

CS151 Intro to Data Structures

Interfaces

Algorithm Analysis

Announcements

- HW02
 - Linked Lists
 - due Friday October 3rd
 - start early!

Outline

- In class assignment
- Objects
 - Comparing Objects
- Interfaces
- Algorithm Analysis

In class assignment review

DLL moveToFront

Array moveToFront

Interfaces

- An interface is a contract - A set of shared methods that users **must** implement
- create a program to calculate the area of different shapes, such as circles, rectangles, triangles etc.
- For each shape, you should be able to print the shape name and area
- Every time someone adds a new shape, they **must** include the methods for getName() and getArea()

Interfaces

- For any new shape that is created, we want to **enforce** that these methods are also implemented.

```
interface Shape {  
    public double getArea();  
    public String getName();  
}
```

```
class Circle implements Shape {
```

Interfaces

A contract - A set of shared methods that users **must** implement

A collection of method signatures with no bodies

A class can implement more than one interface

Interfaces

An interface is not a class!

A class is what an object **is**

An interface is what an object **does**

- can not be instantiated

- no constructors

- incomplete methods

Interface

No modifier - implicitly `public`

No instance variables except for constants (`static final`)

Object Comparison

Object Equality

A custom class must define (override) its own `equals`

Object Comparison

- What if we wanted to compare two students by GPA?

```
int compareTo(T o)
```

Parameters:

`o` - the object to be compared.

Returns:

a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

compareTo

compareTo **returns an int, not a Boolean**

Why?

because it needs to convey three outcomes:

- `negative` if smaller compared to the parameter
- `0` if equal
- `positive` if larger compared to the parameter

Comparable interface

The Comparable interface is designed for objects that have an ordering

```
public interface Comparable<T> {  
    int compareTo(T o);  
}
```

Comparable interface

When would we want to use this? **Let's see in code :)**

Now, what if we wanted to sort from highest to lowest GPA

Custom Exceptions

Making Custom Exceptions

Often times we need to raise a custom exception

Extend `Exception` **or** `RuntimeException`

Custom Exceptions

What is the difference between extending from `Exception` rather than `RuntimeException`?

Subclass of `Exception` are checked exceptions – must be treated/caught

Subclass of `RuntimeException` are not checkable during compile time

Computational Complexity

Run Time Complexity

- Mathematical notation used to describe the performance or complexity of an algorithm.
- Hardware independent
- Represents the upper bound of the time complexity in the **worst-case scenario**.
- Helps us understand how the runtime of an algorithm grows ***as the input size increases***.

Big-O Example 1

```
int n = Integer.parseInt(args[0]);  
int power = 1;  
while (power < n) {  
    System.out.print(power + " ");  
    power *= 2;  
}
```

How does the runtime grow as a function of the input size?

$O(\log n)$

Big-O Example 2

```
int fetchFirstElement(int[] arr) {  
    return arr[0];  
}
```

How does the runtime grow as a function of the size of arr?

$O(1)$

Big-O Example 3

```
int n = Integer.parseInt(args[0]);
int tot = 0;
int i = 0;

while (i < n) {
    tot = tot * i;
    i++;

    for (int j=0; j<10000; j++) {
        System.out.println("hello");
    }
}
```

How does the runtime grow as a function of the input size?

Linearly!

$O(n)$

Big-O Example 4

```
int n = Integer.parseInt(args[0]);

for (int i = 0; i > (-1*n); i--) {
    for (int j = 0; j < n; j++) {
        System.out.println(i, j);
    }
}
```

How does the runtime grow as a function of the input size?

Quadratically!

$O(n^2)$

We do n operations n times

Big-O Example 5

```
String[] lst =  
    {"19", "12", "20", "15"};  
  
for (int i=0; i<100; i++) {  
    System.out.println(getNum(lst));  
}  
  
int getNum(int[] arr) {  
    return Integer.parseInt(arr[0]);  
}
```

How does the runtime grow as a function of the size of `lst`?

Constant! The runtime is not affected by the number of elements in `lst`

$O(1)$

Big-O Example 6

```
int[] lst = {1,2,3,4,5,6,7};

for (int i=0; i<lst.length; i++) {
    findMax(lst);
}

int findMax(int[] arr) {
    int max = Integer.MIN_VALUE;
    for (int i=0; i<arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
```

How does the runtime grow as a function of the size of lst?

