# Lecture 5

## Qn 1

```
BinarySearch(x, A, i, j)
if(j < i)return(“not present”)
mid ← (i+j )/2
if(A[mid] = x) return(“present”)
if(x < A[mid])
return(BinarySearch(x, A, i, mid − 1))
else
return(BinarySearch(x, A, mid + 1, j))
```

Find the recurrence relation for the pseudocode.

BinarySearch(x, A, i, j)

if(j < i)return("not present")

mid ← (i+j )/2 $\longrightarrow$ O(1)

if(A[mid] = x) return("present") $\longrightarrow$ O(1)

if(x < A[mid])

return(BinarySearch(x, A, i, mid − 1)) $\longrightarrow$ N/2

else

return(BinarySearch(x, A, mid + 1, j)) $\longrightarrow$ N/2

Summing up time required to perform all steps:

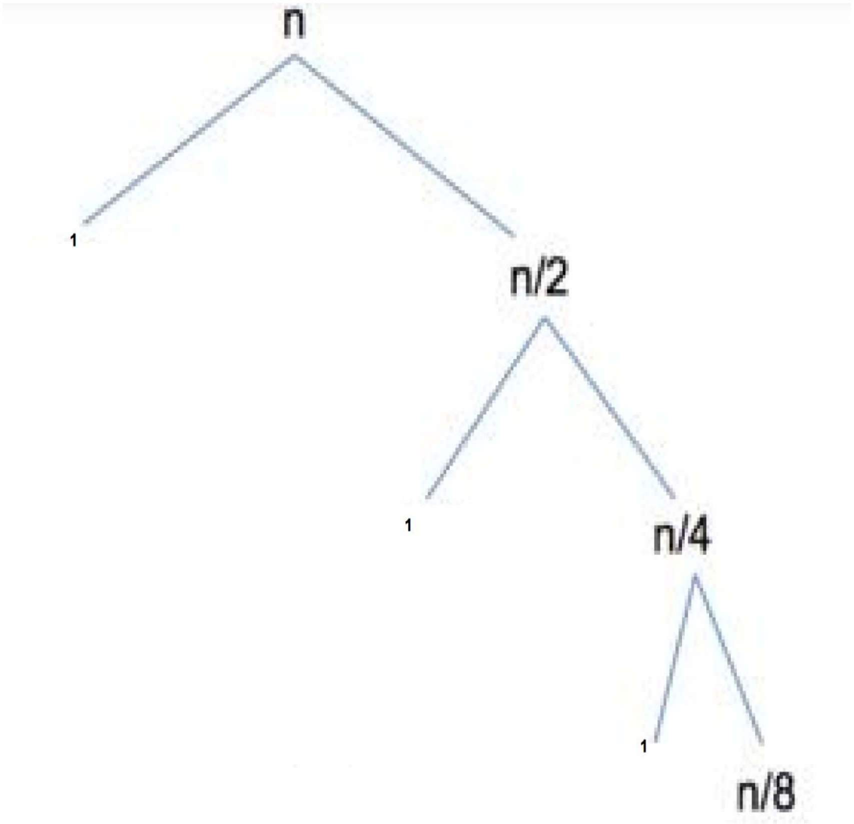
T(n)=T(n/2)+ C  n>1
and T(n)= 1 if n=1

Substitution Method

$$
\begin{aligned}
T(n) &\le T(n/2) + c \\
&\le (T(n/4) + c) + c \\
&= T(n/4) + 2c \\
&\;\;\cdots \\
&\le T(n/2^{i}) + i \cdot c \\
&\;\;\cdots \\
&\le T(1) + \log n \cdot c \\
&\le 1 + c \cdot \log n
\end{aligned}
$$

$T(n)=O(\log n)$

Recurrence Tree Method



At ith level, $T(n) = (n/2^{i}) + k$

Assume $(n/2^{i}) = 1$

$n = 2^{k}$

$k = \log n$

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

$=$**$O(\log n)$**

# Worst Case Complexity Analysis

Let $T(n)$ = worst case number of comparisons in binary search of an array of size n

From an analysis perspective, when binary search is used with an array of size $n > 1$, the sizes of the two halves are:

n even => left side $n/2$, right side $n/2 - 1$

n odd => both sides have size $(n-1)/2 = n/2$

In the worst case, we would end up consistently going to the **left** side with $n/2$ elements. Counting 1 for the middle element comparison we get this recurrence equation for the worst-case number of comparisons:

$T(1) = 1$

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

Our claim is that $T(n) = O(\log(n))$. In fact we want to prove:

$T(n) \leq 2 * \log(n), \quad n \geq 2$ Look at a comparison of some computed values of $T(n)$ and $\log(n)$

$T(2) = 2,\ \log(2) = 1$

$T(3) = 2,\ \log(3) = 1.x$

$T(4) = 3,\ \log(4) = 2$

$T(5) = 3,\ \log(5) = 2.x$

# Proof by induction

We use so-called *strong* induction. We have verified that for base cases n = 2, 3, 4, 5 that:

$T(n) \le 2 * \log(n), n \ge 2$

Assume valid up to (but not including) n. This means that we can make the **inductive assumption** and assume this to be true:

$T(n/2) \le 2 * \log(n/2), \; n/2 < n$

Then the proof goes like this:

$T(n) = 1 + T(n/2)$ (the recurrence)

$\le 1 + 2 * \log( n/2 )$ (substitute from inductive assumption)

$= 1 + 2 * (\log(n) – 1)$ (properties of log)

$= 2 * \log(n) – 1$ (simple algebra)

$\le 2 * \log(n)$ (becoming larger)

The key algebraic step relies on the property:
$\log(a/b) = \log(a) – \log(b)$
which we are using like this:
$\log(n/2) = \log(n) – \log(2) = \log(n) – 1$
However, because n/2 is **truncated** division, this last statement is not technically correct when n is odd. What is true is that
$\log(n/2) \le \log(n) – 1$

## Average comparisons lower bound

To get a lower bound, simply omit all nodes on the last level. So we compute comparisons for a total of L = flr(log(n)) levels. Using the same formula, we get

total comparisons

$\geq$ Comparisons( flr(log(n)) )

$= ( \text{flr}(\log(n)) - 1 ) * 2^{\text{flr}(\log n)} + 1$

The flr(log(n)) expression truncates the decimal part of log(n) and so it will subtract away less than 1 from log(n), i.e.,
flr(log(n)) $\geq$ log(n)–1

Replacing flr(log n) by log(n)–1 and doing the algebra, we get
total comparisons

$\geq ( \log(n) - 2 ) * 2^{\log(n) - 1} + 1$

$= \log(n) * 2^{\log(n)-1} - 2^{\log(n)} + 1$

$= \frac{1}{2} * n * \log(n) - n + 1$

Dividing this total by n gives the average, i.e.,
average comparisons

$\geq \frac{1}{2} * \log(n) - 1 + 1/n$

$> \frac{1}{2} * \log(n) - 1$

$> \frac{1}{4} * \log(n)$ (for sufficiently large n)

From this we can say:

The average number of comparisons is $\Omega(\log(n))$.

We also know that average number must be less that the worst, which we know is
worst case comparisons $\leq \log(n) + 1$

Therefore, the average number of comparisons for any n is somewhere between:
$\frac{1}{2}*\log(n) - 1$ and **1**$*\log(n) + 1$

# Qn 2

```
MergeSort(A,l,r)
if l < r then
 m := (⌈(l+r)/2⌉ )
MergeSort(A,l,m)
MergeSort(A,m+1,r)
Merge(A,l,m,r)
```

Find the recurrence relation for the pseudocode.

```
MergeSort(A,l,r)
if l < r then
 m := (⌈(l+r)/2⌉ )          O(1)
MergeSort(A,l,m)            N/2
MergeSort(A,m+1,r)          N/2
Merge(A,l,m,r)              Θ(N)
```

Summing up time required to perform all steps:

$T(n)=2T(n/2)+ \Theta(N)$  $n>1$

and $T(n)= \Theta(1)$ if $n=1$

```
MergeSort(A,l,r)
if l < r then
 m := (⌈(l+r)/2⌉ )
MergeSort(A,l,m)
MergeSort(A,m+1,r)
Merge(A,l,m,r)
```

O(1)

Divide

N/2

Conquer

N/2

Θ(N)

Combine

Summing up time required to perform all steps:

**$T(n)=2T(n/2)+ \Theta(n)$  $n>1$**
**and $T(n)= \Theta(1)$ if $n=1$**

Solving by Substitution:

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

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

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

$= 2^2(2T(n/8) + n/4) + 2n$

$= 2^3T(n/8) + 3n$

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

let us assume that n is a power of 2, i.e $n = 2^k$ for some k.

$= 2^{\log n}T(n/n) + n \log n$

$= n + n \log n$

Recurrence Tree Method: