

- Informally, *O* notation is the leading (i.e. quickest growing) term of a formula with the coefficient stripped off
- O is sort of a relaxed version of " $\leq$  "

We can then say that Alg1 takes  $O(n^2)$  time. Or, for short, we just say Alg1 is  $O(n^2)$ 

\_\_\_\_\_ Computing big-O of an Algorithm \_\_\_\_\_

Following are some formulas that represent the number of operations of some algorithm. Give the big-O notation for each.

- E.g. n, 10,000n 2000, and .5n + 2 are all O(n)
- $n + \log n$ ,  $n \sqrt{n}$  are O(n)
- $n^2 + n + \log n$ ,  $10n^2 + n \sqrt{n}$  are  $O(n^2)$
- $n \log n + 10n$  is  $O(n \log n)$
- $10 * \log^2 n$  is  $O(\log^2 n)$
- $n\sqrt{n} + n\log n + 10n$  is  $O(n\sqrt{n})$
- 10,000,  $2^{50}$  and 4 are O(1)

bool BinarySearch (int arr[], int s, int e, int key){
if (e-s<=0) return false;
int mid = (e-s)/2;
if (arr[key]==arr[mid]){
 return true;
}else if (key < arr[mid]){
 return BinarySearch (arr,s,mid,key);}
else{
 return BinarySearch (arr,mid,e,key)}</pre>

```
}
```

6

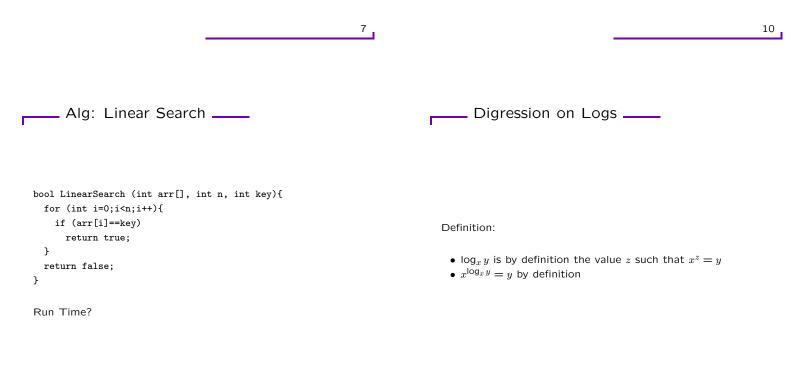
Following is a shorter way to compute big-O for an algorithm:<br/>"Atomic operations"Constant time"Atomic operations"Constant timeConsecutive statementsSum of timesConditionalsLarger branch time plus test timeLoopsSum of iterationsFunction CallsTime of function bodyRecursive FunctionsSolve Recurrence Relation

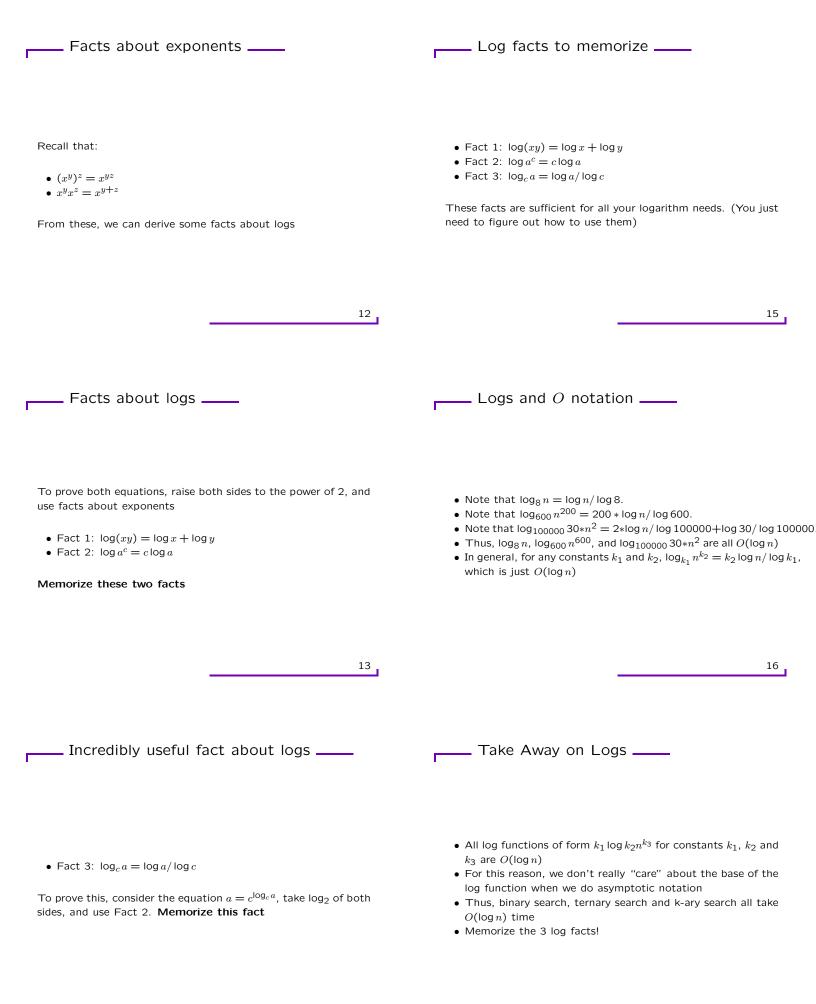
\_\_\_\_ Computing big-O of an Algorithm \_\_\_\_\_