$O(n^2)$

Space (Memory) Complexity

How much memory a program needs

The space requirements time typically grows with input size. Expressed as a size of the input. (Big O notation)

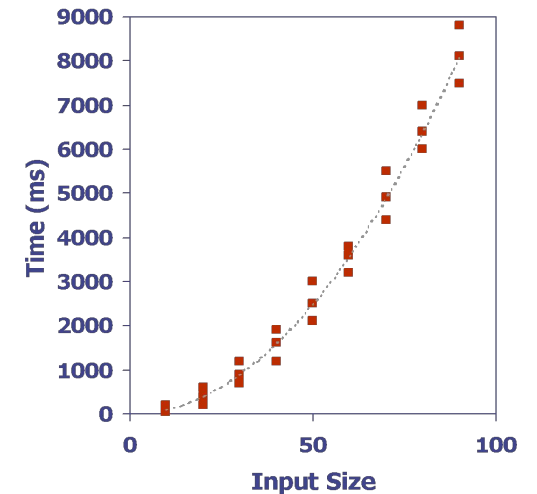
We focus on *worst case* analysis

- how much space will it take in the worst case?

Big O Notation and Theoretical Analysis

- Why do we express runtime notation with Big O notation? Why not just say the run time in number of seconds?

```
1 long startTime = System.currentTimeMillis();           // record the starting time
2 /* (run the algorithm) */
3 long endTime = System.currentTimeMillis();           // record the ending time
4 long elapsed = endTime - startTime;                  // compute the elapsed time
```



- Answer: comparing two algorithms requires exact same hardware and software environments

Constant Time Operations

- Constant time operations require the same amount of time, regardless of the size of the input
- Examples:
 - Basic computations: Assigning variables, adding, multiplying, boolean operators
 - What were some constant time operations in `ExapandableArray`?
 - `LinkedList`?

Linear Time Algorithms: $O(n)$

- The runtime grows linearly as the size of the input grows
- Processes the input in a single pass spending constant time on each item
- Examples:
 - A single loop over an array
 - ExpandableArray?
 - LinkedList?

Example: Find Max

Worst case: $4n + 1 \implies O(n)$

Best case: $3n + 2 \implies O(n)$

Quadratic Time: $O(n^2)$

Nested loops...

Example:

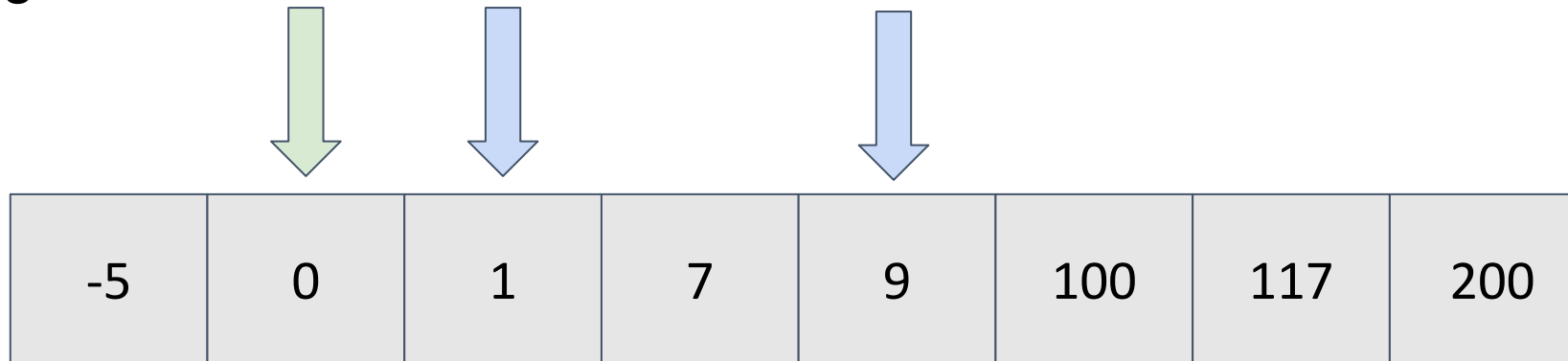
worst case: $4 + 3n^2$

best case: 7

$O(n \log n)$ time

Example: Binary Search!

find 0



How many elements did we touch?

