Recurrence Relation

T(n) = aT (n/b) + *f*(n) divide+Conquer

T(n)=1.T(n/2)+1 Binary search

T(n)=2.T(n/2)+n Merge-Sort

Let's map this recurrence on the recurrence for Merge Sort as an example: **T(n) = 2T(n/2) + n**.

- **n -** The size of the problem. For Merge Sort for example, **n** would be the length of the list being sorted.
- **a -** The number of sub-problems in each recursive step. So in our Merge Sort example, since we are dividing the array into two halves and recurring down each half, **a = 2**.
- **b -** The amount we're reducing the sub-problems by. For Merge Sort **b = 2** because we are passing half of the array (length **n**) to each sub-problem. **n/b** is the total size of each sub-problem.
- ***f*(n) -** The work to be done on **n** outside of the recursive steps. For Merge Sort this represents the merging step for the results of the recursion.

Binary Search (Iterative Approach)

```
While(low<high)
{       mid=low+(high-low)/2
        If(arr[mid]==x)
        {       printf("\n Found at location mid",mid);
                Break;
                }
```

Example Recurrences

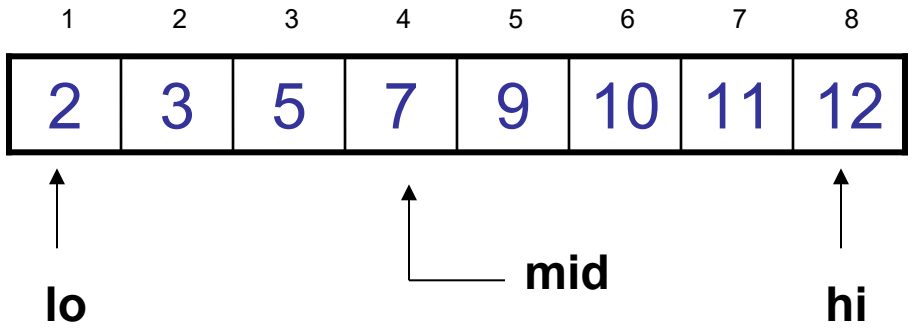
- $T(n) = T(n-1) + n$ $\Theta(n^2)$
  - Recursive algorithm that loops through the input to eliminate one item
- $T(n) = T(n/2) + c$ $\Theta(\lg n)$
  - Recursive algorithm that halves the input in one step
- $T(n) = T(n/2) + n$ $\Theta(n)$
  - Recursive algorithm that halves the input but must examine every item in the input
- $T(n) = 2T(n/2) + 1$ $\Theta(n)$
  - Recursive algorithm that splits the input into 2 halves and does a constant amount of other work

Recurrent Algorithms
BINARY-SEARCH

- for an ordered array A, finds if x is in the array A[lo…hi]

Alg.: BINARY-SEARCH (A, lo, hi, x)

```
if (lo > hi)
    return FALSE
mid ← ⌊(lo+hi)/2⌋
if x = A[mid]
    return TRUE
if ( x < A[mid] )
    BINARY-SEARCH (A, lo, mid-1, x)
if ( x > A[mid] )
    BINARY-SEARCH (A, mid+1, hi, x)
```



Analysis of BINARY-SEARCH

Alg.: BINARY-SEARCH (A, lo, hi, x)

```
if (lo > hi)
    return FALSE
mid ← ⌊(lo+hi)/2⌋
if x = A[mid]
    return TRUE
if ( x < A[mid] )
    BINARY-SEARCH (A, lo, mid-1, x)
if ( x > A[mid] )
    BINARY-SEARCH (A, mid+1, hi, x)
```

← constant time: c_1

← constant time: c_2

← constant time: c_3

← same problem of size n/2

← same problem of size n/2

- $T(n) = 1 + T(n/2)$
  - T(n) – running time for an array of size n

Ternary Search

```
int ternarySearch(int array[], int start, int end, int
key)
{       if(start <= end)
                {int midFirst = (start + (end - start) /3);
                int midSecond = (midFirst + (end -
                        start) /3);
                if(array[midFirst] == key)
                        return midFirst;
                if(array[midSecond] == key)
                        return midSecond;
```

```
            if(key < array[midFirst])
                return ternarySearch(array, start,
                midFirst-1, key);
            if(key > array[midSecond])
                return ternarySearch(array,
                midSecond+1, end, key);
        return ternarySearch(array, midFirst+1,
midSecond-1, key);
 }
}
```

Recurrence relation

- T(n)=T(n/3)+T(2n/3)+n
- T(n)=3T(n/3)+n=T(n/3)+T(n/3)+T(n/3)+n
- O(n/n2/n3/n4…)=>F(n)=2n=>o(n2)=>Omega(n)

- T(n)=2.T(n/2)+n
- T(n)=T(n/2)+1…

Methods for Solving Recurrences

- Master Method : Cook-Book method
- Substitution Method
- Recursion tree Method

Master Method

Recurrence relations that can be solved by the Master Theorem fall into three cases describing where the bulk of the time complexity cost lies for the recurrence. These cases are:

$$T(n) = aT(n/b) + f(n)$$

- Work performed in the sub-problems (**aT(n/b)** portion) has the greatest impact on overall time complexity.
- Work performed in the sub-problems has about the same level of impact as work performed in the dividing/combining steps (***f*(n)** portion)
- Work performed in the sub-problems has a lower impact on overall time complexity than the work performed in the dividing/combining steps.

- **T(n) = aT (n/b) + *f*(n)=divide+conquer**
- 1. find the value of a,b, f(n) from the given recurrence relation.
- 2. Calculate $\mathbf{n^{\log_b(a)}}$
- **3. compare this with f(n)**

Master Method

To figure out which case applies you'll need to compare the size of the sub-problems (i.e. **aT(n/b)**) with the size of the work performed outside (***f*(n)**). This leaves us with the following comparison:

T(n) = aT (n/b) + *f*(n), T(n) = 2T (n/2) + *n(f(n)=n,*

$$n^{\log_b(a)} <=> f(n),\ n^{\log_2(2)} = n1 = n$$

Divison=n, Conquer=n,So let's rethink our three cases in these terms:

1. $n^{\log_b(a)} > f(n)$,; $n^{\log_b(a)-\varepsilon} = f(n)$,; then $T(n) = \Theta(n^{\log_b(a)})$
2. $n^{\log_b(a)} == f(n)$; then $T(n) = \Theta(n^{\log_b(a)} * \lg(n))$
3. $n^{\log_b(a)} < f(n)$, $n^{\log_b(a)+\varepsilon} = f(n)$ **and** af(n/b) ≤ cf(n) for some 0<c < 1 and all sufficiently large n then $T(n) = \Theta(f(n))$ **here** $\varepsilon > 0$.

Example of Master Method

Now let's work through a concrete example using the Merge Sort recurrence.

Example 1. Master Theorem Merge Sort Example

Recurrence relation:

$$\mathbf{T(n) = 2T(n/2) + \mathit{n}}$$

Variables:

$$\mathbf{a = 2, b = 2 , \mathit{f}(n) = \mathit{n}}$$

Comparison:

$\mathbf{n^{\log_b(a)} <=> n}$

$\mathbf{n^1 == n}$

- Here we see that the cost of *f*(n) and the sub-problems are the same, so this is **Case 2**:

$\mathbf{T(n) = \Theta(n^{\log_b(a)} \lg n) = \Theta(n \lg n)}$

Binary Search

T(n) = T(n/2) + *1*

Variables:

a = 1, b = 2 , *f*(n) = *1*

Comparison:

$\mathbf{n^{\log_b(a)} <=> 1}$

$\mathbf{n^{\log_2(1)} = n^{\log} = 1}$

1 == 1=>n0==n0

- Here we see that the cost of *f*(n) and the sub-problems are the same, so this is **Case 2**:
  $\mathbf{T(n) = \Theta(n^{\log_b(a)} \lg n) = \Theta(1.\lg n)}$

Example 2: $T(n) = 9T(n/3)+n^3$.Here a = 9, b = 3, $f(n) = n^3$

$$\text{Log}_3\ 9 \Rightarrow \text{Log}_3\ 32 \Rightarrow 2\log 3$$

Log ab

=>b Log a

$$\mathbf{n^{\log_b(a)} = n^{\log_3 9} = n^{\log_3 3\text{^}2} = n^{2\log_3 3} = n^{2*1}} = (n^2)$$

$$\mathbf{N^{\log_3 9} < f(n)}$$

Since $f(n) = (\mathbf{n^{\log_3 9 + \varepsilon}})$ for $\varepsilon > 0$, case 3 of the master theorem applies.

