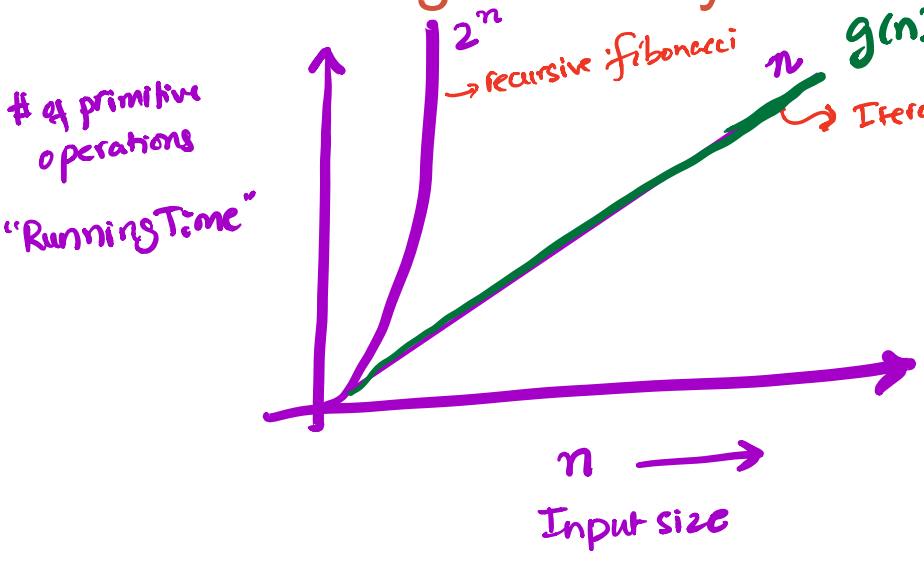
RUNNING TIME ANALYSIS - PART 2
BINARY SEARCH TREES RUNNING TIME

Problem Solving with Computers-II

What does Big-Oh really mean?



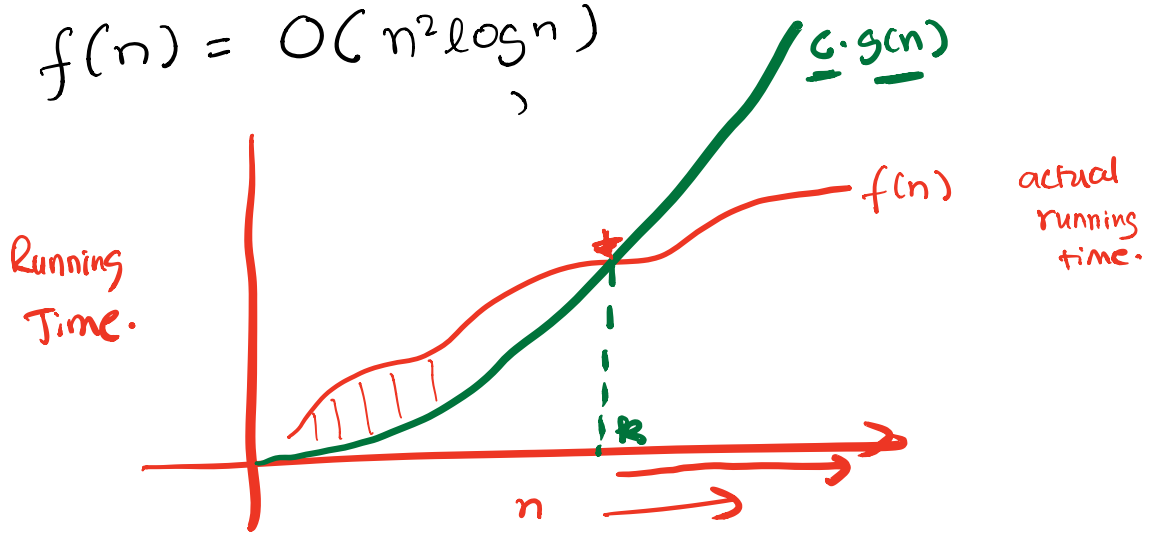
(not the actual running time)

Worst case.

