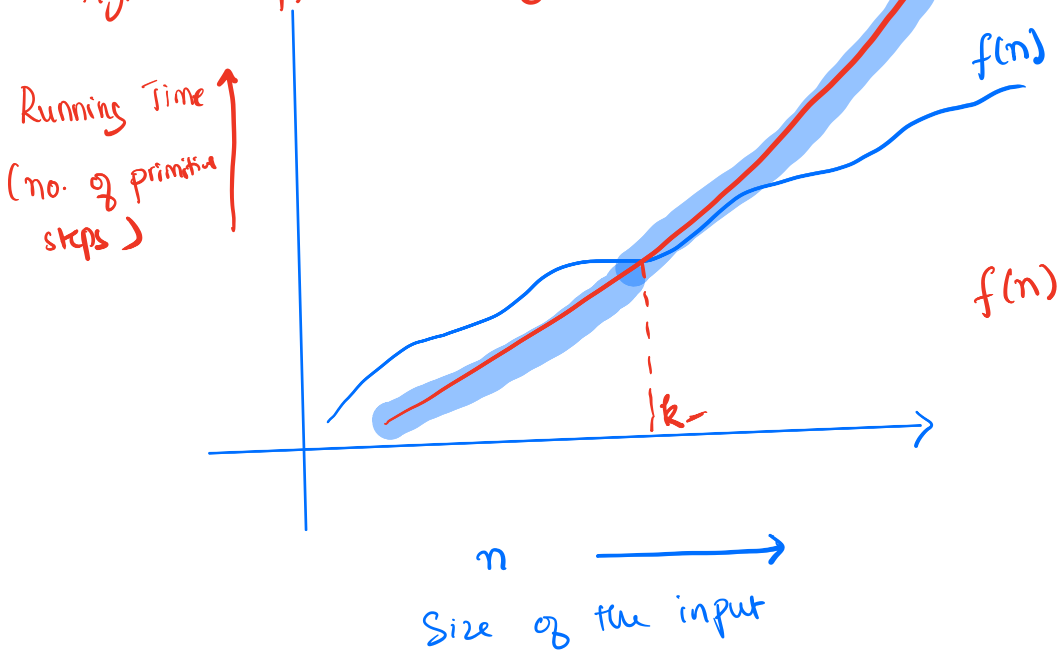
RUNNING TIME ANALYSIS - PART 2

Problem Solving with Computers-II

$f(n)$ grows no faster than $g(n)$ (n^2)

Tightest upper bound $g(n)$



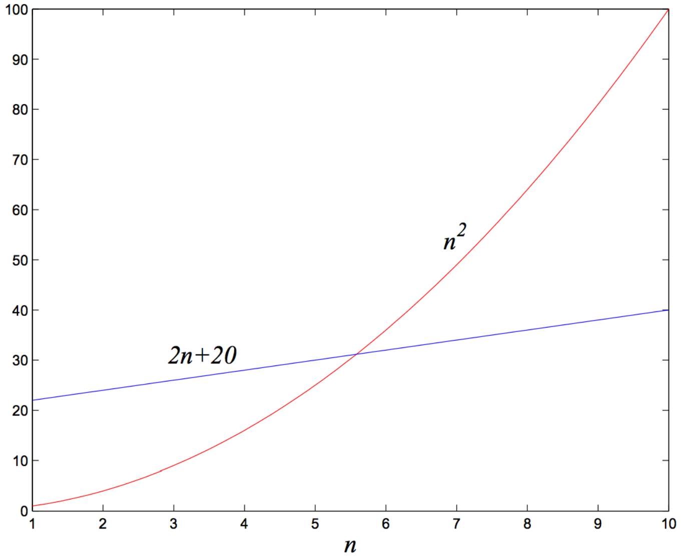
$f(n) = O(g(n))$

Definition of Big-O

f(n) and g(n) map positive integer inputs to positive reals.

