23

Asymptotic Analysis

- Consider only the *order* of the running time
 - A valuable tool when the input gets "large"
 - Ignores the effects of *different machines* or *different implementations* of same algorithm

24

Asymptotic Analysis

- To find the asymptotic runtime, throw away the constants and low-order terms

– Linear search is $T_{\text{worst}}^{\text{LS}}(n) = 3n + 3 \in O(n)$

– Binary search is $T_{\text{worst}}^{\text{BS}}(n) = 7 \lfloor \log_2 n \rfloor + 9 \in O(\log n)$

Remember: the "fastest" algorithm has the slowest growing function for its runtime

3/30/2022

CSE 332

25

Asymptotic Analysis

Eliminate low order terms

– $4n + 5 \Rightarrow$

– $0.5 n \log n + 2n + 7 \Rightarrow$

– $n^3 + 3 \cdot 2^n + 8n \Rightarrow$

Eliminate coefficients

– $4n \Rightarrow$

– $0.5 n \log n \Rightarrow$

– $3 \cdot 2^n \Rightarrow$

3/30/2022

CSE 332

26

Properties of Logs

Basic:

- $A^{\log_A B} = B$
- $\log_A A = 1$

Independent of base:

- $\log(AB) = \log A + \log B$
- $\log(A/B) = \log A - \log B$
- $\log(A^B) = B \log A$
- $\log((A^B)^C) = BC \log A$

3/30/2022

CSE 332

27

Properties of Logs

Changing base \rightarrow multiply by constant

– For example: $\log_2 x = 3.22 \log_{10} x$

– More generally

$$\log_A n = \left(\frac{1}{\log_B A} \right) \log_B n$$

– Means we can ignore the base for asymptotic analysis (since we're ignoring constant multipliers)

3/30/2022

CSE 332

28

Another example

- Eliminate low-order terms
- Eliminate constant coefficients

$$16n^3 \log_8(10n^2) + 100n^2$$

3/30/2022

CSE 332

29

Comparing functions

- $f(n)$ is an **upper bound** for $h(n)$ if $h(n) \leq f(n)$ for all n

This is too strict – we mostly care about *large* n

Still too strict if we want to ignore *scale factors*

3/30/2022

CSE 332

30

Definition of Order Notation

- $h(n) \in O(f(n))$ **Big-O "Order"**
if there exist positive constants c and n_0
such that $h(n) \leq c f(n)$ for all $n \geq n_0$

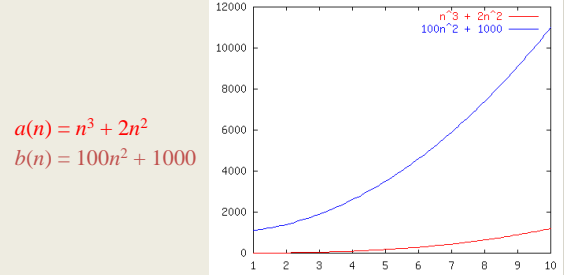
$O(f(n))$ defines a class (set) of functions

3/30/2022

CSE 332

31

Order Notation: Intuition



$$a(n) = n^3 + 2n^2$$

$$b(n) = 100n^2 + 1000$$

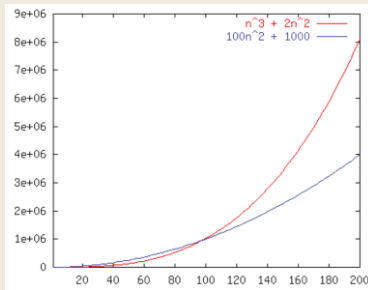
Although not yet apparent, as n gets "sufficiently large", $a(n)$ will be "greater than or equal to" $b(n)$

3/30/2022

CSE 332

32

Order Notation: Example



$$100n^2 + 1000 \leq (n^3 + 2n^2) \text{ for all } n \geq 100$$

$$\text{So } 100n^2 + 1000 \in O(n^3 + 2n^2)$$

3/30/2022

CSE 332

33

Example

$h(n) \in O(f(n))$ iff there exist positive constants c and n_0 such that:
 $h(n) \leq c f(n)$ for all $n \geq n_0$

Example:

$$100n^2 + 1000 \leq 1(n^3 + 2n^2) \text{ for all } n \geq 100$$

$$\text{So } 100n^2 + 1000 \in O(n^3 + 2n^2)$$

3/30/2022

CSE 332

34

Constants are not unique

$h(n) \in O(f(n))$ iff there exist positive constants c and n_0 such that:
 $h(n) \leq c f(n)$ for all $n \geq n_0$

