Running Time of a Recursive Function

- We have seen how we can count statements involving assignments, comparisons, looping, ..
- What happens when we have a recursive function?

Running Time of a Recursive Function

- Let’s start with an easy one, factorial

```
public static int factorial( n )
{
 if ( n == 0 )
    return 1;
  else
    return n * factorial( n – 1);
}
```

Running Time of a Recursive Function

- $T(n) = a + ???$
- ??? = time for recursive call, plus multiplication plus return statement

Running Time of a Recursive Function

- $T(n) = a + b + T(n - 1)$
- $T(n) = c + T(n - 1)$

- Where c is a positive constant

Running Time of a Recursive Function

- $T(n) = c + T(n - 1)$

- We drill down, using derivation, to come up with a formula for $T(n)$ as a function of n

Running Time of a Recursive Function

- $T(n) = c + T(n – 1)$ // equation (1)
- $T(x) = c + T(x – 1)$ // equation (A)

- We substitute x for n - 1 in equation (A)
- T(n - 1) = c + T(???) // equation (2)

Running Time of a Recursive Function

- $T(n) = c + T(n – 1)$ // equation (1)
- $T(n - 1) = c + T(n - 2)$ // equation (2)
- We now get:
- $T(n) = c + c + T(n – 2)$
- $T(n) = 2c + T(n – 2)$ // equation (3)

Running Time of a Recursive Function

- $T(n) = 2c + T(n – 2)$ // equation (3)
- $T(x) = c + T(x – 1)$ // equation (A)

- We substitute x for n - 2 in equation (A)
- $T(n - 2) = c + T(???)$ // equation (4)

Running Time of a Recursive Function

- $T(n) = 2c + T(n - 2)$ // equation (3)
- $T(n - 2) = c + T(n - 3)$ // equation (4)
- We get:
- $T(n) = 2c + c + T(n - 3)$
- $T(n) = 3c + T(n - 3)$ // equation (5)

Running Time of a Recursive Function

- $T(n) = c + T(n - 1)$
- $T(n) = 2c + T(n - 2)$
- $T(n) = 3c + T(n - 3)$

- Do we see a general pattern here?

Running Time of a Recursive Function

- $T(n) = k\,c + T(n - k)$

- We reach the base case (or bottom of recursion) when the argument is 0
- $T(n) = k\,c + T(0)$ occurs when $n = k$
- ➔ $T(n) = c\,n + T(0) = c\,n + d$
- $T(n)$ is $\Theta(n)$

Running Time of a Recursive Function

- We can formally prove that
- $T(n) = c\,n + T(0)$
- How would you do that?

Running Time of a Recursive Function

- We can formally prove that
- $T(n) = c\, n + T(0)$
- How would you do that?

- By Induction (this is left as an exercise)

Running Time of a Recursive Function

- $T(n) = c + T(n - 1)$
- We could also generate a general formula from the bottom up

- If $n = 1$:
- $T(1) = c + T(0)$

Running Time of a Recursive Function

- $T(n) = c + T(n - 1)$
- $T(1) = c + T(0)$
- $T(2) = c + T(1) = c + c + T(0) = 2c + T(0)$
- $T(3) = c + T(2) = c + 2c + T(0)$
- $T(3) = 3c + T(0)$

Running Time of a Recursive Function

- $T(1) = c + T(0)$
- $T(2) = 2c + T(0)$
- $T(3) = 3c + T(0)$
- More generally,
- $T(k) = kc + T(0)$
- When $k = n$
- $T(n) = nc + T(0) = cn + T(0)$

MCS – Divide and Conquer Approach – Running Time Analysis

```
mcs( list, l, r )                                  // T( n )
        if l == r                                  // c1
         return list[l]
      else // general case
         c = ( l + r ) / 2                         // c2
         lmax = mcs( list, l, c )                  // T( n / 2 )
         rmax = mcs( list, c + 1, r )              // T( n / 2 )
         cmax = MCS stradling center (need to compute) // Θ( n )
         return maximum( lmax, rmax, cmax ) // c3
```