The solution is

$$T(n) = \Theta(\mathbf{f(n)}) = \Theta(n^3)$$

Example 3: $T(n) = T(2n/3)+1$.

Here $a = 1$, $b = 3/2$, $f(n) = 1$

$$\mathbf{n}^{\log_b(a)} = \mathbf{n}^{\log_{3/2}1} = n^0 = 1$$

Since $f(n) = \mathbf{n}^{\log_b(a)}$, case 2 of the master theorem applies, so the solution is :

$$T(n) = \Theta(\mathbf{n}^{\log_b(a)} \log n) = \Theta(\log n).$$

Exercise

- 1. $T(n) = 9T(n/3)+n^2$
- 2. $T(n) = 9T(n/3)+n$
- 3. $T(n) = 16T(n/4)+n^2$
- 4 $T(n) = 16T(n/4)+n^3$
- 5. $T(n) = 16T(n/4)+n$
- Floor(n/2)+17<=n/2 ---(1)
- Floor(n/2)+17>n/2 ------(2)

- Floor(n/2)<=n/2
- Ceiling(n/2)>=n/2
- T(n)=floor(n/2)

Assignment

- T(n)=T(floor(n/2))+1
- T(n)=T(Ceil(n/2))+1
- T(n)=2T(floor(n/2)+5)+n
- T(n)=2T(Ceil(n/2)-5)+n

- $T(n)=T(floor(n/2))+1$
- $T(n)=T(n/2)+1$
- $T(n)=lgn$
- $T(n)=lg(floor(n/2))+1$
- $T(n)<=lg(n/2)+1$
- $T(n)<=lg(n)-lg(2)+1$
- $T(n)<=lg(n)-1+1$
- $T(n)<=lg(n)$
- $T(n)=O(lg\ n)$

- T(n)=T(ceiling(n/2))+1
- T(n)=T(n/2)+1
- T(n)=lgn
- T(n)=lg(ceiling(n/2))+1
- T(n)>= lg(n/2)+1
- T(n) >=lg(n)-lg(2)+1
- T(n)>= lg(n)-1+1
- T(n)>= lg(n)
- T(n)=omega(lgn)

- T(n)=2T((Floor(n/2))+17)+n
- T(n)=theta(nlgn)=cnlgn where c=1
- T(n)=2*(floor(n/2)+17) lg(floor(n/2)+17)+n
- T(n)>2*(n/2)lg(n/2)+n
- T(n)>nlg(n/2)+n
- T(n)>nlg(n)-n+n
- T(n)>nlgn
- T(n)=Omega(nlgn)

Recurrence tree method

- T(n)=3T(n/4)+n2
- T(n)=Theeta(n2)
- T(n)=divison complexity+conquer complexity
- T(n)= $\mathbf{n^{\log_b(a)}}$

- $T(n)=3T(floor(n/4))+n^2$
- T(n)=O(n2)

Recurrence Tree Method Examples

- T(n)=T(1)+T(n-1)+n= T(@)+T(n-@)+n where @<<<<n
- T(n)=T(n/2)+1
- T(n)=2T(n/2)+n

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

- n
- n/2 n/2 =>n
- n/4 n/4 n/4 n/4 =>n
- ……

Height of the binary tree=lg2n

$T(n)=n(lg2n -1)+n^{lg}{}_{b}{}^{a}$

T(n)=n(lg2n -1)+n

T(n)=n(lg2n -1)+n=nlg2n-n+n=nlg2n

T(n)=Theeta (nlg2n)

T(n)=T(n/2)+1

- n
- n/2 n/2=>1
- …….. =>1
- =>1
- Division cost=$n^{lg}{}_{b}{}^{a}$ =1
- Conquer cost=1+1+1+…..=1.(lg2n-1)
- T(n)=Lg2n-1+1=lg2n=theta(lg2n)
-