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

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



What is the Big O running time of sumArray2

A. $O(n^2)$
B. $O(n)$
C. $O(n/2)$
D. $O(\log n)$
E. None of the array

```
/* n is the length of the array*/
int sumArray2(int arr[], int n)
{
        int result = 0;
        for(int i=0; i < n; i=i+2)
              result+=arr[i];
        return result;
}
```

$T(n) = 1+1+1+$ # of times the loop runs $(1+1+1)$

times the loop runs

A. n

B. $\left\lceil \frac{n}{2} \right\rceil$

C $\frac{n-1}{2}$

$n = 4 \quad \left\lceil \frac{n}{2} \right\rceil = 2$

$n = 5 \quad \left\lceil \frac{n}{2} \right\rceil = 3$

$$T(n) = 3 + \frac{3n}{2}$$

$$T(n) = O(n)$$

Justification for how the above approach is consistent with the definition of Big-O

$$T(n) = 3 + \frac{3n}{2}$$

$$\leq 3n + 3n \quad \text{for } n \geq 1$$

$$= 6n$$

$$T(n) \leq 6n \quad \text{for } n \geq 1$$

$$T(n) = O(n) \quad c = 6, \; k = 1$$

What is the Big O of sumArray3

A. $O(n^2)$
B. $O(n)$
C. $O(n/2)$
D. $O(\log_2 n)$
E. None of the array

```
/* N is the length of the array*/
int sumArray3(int arr[], int n)
{
        int result = 0;
        for(int i = 1; i < n; i=i*2)
            result+=arr[i];
        return result;
}
```

$T(n) = O(1) + \#\text{times the loop runs} * O(1)$

$= O(1) + (\log_2 n + 1)\, O(1)$

$= O(\log_2 n)$

Iteration #	i
1	1
2	$\boxed{2}$
3	4
$\vdots$	$\vdots$
k	$2^{(k-1)}$

Loop stops running when

$$2^{k-1} \geq n$$

$$k-1 \geq \log n$$

$$k \geq (\log n) + 1$$

Given the step counts for different algorithms, express the running time complexity using Big-O

1. $\boxed{10000000}$ $O(1)$
2. 3*n $O(n)$
3. 6*n-2 $O(n)$
4. 15*n + 44 $O(n)$
5. 50*n*log(n) $O(n \log n)$
6. n^2 $O(n^2)$
7. n^2-6n+9 $O(n^2)$
8. $3n^2+4*\log(n)+1000$ $O(n^2)$

For polynomials, use only leading term, ignore coefficients: linear, quadratic

Common sense rules of Big-O

1. Multiplicative constants can be omitted: $14n^2$ becomes n^2 .

2. n^a dominates n^b if $a > b$: for instance, n^2 dominates n.

3. Any exponential dominates any polynomial: 3^n dominates n^5 (it even dominates 2^n).

$$T(n) = 3^n + n^5$$

$$= O(3^n)$$

Best case and worst case running times

Operations on sorted arrays of size n

- Min : O(1)
- Max: O(1)
- Median: O(1)
- Successor: O(1)
- Predecessor: O(1)
- Search: (naive approach) (binary search)
- Insert :
- Delete:

	Best case	Worst case
	O(1)	O(1)
	O(1)	O(1)
	O(1)	O(1)
	O(1)	O(1)
naive search	O(1)	O(n)
Binary search	O(1)	O(log n)
Insert	O(1)	O(n)
Delete	O(1)	O(n)

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

arr[0]

$arr[\lfloor \frac{n}{2} \rfloor]$

Worst case analysis of binary search

```
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;
}
```

$O(1)$

$O(1)$

Iteration #	end - begin
1	$n-1$
2	$\frac{n-1}{2}$
3	$\frac{n-1}{2^2}$
⋮	
k	$\frac{n-1}{2^{k-1}}$

Loop stops when end - begin < 1

Loop stops when (end - begin) < 1

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

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

$$\log(n-1) < k-1$$

$$k > \log(n-1) + 1$$

$$T(n) = O(1) + (\log(n-1) + 1) \cdot O(1)$$

$$= O(\log n)$$