MCS – Divide and Conquer Approach – Running Time Analysis

$T(n) = c1 + c2 + T(n/2) + T(n/2) + \Theta(n) + c3$

$T(n) = 2\,T(n/2) + \Theta(n) + c1 + c2 + c3$

$T(n) = 2\,T(n/2) + \Theta(n) + c4$ // $c4 = c1 + c2 + c3$

$T(n) = 2\,T(n/2) + \Theta(n)$ // $c4$ is $\Theta(1)$, less than $\Theta(n)$

MCS Divide and Conquer

- Let's simplify and say that:
- $T(n) = 2 T(n / 2) + c n$
- where c is some positive constant

- Note: $\Theta(c n)$ is $\Theta(n)$ and vice versa

MCS Divide and Conquer

- $T(n) = 2 T(n / 2) + c n$
- If we use x instead of n:
- $T(x) = 2 T(x / 2) + c x$

- The idea is to “drill down” and eventually arrive at the base case and its running time

MCS Divide and Conquer

- $T(n) = 2T(n/2) + cn$ // equation (1)
- If we use x instead of n:
- $T(x) = 2T(x/2) + cx$
- Substituting x for n / 2 above, we get:
- $T(n/2) = 2T(n/2^2) + cn/2$
- We plug that into equation (1)

MCS Divide and Conquer

$T(n) = 2\,T(n/2) + c\,n$ // equation (1)

$T(n/2) = 2T(n/2^2) + c\,n/2$

$T(n) = 2\,(\,2T(n/2^2) + c\,n/2\,) + c\,n$

$T(n) = 2^2\,T(n/2^2) + c\,n + c\,n$

$T(n) = 2^2\,T(n/2^2) + 2\,c\,n$

MCS Divide and Conquer

- Let's keep drilling down

$T(n) = 2^2 \, T(n / 2^2) + 2cn$

$T(x) = 2\,T(x / 2) + cx$ // equation (2)

Plug in $n / 2^2$ for x in equation (2)

$T(n / 2^2) = 2\,T(n / 2^3) + cn / 2^2$

MCS Divide and Conquer

$T(n) = 2^2 T(n/2^2) + 2cn$

$T(n/2^2) = 2T(n/2^3) + cn/2^2$

- Replace $T(n/2^2)$ by its value above

$T(n) = 2^2 (2T(n/2^3) + cn/2^2) + 2cn$

$T(n) = 2^3 T(n/2^3) + cn + 2cn$

$T(n) = 2^3 T(n/2^3) + 3cn$

MCS Divide and Conquer

$T(n) = 2\,T(n/2) + c\,n$

$T(n) = 2^2\,T(n/2^2) + 2\,c\,n$

$T(n) = 2^3\,T(n/2^3) + 3\,c\,n$

Do we see a pattern?

Note: we can rewrite the first one as

$T(n) = 2^1\,T(n/2^1) + 1\,c\,n$

MCS Divide and Conquer

$T(n) = 2\,T(n/2) + c\,n$

$T(n) = 2^2\,T(n/2^2) + 2\,c\,n$

$T(n) = 2^3\,T(n/2^3) + 3\,c\,n$

$T(n) = ??\,T(n/2^k) + ??$

MCS Divide and Conquer

$T(n) = 2\,T(n/2) + c\,n$

$T(n) = 2^2\,T(n/2^2) + 2\,c\,n$

$T(n) = 2^3\,T(n/2^3) + 3\,c\,n$

$T(n) = 2^k\,T(n/2^k) + k\,c\,n$

k goes by 1 at every step

When do we stop? (base case of the recursion)

MCS Divide and Conquer

$T(n) = 2^k\, T(n / 2^k) + k\,c\,n$

When do we stop? (base case of the recursion)