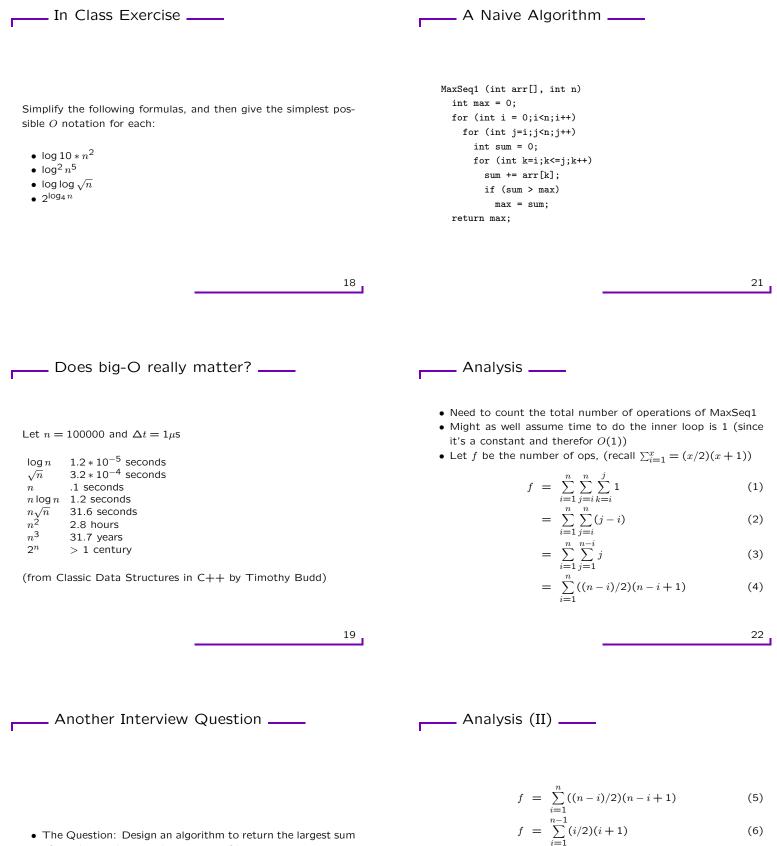
- Note that even in the worst case, the size of the array we search is being split in half in each call
- Thus if x is the number of recursive calls, and n is the original size of the array,  $n(1/2)^x = 1$  in the worst case
- This implies that  $2^x = n$
- Taking log of both sides, we get  $x = \log n$

\_\_\_\_\_ Analysis of Binary Search \_\_\_\_\_

- Since each invocation of the function takes O(1) time (minus the recursive calls), and the total number of invocations is at most log *n*, the running time is  $O(\log n)$
- Much better than Linear Search







 $f = \frac{1}{2} \sum_{i=1}^{n-1} (i^2 + i)$ (7)

$$f = 1/2 * \left(\sum_{i=1}^{n-1} i^2 + \sum_{i=1}^{n-1} i\right)$$
(8)

$$f = 1/2 * (O(n^3) + O(n^2))$$
(9)  
$$f = O(n^3)$$
(10)

$$f = O(n^3) \tag{10}$$

20

of contiguous integers in an array of ints

sum is 8, which we get from (2, 3, -2, 0, 5).

• Example: if the input is (-10, 2, 3, -2, 0, 5, -15), the largest

Challenge \_\_\_\_\_

• MaxSeq1 is very slow

• This kind of algorithm won't impress an interviewer

• Can you do better?

24

\_\_\_\_\_ Todo \_\_\_\_\_

• Finish Chapter 3 (Growth of Functions) in textbook