$f_i(n) = O(g(n))$
$= O(n)$

$f_e(n) = O(2^n)$
↓
$s(n)$
greater function

$f(n) = O(n^2 \log n)$



$f(n) = 5n^2 \log n + n + 1000$

$= O(n^2 \log n)$

$c \cdot n^2 \log n > (5n^2 \log n + n + 1000)$ for all $n \geq k$

$f(n) = 5n^2 \log n + n + 1000$

$< 5n^2 \log n + n^2 \log n + n^2 \log n, \quad n > \frac{100}{k}$

$= \underbrace{7}_{c} \underbrace{n^2 \log n}_{g(n)}$

$f(n) < c \cdot g(n)$ for all $n > k$

Formal definition of Big-O

$n^2 \cdot \log n = n^2 \log n$
$n^2 \cdot n = n^3$

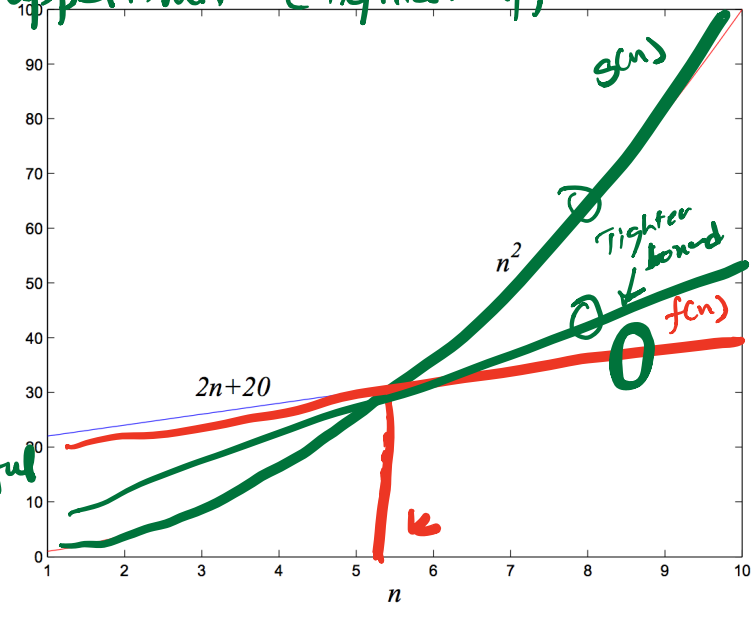
- f(n) and g(n): running times of two algorithms on inputs of size n.
- f(n) and g(n) map positive integer inputs to positive reals.

Choose g(n) to be the smallest upper limit. (tightest upper bound)

We say f = O(g) if there is a constant c > 0 and k>0 such that
f(n) ≤ c · g(n) for all n >= k.

f = O(g)
means that "f grows no faster than g"

$f(n) = 5n^2 \log n + n + 1000$
$= O(n^3)$ okay but not helpful
$= O(n^2 \log n)$ (tighter bound)

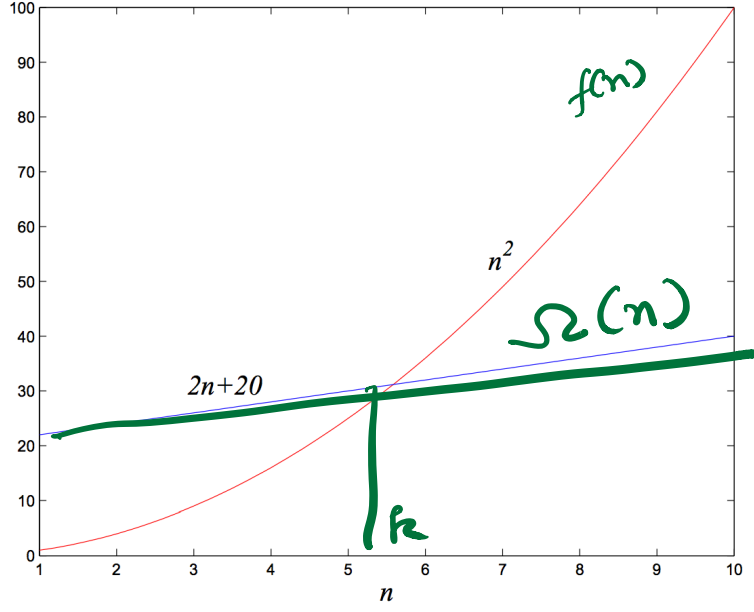


Big-Omega

- f(n) and g(n): running times of two algorithms on inputs of size n.
- f(n) and g(n) map positive integer inputs to positive reals.

We say $f = \Omega(g)$ if there are constants $c > 0, k>0$ such that $c \cdot g(n) \leq f(n)$ for $n >= k$

$f = \Omega(g)$
means that "f grows at least as fast as g"



Big-Theta

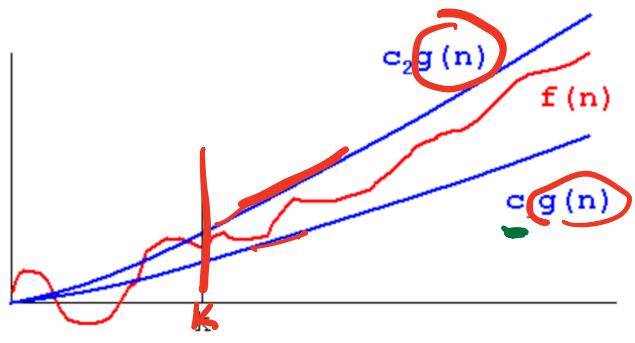
- f(n) and g(n): running times of two algorithms on inputs of size n.
- f(n) and g(n) map positive integer inputs to positive reals.

We say $f = \Theta(g)$ if there are constants $c_1, c_{2\,,k}$ such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$, for $n >= k$

$f(n) = 5n^2 \log n + n + 1000$

$3n^2 \log n \le f(n) \le 7n^2 \log n$

c_1 c_2



Running time

$g(n) = n^2 \log n$

$f(n) = \Theta(n^2 \log n)$

$c_1 = 3$

$c_2 = 7$

What is the Big-O running time of algoX?

- Assume **dataA** is some data structure.
- Given: running time of operations for dataA, where M is the number of keys stored in dataA
  - insert: O(log M)
  - min: O(1)
  - delete: O(log M)

```
void algoX(int arr[], int N)
{
        dataA ds;//ds contains no keys
        for(int i=0; i < N; i++)
                ds.insert(arr[i]);
        for(int i=0; i < N; i++){
                arr[i] = ds.min();
                ds.delete(arr[i]);
        }
}
```

A. $O(N^2)$
B. O(N logN)
C. O(N)
D. O(log N)
E. Not enough information to compute

```
for(int i=0; i < N;   i++)
      ds.insert(arr[i]);
```

Running time of this loop is less than

$c_1 N \log N$

```
for(int i=0; i < N; i++){
      arr[i] = ds.min();
      ds.delete(arr[i]);
}
```

Running time of this loop is less than

$N(c_2 + c_3 \log N)$

Overall running time is

$c_1 N \log N + c_2 N + c_3 N \log N$

$= O(N \log N)$

Reason:

Each insert takes a different amount of time because the running time depends on the number of keys already in ds.

The first insert takes the least time, the last one takes the most.

Although we don't know the exact number of operations for each insert, we can find an upper limit.

Specifically, the running time of each insert is less than $c_1 * \log N$

Best case, worst case, average case running times

Operations on sorted arrays

- Min : O(1)
- Max: O(1)
- Median: O(1)
- Successor: O(1)
- Predecessor: O(1).
- Search:
- Insert : O(N)
- Delete: O(N)

Naïve (Linear) Search

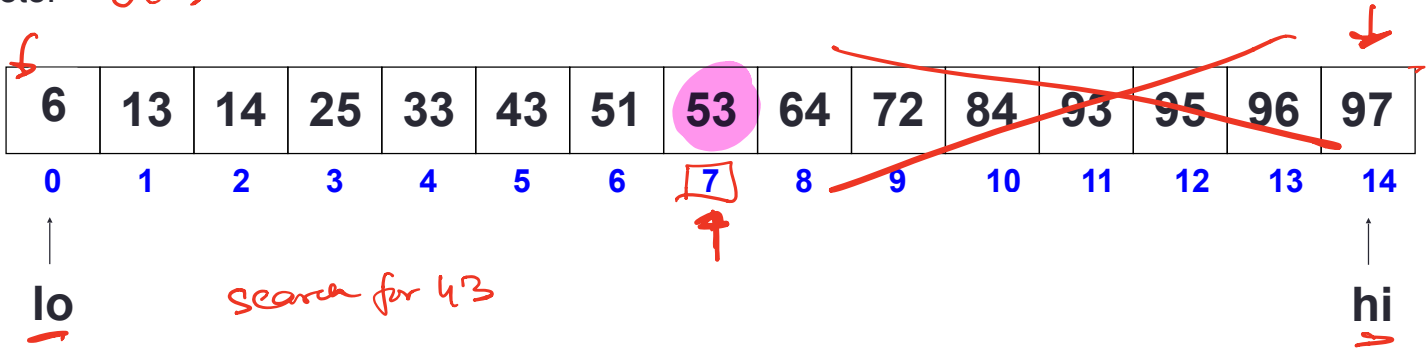
Best case
O(1)
(search key is the min element)

Worst case
O(N)
(search key is the max element)

Binary Search

Best case
O(1)
(search key is the mid element)

Worst case
O(log N)
(key is not in the array)



Worst case analysis of binary search

Worst case

```
bool binarySearch(int arr[], int element, int N){
//Precondition: input array arr is sorted in ascending order
  int begin = 0;
  int end = N-1;
  int mid;
  while (begin <=  end){
    mid = (end + begin)/2;
    if(arr[mid]==element){
      return true;
    }else if (arr[mid]< element){
      begin = mid + 1;
    }else{
      end = mid - 1;

    }
  }
  return false;
}
```

→ O(1)

? How many times does the loop run?

O(log N)

O(1)

Iteration	size of search space
1	N
2	$N/2$
3	$N/4$
⋮	⋮
k	$N/2^{k-1}$

Worst case: Search key is not in the array. In that case the loop stops when the search space is less than 1 as expressed below.

$$\frac{N}{2^{k-1}} < 1$$

Solve for k:

$$N < 2^{k-1}$$

$$k-1 > \log_2 N$$

$$k > (\log_2 N + 1)$$

In the worst case the loop will run $(\log_2 N + 1)$ times.

Using this the running time of binary search is

$$O(1) + O(1) \cdot (\log N + 1)$$

$$= O(\log N)$$