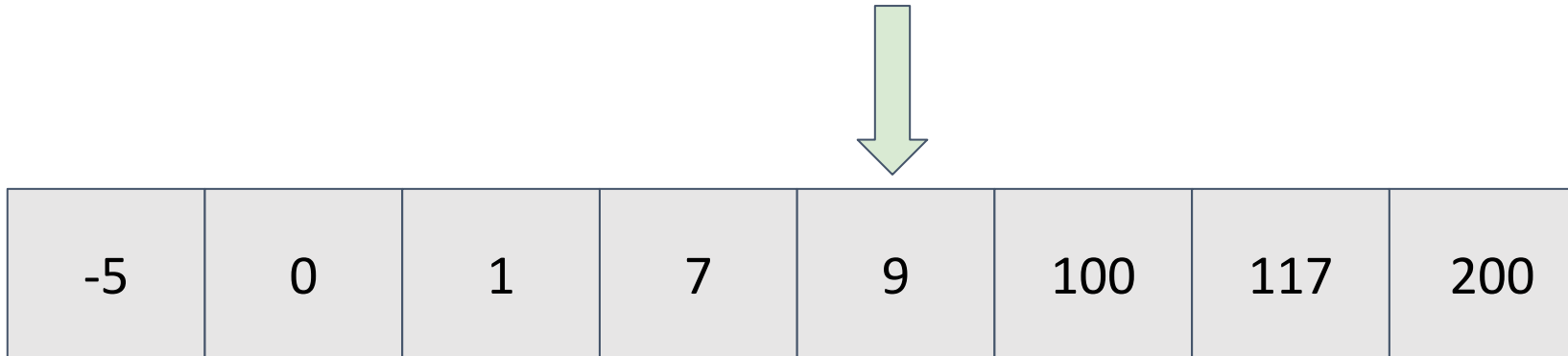
$3 = \log(8)$

Where did the n come from?

$O(n \log n)$ time

Example: Binary Search!

Best case?



Exponential Time: $O(2^n)$

- Generate all possible subsets

$\{a, b, c\} = \dots$

How many subsets are there?

$\{\emptyset\}, \{a\}, \{b\}, \{c\}, \{a,b\}, \{b,c\}, \{a,c\}, \{a,b,c\}$

8

$$2^3 = 8$$

Growth Rate

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
-----	----------	-----	------------	-------	-------	-------

2000-2001	2001-2002	2002-2003	2003-2004	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	2020-2021	2021-2022	2022-2023	2023-2024	2024-2025	2025-2026	2026-2027	2027-2028	2028-2029	2029-2030	2030-2031	2031-2032	2032-2033	2033-2034	2034-2035	2035-2036	2036-2037	2037-2038	2038-2039	2039-2040	2040-2041	2041-2042	2042-2043	2043-2044	2044-2045	2045-2046	2046-2047	2047-2048	2048-2049	2049-2050	2050-2051	2051-2052	2052-2053	2053-2054	2054-2055	2055-2056	2056-2057	2057-2058	2058-2059	2059-2060	2060-2061	2061-2062	2062-2063	2063-2064	2064-2065	2065-2066	2066-2067	2067-2068	2068-2069	2069-2070	2070-2071	2071-2072	2072-2073	2073-2074	2074-2075	2075-2076	2076-2077	2077-2078	2078-2079	2079-2080	2080-2081	2081-2082	2082-2083	2083-2084	2084-2085	2085-2086	2086-2087	2087-2088	2088-2089	2089-2090	2090-2091	2091-2092	2092-2093	2093-2094	2094-2095	2095-2096	2096-2097	2097-2098	2098-2099	2099-2100	2100-2101	2101-2102	2102-2103	2103-2104	2104-2105	2105-2106	2106-2107	2107-2108	2108-2109	2109-2110	2110-2111	2111-2112	2112-2113	2113-2114	2114-2115	2115-2116	2116-2117	2117-2118	2118-2119	2119-2120	2120-2121	2121-2122	2122-2123	2123-2124	2124-2125	2125-2126	2126-2127	2127-2128	2128-2129	2129-2130	2130-2131	2131-2132	2132-2133	2133-2134	2134-2135	2135-2136	2136-2137	2137-2138	2138-2139	2139-2140	2140-2141	2141-2142	2142-2143	2143-2144	2144-2145	2145-2146	2146-2147	2147-2148	2148-2149	2149-2150	2150-2151	2151-2152	2152-2153	2153-2154	2154-2155	2155-2156	2156-2157	2157-2158	2158-2159	2159-2160	2160-2161	2161-2162	2162-2163	2163-2164	2164-2165	2165-2166	2166-2167	2167-2168	2168-2169	2169-2170	2170-2171	2171-2172	2172-2173	2173-2174	2174-2175	2175-2176	2176-2177	2177-2178	2178-2179	2179-2180	2180-2181	2181-2182	2182-2183	2183-2184	2184-2185	2185-2186	2186-2187	2187-2188	2188-2189	2189-2190	2190-2191	2191-2192	2192-2193	2193-2194	2194-2195	2195-2196	2196-2197	2197-2198	2198-2199	2199-2200	2200-2201	2201-2202	2202-2203	2203-2204	2204-2205	2205-2206	2206-2207	2207-2208	2208-2209	2209-2210	2210-2211	2211-2212	2212-2213	2213-2214	2214-2215	2215-2216	2216-2217	2217-2218	2218-2219	2219-2220	2220-2221	2221-2222	2222-2223	2223-2224	2224-2225	2225-2226	2226-2227	2227-2228	2228-2229	2229-2230	2230-2231	2231-2232	2232-2233	2233-2234	2234-2235	2235-2236	2236-2237	2237-2238	2238-2239	2239-2240	2240-2241	2241-2242	2242-2243	2243-2244	2244-2245	2245-2246	2246-2247	2247-2248	2248-2249	2249-2250	2250-2251	2251-2252	2252-2253	2253-2254	2254-2255	2255-2256	2256-2257	2257-2258	2258-2259	2259-2260	2260-2261	2261-2262	2262-2263	2263-2264	2264-2265	2265-2266	2266-2267	2267-2268	2268-2269	2269-2270	2270-2271	2271-2272	2272-
-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-------

