Lecture 6
Time Complexity of Recursive Algorithms

1. measure running time in terms of input
2. calculate Big-Oh of the function

Base case and recursive step

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot f(n-1) & else \end{cases}$$

```
int fact(int n){
    if(n==0)
        return 1;
    return n*fact(n-1);
}
```

Recursive T(n)

```
int fact(int n){
    if(n==1)
        return 1;
    return n*fact(n-1);
}
```

$$T(n) = \begin{cases} 2 & \text{if } n = 0 \\ T(n-1) + 4 & \text{if } n > 0 \end{cases}$$

How to get the closed form?

- expand T(n)
- observe progression

$$
\begin{aligned}
T(n) &= T(n-1) + 4 \\
&= T(n-2) + 4 + 4 \\
&= T(n-3) + 4 + 4 + 4 \\
&= T(n-k) + 4 + 4 \text{ (k times)} \\
&= 2 + 4 + 4 \ldots \text{(n times)} \\
&= 2 + 4n
\end{aligned}
$$

$$T(n) = 2+ 4n$$

$$O(n)$$

1. obtain recursive T(n)

2. convert to closed form

3. obtain Big-Oh of T(n)

Searching for an Element

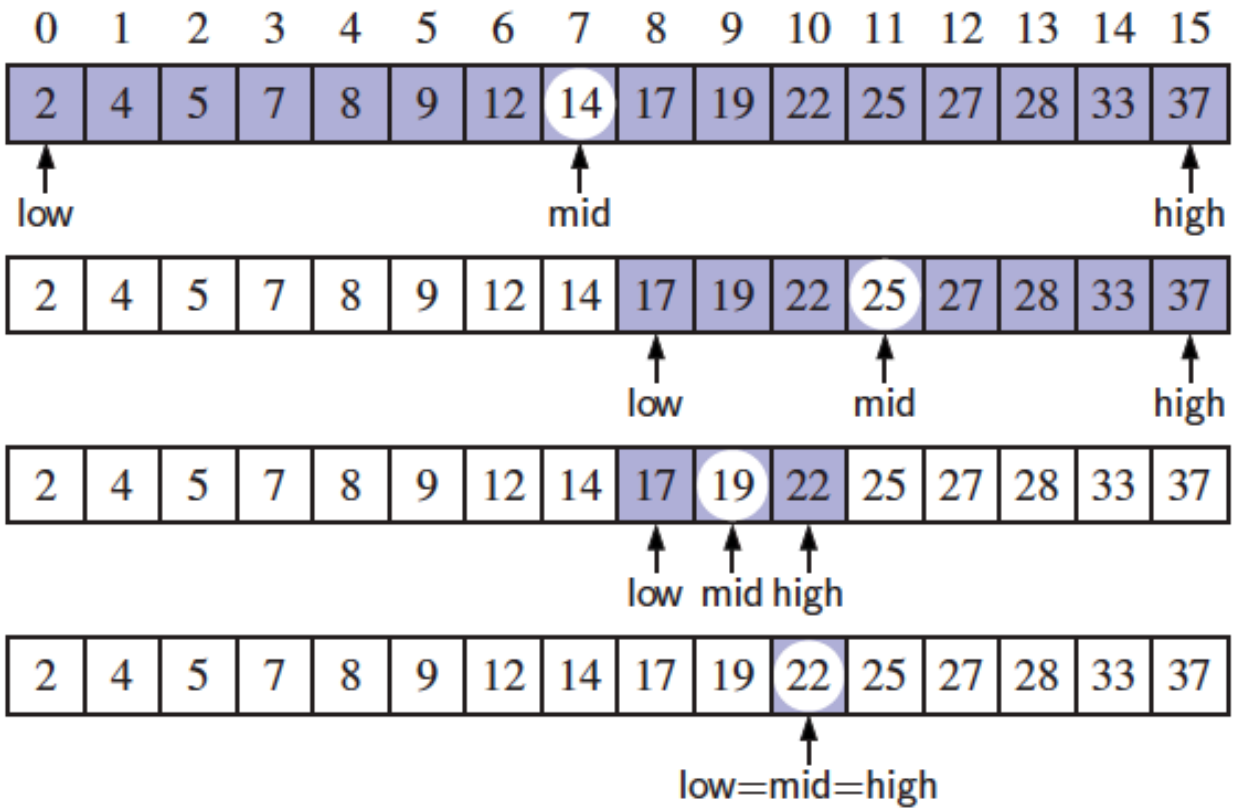
Problem: Search for an element in a sorted array!

Naïve approach = O(n)

Can we do it faster?

Binary Search

```
bool binarySearch(int A[],int K, int low, int high) {
    if (low == high)
        return K==A[low];
    else {
            int mid = (low + high) / 2;
            if (K == A[mid]) return true;
            else if (K < A[mid])
                return binarySearch(A, K, low, mid - 1);
            else// (x >= A[mid])
                return binarySearch(A, K, mid + 1, high);
    }
}
```



Time complexity?

```
int pow(int a, int n) {
  if (n == 1)
    return a;
  return a*pow(a, n-1);
}
```

Exercise: write log n algorithm for computing powers!

```
int pow(int a, int n) {
  if (n == 1)
    return a;
  if (n% 2 == 0)
    return pow(a*a, n/2);
  return pow(a*a, n/2) * a;
}
```