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

- n
- 1 n-1 =>n
- 1 n-2 =>n-1
- 1 n-3 =>n-2
- ……..
- a=1,b=1, division cost=n
- T(n)=n+n-1+n-2+……+n=O(n^2)+n
- T(n)=O(n^2)

T(n)=T(n/3)+T(2n/3)+n

- n
- n/3 2n/3 =>n
- n/9 2n/9 2n/9 4n/9..=>n
- ……..
- Height=$lg_{3/2}$ n
- T(n)=conquer+division
- T(n)=n($lg_{3/2}$ n-1) +n^$log_3$1+ n^$log_{3/2}$1
- T(n)=n($lg_{3/2}$ n-1) +1+1
- T(n)= n$lg_{3/2}$ n-n +1+1= n$lg_{3/2}$ n-O(n)
- T(n)=O(n$lg_{3/2}$ n)

Substitution method

- $T(n)=2T(\sqrt{n})+\lg n$
- $n=2^m$ taking log both side
- $\lg n = m\lg 2 = m$
- $T(2^m)=2T(2^{m/2})+m$
- $Y=x+5$, $f(y)=>f(x+5)=>z(x)$ where, $f(x+5)=z(x)$
- $S(m)=2S(m/2)+m$
- Apply master's
- $S(m)=\text{Theeta}(m\lg m)$
- Place value
- $T(n)=\text{theta}(\lg n \lg\lg n)$

- $T(n)=T(\sqrt{n})+1$
- $n=2^m$ taking log both side
- Lg n =m
- $T(2^m)=T(2^{m/2})+1$
- $S(m)=S(m/2)+1$
- Apply master's
- $S(m)=Theeta(lgm)$
- Place value
- $T(n)=theta(lglgn)$
- $T(n)= T(\sqrt{n})+n \Rightarrow Theeta(lgn)$

$T(n)= \sqrt{n}\, T(\sqrt{n})+n$

- $n=2^m$ taking log both side
- Lg n =m
- $T(2^m)= 2^{m/2}\, T(2^{m/2})+m$
- $S(m)=m/2S(m/2)+m=m/2(S(m/2)+2)$
- $S(m)=Theeta\ (m/2lgm)=O(mlgm)$
- $T(n)=O(lgnlglgn)$

- $T(n)= T(\sqrt{n})+n$
- $n=2^m$ taking log both side
- Lg n =m
- $T(2^m)=T(2^{m/2})+ 2^m$
- S(m)=S(m/2)+m
- Apply master's (case 3)
- S(m)=Theeta(m)
- Place value
- T(n)=theta(lgn)

- $T(n)= \sqrt{n}\, T(\sqrt{n})+n$
- $n=2^m$ taking log both side
- Lg n =m
- $T(2^m)= 2^{m/2}\, T(2^{m/2})+ 2^m$
- S(m)=m/2S(m/2)+m=>m/2(S(m/2)+2)=>m/2(lg(m)=O(mlgm)
- Place value
- T(n)=theta(lgnlglgn)

Counting sort

- A[8]={4,1,45,79,21,56,34,90}=>8
- Min.=1, max.=90
- Count array[1-90]=>90

Greedy Method Examples

- 3. Schedule the following jobs by greedy approach and find total profit.(t<deadline)
- {j4,j1,j2,j3}=>t=0=>t=1=>t=2
- Profit=45+34

Jobs	Profit	Deadline
J1	40	1
J2	34	3
J3	23	2
J4	45	2

- M=30, W={15,10,8,7}, P={35,20,14,13} solve knapsack problem (0/1 and fractional)
- 0/1 55/45
- Fractional-64.28
- p/w={2,33,2,1.75,1.855..}=>P1,p2,p4,p3
- p/w={2.33,2,1.855,1.75.}=>
- 0/1 knapsack profit=35+20=55
- Fractional=35+20+1.855*5=55+9.275=64.275

- . M=40, W={15,20,8,7,10}, P={35,20,14,13,12} solve knapsack problem (0/1 and fractional)
- P/w={2.3,1,1.75,1.85,1.2}=>{p1,p4,p3,p5,p2}
- 0/1 knapsack=35(25)+13(18)+14(10)+12(0)=74
- fractional

- Schedule the following jobs by greedy approach and find total profit.