RUNNING TIME ANALYSIS - PART 2

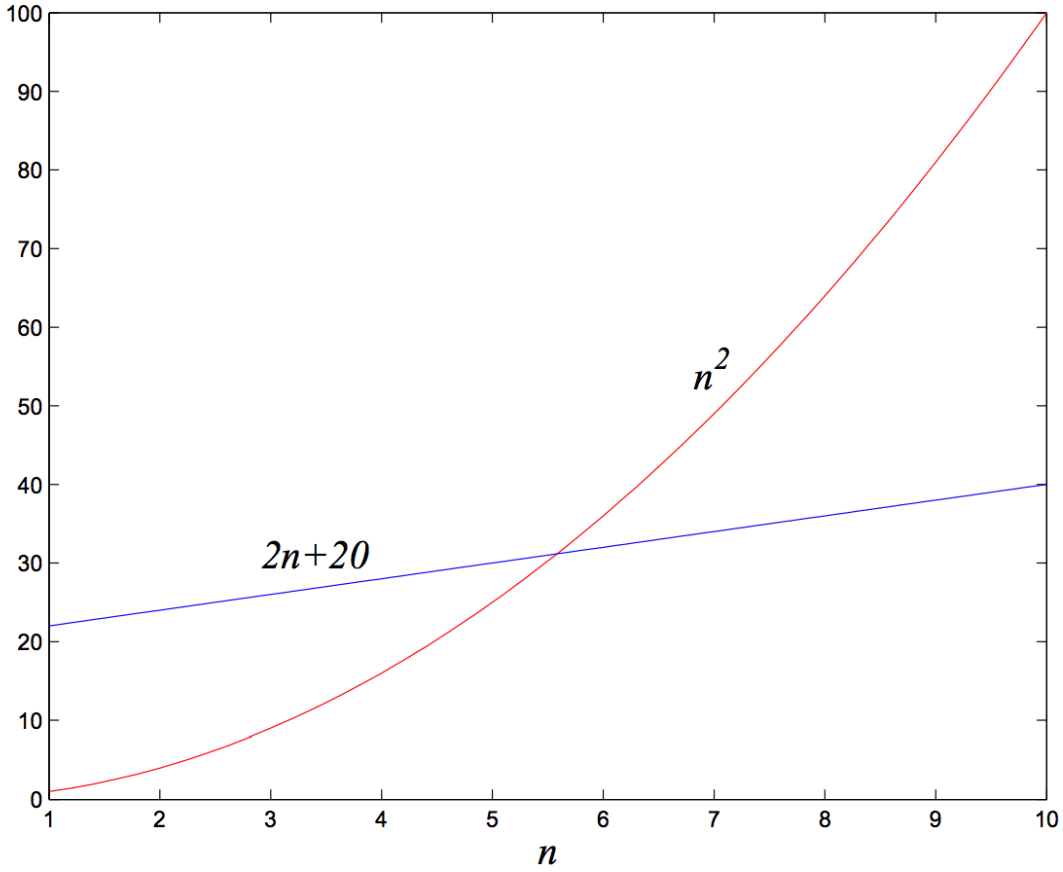
Problem Solving with Computers-II

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) ≤ c · 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;
}
```

What is the Big O of sumArray3

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 sumArray3(int arr[], int n)
{
        int result = 0;
        for(int i= 1; i < n; i=i*2)
                result+=arr[i];
        return result;
}
```

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

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

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).

Best case and worst case running times

Operations on sorted arrays of size n

- Min :
- Max:
- Median:
- Successor:
- Predecessor:
- Search:
- Insert :
- Delete:

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

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;
}
```