

CSE 332: Data Structures and Parallelism

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

Announcements

- My office hours (CSE2 344)
 - Tuesday, 10 AM – 11 AM
 - Friday, 3 PM - 4 PM
- Exercise #1: Due Monday
 - 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

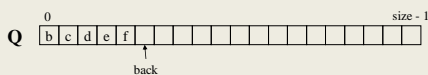


Queues in practice

- Print jobs
- File serving
- Phone calls and operators

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

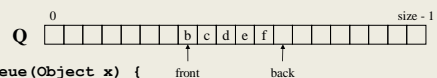
Array Queue Data Structure



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

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

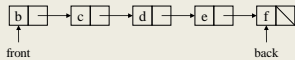
Circular Array Queue Data Structure



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

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

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
    }
}

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

bool is_empty() {
    return front == null
}
    
```

9/30/2022

CSE 332

7

Circular Array vs. Linked List

- Advantages of a circular array
- Advantages of a linked list

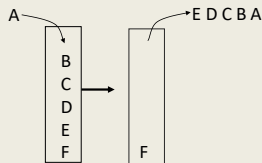
9/30/2022

CSE 332

8

Second Example: Stack ADT

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



9/30/2022

CSE 332

9

Stacks in Practice

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

9/30/2022

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.

9/30/2022

CSE 332

11

9/30/2022

CSE 332

12

How to measure performance?

9/30/2022

CSE 332

13

Analyzing Performance

- Focus on **Worst Case Time Complexity**

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...

9/30/2022

CSE 332

14

Worst case complexity

Problem size **N**

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

9/30/2022

CSE 332

15

Exercise - Searching

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

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

9/30/2022

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:

9/30/2022

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:

9/30/2022

CSE 332

18

Solving Recurrences

$$T(n) = T(n/2) + 7; T(1) = 9$$

1. Determine the recurrence relations and base cases
2. Expand relation in terms of number of expansions k
3. Find a closed form by setting k to value that reduces problem to the base case

9/30/2022

CSE 332

19

Linear Search vs Binary Search

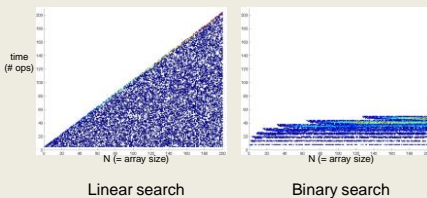
	Linear Search	Binary Search
Best Case	4	5 at [middle]
Worst Case	$3n+3$	$7 \lfloor \log n \rfloor + 9$

9/30/2022

CSE 332

20

Empirical comparison



9/30/2022

CSE 332

21

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

22

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

9/30/2022

CSE 332

23

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

- $4n \Rightarrow$
- $0.5 n \log n \Rightarrow$
- $3 \cdot 2^n \Rightarrow$

9/30/2022

CSE 332

24

Properties of Logs

Basic:

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

Independent of base:

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

9/30/2022

CSE 332

25

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)

9/30/2022

CSE 332

26