Growth Rate

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256

Growth Rate

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536

Growth Rate

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296

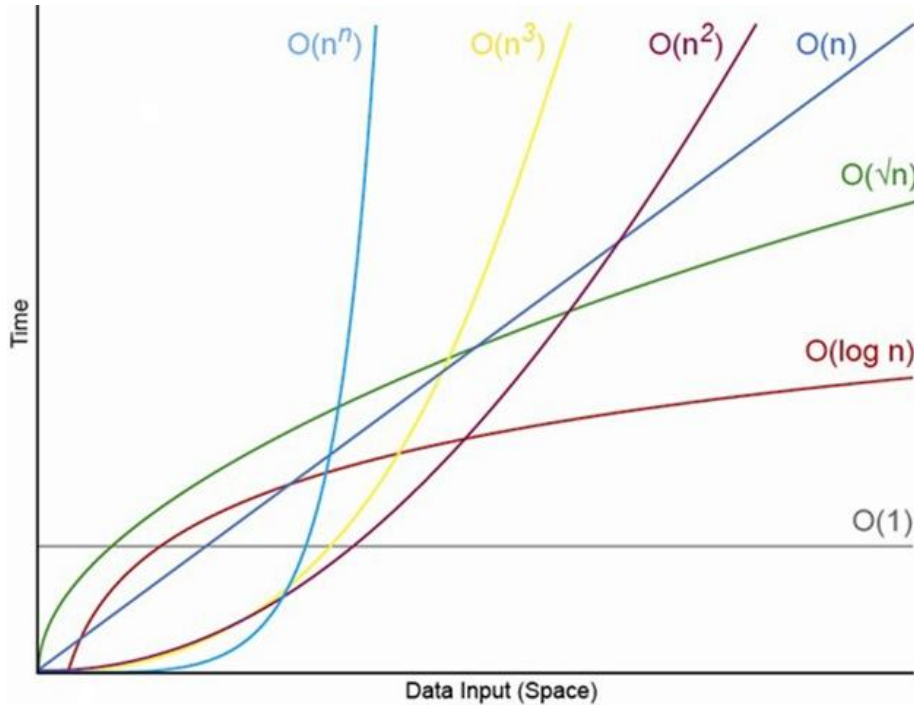
Growth Rate

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84×10^{19}

Growth Rate

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84×10^{19}
128	7	128	896	16,384	2,097,152	3.40×10^{38}
256	8	256	2,048	65,536	16,777,216	1.15×10^{77}
512	9	512	4,608	262,144	134,217,728	1.34×10^{154}

Asymptotic Notation



As the number of elements approaches infinity, only the dominant term matters

That is why we simplify $O(n+1)$ to $O(n)$ etc.

Big- O Analysis

1. Write a polynomial in terms of input size n

- Only loops contribute
- Each nested factor is multiplied
- Each sequential factor is summed

2. Simplify the polynomial

- Identify dominant term – highest degree polynomial
- Polynomials beat polylogs
- Exponentials beat polynomials
- Discard constants

Summary

- Every non-primitive is an OBJECT in Java
- Every Object is a REFERENCE
- Interfaces are a *contract* of which methods you will implement
- You can compare objects with compareTo