When $n / 2^k$ is equal to 1 (not 0, because n is not equal to 0, n is "big")

MCS Divide and Conquer

$$T(n) = 2^{k} T(n / 2^{k}) + k c n$$

$$n / 2^{k} = 1 \Rightarrow n = 2^{k}$$

$$\Rightarrow \log n = \log 2^{k} = k \log 2 = k$$

$$T(n) = n T(1) + c n \log n$$

MCS Divide and Conquer

$T(n) = n\,T(1) + c\,n \log n$

$T(1)$ = running time of the algorithm in the base case (list of 1 element). $T(1)$ is $\Theta(1)$

➔ $T(n)$ is $\Theta(n \log n)$

// Remember that c is a constant

MCS Divide and Conquer

$T(n) = n\,T(1) + c\,n \log n$

$T(n)$ is $\Theta(n \log n)$

We can see that it does not matter whether c is equal to 1, 2, 3, 4 or even 10 as long as c is a constant

MCS Divide and Conquer

$T(n) = 2\,T(n/2) + c\,n$ // equation (1)

- We could start from the bottom and move up

$T(2) = 2\,T(1) + 2\,c$

$T(2^2) = 2\,T(2) + 2^2\,c$

$T(2^2) = 2\,(2\,T(1) + 2\,c) + 2^2\,c$

$T(2^2) = 2^2\,T(1) + 2 * 2^2\,c$

MCS Divide and Conquer

$T(n) = 2\,T(n/2) + c\,n$ // equation (1)

$T(2^2) = 2^2\,T(1) + 2 * 2^2\,c$

$T(2^3) = 2\,T(2^2) + 2^3\,c$

$T(2^3) = 2\,(2^2\,T(1) + 2^3\,c) + 2^3\,c$

$T(2^3) = 2^3\,T(1) + 2 * 2^3\,c + 2^3\,c$

$T(2^3) = 2^3\,T(1) + 3 * 2^3\,c$

MCS Divide and Conquer

$T(n) = 2\,T(n/2) + c\,n$ // equation (1)

$T(2^3) = 2^3\,T(1) + 3 * 2^3\,c$

$T(2^4) = 2\,T(2^3) + 2^4\,c$

$T(2^4) = 2\,(2^3\,T(1) + 3 * 2^3\,c) + 2^4\,c$

$T(2^4) = 2^4\,T(1) + 4 * 2^4\,c$

MCS Divide and Conquer

$T(2) = 2\ T(1) + \ 2\ c$ OR

$T(2^{1}) = 2^{1}\ T(1) + 1 * 2^{1}\ c$

$T(2^{2}) = 2^{2}\ T(1) + 2 * 2^{2}\ c$

$T(2^{3}) = 2^{3}\ T(1) + 3 * 2^{3}\ c$

$T(2^{4}) = 2^{4}\ T(1) + 4 * 2^{4}\ c$

- Do we see a pattern?

MCS Divide and Conquer

$T(2^k) = 2^k T(1) + k * 2^k c$

- Plug in $n = 2^k$, i.e. $k = \log n$, we get

$T(n) = n T(1) + (\log n) n c$

$T(n) = n T(1) + c n \log n$

- Same formula as before

Recursive Binary Search

```
public static int binarySearch( int arr[], int key, int start, int end )
{
  if( start > end ) // empty subarray
     return -1;
   else
  {
     int middle = ( start + end ) / 2;
     if ( arr[middle] == key )
        return middle;
     else if( arr[middle] > key ) // look left
        return ???
      else // look right
        return ???
   }
}
```

Recursive Binary Search

```
public static int binarySearch( int arr[], int key, int start, int end )
{
 if( start > end ) // empty subarray
    return -1;
  else
 {
    int middle = ( start + end ) / 2;
    if ( arr[middle] == key )
       return middle;
    else if( arr[middle] > key ) // look left
       return binarySearch( arr, key, start, middle – 1 );
     else // look right
       return binarySearch( arr, key, middle + 1, end );
  }
}
```

Recursive Binary Search

$T(n) = a + ???$

- How many recursive calls are made in the general case? How much time?