Example:

$$100n^2 + 1000 \leq 1(n^3 + 2n^2) \text{ for all } n \geq 100$$

$$100n^2 + 1000 \leq 1/2(n^3 + 2n^2) \text{ for all } n \geq 198$$

3/30/2022

CSE 332

35

Another Example: Binary Search

$h(n) \in O(f(n))$ iff there exist positive constants c and n_0 such that:
 $h(n) \leq c f(n)$ for all $n \geq n_0$

$$\text{Is } 7\log_2 n + 9 \in O(\log_2 n)?$$

3/30/2022

CSE 332

36

Order Notation: Worst Case Binary Search

3/30/2022

CSE 332

37

Some Notes on Notation

Sometimes you'll see (e.g., in Weiss)

$$h(n) = O(f(n))$$

or

$$h(n) \text{ is } O(f(n))$$

These are equivalent to

$$h(n) \in O(f(n))$$

3/30/2022

CSE 332

38

Big-O: Common Names



- constant: $O(1)$
- logarithmic: $O(\log n)$ ($\log_2 n, \log n^2 \in O(\log n)$)
- linear: $O(n)$
- log-linear: $O(n \log n)$
- quadratic: $O(n^2)$
- cubic: $O(n^3)$
- polynomial: $O(n^k)$ (k is a constant)
- exponential: $O(c^n)$ (c is a constant > 1)

3/30/2022

CSE 332

39

Asymptotic Lower Bounds

- $\Omega(g(n))$ is the set of all functions asymptotically **greater than or equal to** $g(n)$
- $h(n) \in \Omega(g(n))$ iff
There exist $c > 0$ and $n_0 > 0$ such that $h(n) \geq c g(n)$ for all $n \geq n_0$

3/30/2022

CSE 332

40

Asymptotic Tight Bound

- $\Theta(f(n))$ is the set of all functions asymptotically **equal to** $f(n)$
- $h(n) \in \Theta(f(n))$ iff
 $h(n) \in O(f(n))$ and $h(n) \in \Omega(f(n))$
- This is equivalent to:

$$\lim_{n \rightarrow \infty} h(n)/f(n) = c \neq 0$$

3/30/2022

CSE 332

41

Full Set of Asymptotic Bounds

- $O(f(n))$ is the set of all functions asymptotically **less than or equal to** $f(n)$
 - $o(f(n))$ is the set of all functions asymptotically **strictly less than** $f(n)$
- $\Omega(g(n))$ is the set of all functions asymptotically **greater than or equal to** $g(n)$
 - $\omega(g(n))$ is the set of all functions asymptotically **strictly greater than** $g(n)$
- $\Theta(f(n))$ is the set of all functions asymptotically **equal to** $f(n)$

3/30/2022

CSE 332

42

Formal Definitions

- $h(n) \in O(f(n))$ iff
There exist $c > 0$ and $n_0 > 0$ such that $h(n) \leq c f(n)$ for all $n \geq n_0$
- $h(n) \in o(f(n))$ iff
There exists an $n_0 > 0$ such that $h(n) < c f(n)$ for all $c > 0$ and $n \geq n_0$
– This is equivalent to: $\lim_{n \rightarrow \infty} h(n)/f(n) = 0$
- $h(n) \in \Omega(g(n))$ iff
There exist $c > 0$ and $n_0 > 0$ such that $h(n) \geq c g(n)$ for all $n \geq n_0$
- $h(n) \in \omega(g(n))$ iff
There exists an $n_0 > 0$ such that $h(n) > c g(n)$ for all $c > 0$ and $n \geq n_0$
– This is equivalent to: $\lim_{n \rightarrow \infty} h(n)/g(n) = \infty$
- $h(n) \in \Theta(f(n))$ iff
 $h(n) \in O(f(n))$ and $h(n) \in \Omega(f(n))$
– This is equivalent to: $\lim_{n \rightarrow \infty} h(n)/f(n) = c \neq 0$

3/30/2022

CSE 332

43

Big-Omega et al. Intuitively

Asymptotic Notation	Mathematics Relation
O	\leq
Ω	\geq
Θ	$=$
o	$<$
ω	$>$

44

Complexity cases (revisited)

Problem size **N**

- **Worst-case complexity:** **max** # steps algorithm takes on “most challenging” input of size **N**
- **Best-case complexity:** **min** # steps algorithm takes on “easiest” input of size **N**
- **Average-case complexity:** **avg** # steps algorithm takes on *random* inputs of size **N**
- **Amortized complexity:** **max** total # steps algorithm takes on **M** “most challenging” *consecutive* inputs of size **N**, divided by **M** (i.e., divide the max total by **M**).

3/30/2022

CSE 332

45

Bounds vs. Cases

Two orthogonal axes:

- **Bound Flavor**
 - Upper bound (O, o)
 - Lower bound (Ω, ω)
 - Asymptotically tight (Θ)
- **Analysis Case**
 - Worst Case (Adversary), $T_{\text{worst}}(n)$
 - Average Case, $T_{\text{avg}}(n)$
 - Best Case, $T_{\text{best}}(n)$
 - Amortized, $T_{\text{amort}}(n)$

One can estimate the bounds for any given case.

3/30/2022

CSE 332

46