Recursive Binary Search

$T(n) = a + ???$

- How many recursive calls are made in the general case? Only 1 will be made (searching left OR searching right but NOT both)

$T(n) = a + b + T(n / 2) = c + T(n / 2)$

- Where c is a positive constant

Recursive Binary Search

$T(n) = c + T(n / 2)$ // equation (1)

$T(x) = c + T(x / 2)$ // equation (A)

- Let's drill down, substituting x for n / 2 in equation (A)

$T(n / 2) = c + T(n / 2^2)$ // equation (2)

- Note: it is better to write $n / 2^2$ as opposed to $n / 4$ in order to see a pattern emerge

Recursive Binary Search

$T(n) = c + T(n/2)$ // equation (1)

$T(n/2) = c + T(n/2^2)$ // equation (2)

$T(n) = c + c + T(n/2^2)$

$T(n) = 2c + T(n/2^2)$ // equation (3)

Recursive Binary Search

$T(n) = 2c + T(n / 2^2)$ // equation (3)

$T(x) = c + T(x / 2)$ // equation (A)

- Let's drill down, substituting x for $n / 2^2$ in equation (A)

$T(n / 2^2) = c + T(n / 2^3)$ // equation (4)

- Note: it is better to write $n / 2^3$ as opposed to $n / 8$ in order to see a pattern emerge

Recursive Binary Search

$T(n) = 2c + T(n / 2^2)$ // equation (3)

$T(n / 2^2) = c + T(n / 2^3)$ // equation (4)

$T(n) = 2c + c + T(n / 2^3)$

$T(n) = 3c + T(n / 2^3)$ // equation (5)

Recursive Binary Search

$T(n) = 3c + T(n / 2^3)$ // equation (5)

$T(x) = c + T(x / 2)$ // equation (A)

- Let's drill down, substituting x for $n / 2^3$ in equation (A)

$T(n / 2^3) = c + T(n / 2^4)$ // equation (6)

- Note: it is better to write $n / 2^4$ as opposed to $n / 16$ in order to see a pattern emerge

Recursive Binary Search

$T(n) = 3c + T(n/2^3)$ // equation (5)

$T(n/2^3) = c + T(n/2^4)$ // equation (6)

$T(n) = 3c + c + T(n/2^4)$

$T(n) = 4c + T(n/2^4)$ // equation (7)

Recursive Binary Search

$T(n) = c + T(n / 2)$ // equation (1) OR

$T(n) = 1 c + T(n / 2^1)$ // equation (1b)

$T(n) = 2 c + T(n / 2^2)$ // equation (3)

$T(n) = 3 c + T(n / 2^3)$ // equation (5)

$T(n) = 4 c + T(n / 2^4)$ // equation (7)

Do we see a pattern?

Recursive Binary Search

$T(n) = k\,c + T(n / 2^k)$ // equation (8)

- When do we reach the base case (bottom of the recursion)?

Recursive Binary Search

$T(n) = k\,c + T(n / 2^k)$ // equation (8)

- When do we reach the base case (bottom of the recursion)?
- When $n / 2^k = 1$ (not 0, that would yield $n = 0$, but n is "big")

Recursive Binary Search

$T(n) = k\,c + T(n / 2^k)$ // equation (8)

- When $n / 2^k = 1$, we get
- $T(n) = k\,c + T(1)$
- And
- $n = 2^k$

Recursive Binary Search

$T(n) = k\,c + T(1)$ // equation (8)

- $n = 2^k \Rightarrow k = \log n$

$T(n) = (\log n)\,c + T(1)$

$T(n) = c \log n + T(1)$ is $\Theta(\log n)$

Recursive Binary Search

- The non-recursive binary search method was also $\Theta($ log n $)$, same for the recursive formulation