

Algorithms Design Methods:

Divide and Conquer Method:

To solve a large problem using divide and conquer approach, it most divided it into number of smaller problems and solve each part, then combine these sub solutions to obtain the solution of the original problem. Often, the subproblems that generated are, simply, smaller instances of the original problem and may be solved using divide and conquer strategy recursively.

The divide and conquer strategy involves the following steps:

1. Divide an instance of a problem into two or more smaller instances.
2. Solve each of the smaller instances.
3. Combine (if necessary) the solutions to the smaller instances to obtain the solution to the original instance.

In some problems, there is no need to the step 3 above, such as in binary search problem, that because the solution is find the element that we at look for it. So that there is no need to combine the solutions.

The control abstraction for the divide and conquer method as following:

DandC (P)

Begin

if small (P) return S(P)

else

divide P into smaller instances P_1, P_2, \dots, P_k , $k > 1$

apply DandC to each of these subproblems

return combine(DandC(P_1), DandC(P_2), ..., DandC(P_k))

Endif

End

Time complexities of Divide and Conquer method

If the size of the problem p is n and the sizes of subproblems P_1, P_2, \dots, P_k are n_1, n_2, \dots, n_k respectively, then the time complexities for the divide and conquer strategy is described as follow:

$$T(n) = \begin{cases} g(n) & \text{if } n \text{ small} \\ T(n_1) + T(n_2) + T(n_3) + \dots + T(n_k) + F(n) & \text{otherwise} \end{cases}$$

Where:

$T(n)$: is the execution time of DandC for any inputs with size n .

$g(n)$: the computing time of solving the small problem

$F(n)$: the time of dividing and combining the problem P to its subproblems.

Binary Search

One of the problem s that solved using Divide and conquer strategy is Binary search problem, that locate for element k in a sorted list $a[1..n]$.

The idea: to search about the element k in the $a[\text{low}..\text{high}]$ list, we search about k in three sublist $a[\text{low}..\text{mid}-1]$, $a[\text{mid}..\text{mid}]$, $a[\text{mid}+1..\text{high}]$. by comparing k with $a[\text{mid}]$ two of the sublist will be removed.

This can be solved using divide and conquer method as in the following algorithm:

BinaySearch (a(), k, low, high)

Begin

if (low > high) then return 0

else

mid= (low+high)/2

```

    if ( k = a(mid)) then return mid
    else if ( k < a(mid)) then
        BinarySreach(a,k,low,mid-1)
    else BinarySearch(a,k,mid+1,high)
    endif
endif
endif
End

```

Algorithm analysis:

1. Space complexities: each activation for the function require seven words that a, low, high, k, mid, name of function and return address.

In 1st activation $n+1/2^1$, 2nd activation $n+1/2^2$, ..., mth activation $n+1/2^m$

The last comparison is stopped when $n+1=2^m$

$\log(n+1)=\log 2^m$, $m=\log(n+1)$

Therefore, $S_{\text{BinarySearch}}(n)=\Theta(\log n)$

2. Time complexities: The time complexities for this algorithm can be described as following:

$$T_{\text{BinarySearch}}(n)=\begin{cases} T_{\text{BinarySearch}}(1) & \text{if } n = 1 \\ T_{\text{BinarySearch}}(n/2) + c & \text{otherwise} \end{cases}$$

$$\begin{aligned}
 T_{\text{BinarySearch}}(n) &= T_{\text{BinarySearch}}(n/2)+c \\
 &= T_{\text{BinarySearch}}(n/2^2)+c+c \\
 &= T_{\text{BinarySearch}}(n/2^3)+3c \\
 &= T_{\text{BinarySearch}}(n/2^m)+mc
 \end{aligned}$$

Suppose $n = 2^m$, $m = \log_2 n$

$$= T_{\text{BinarySearch}}(1) + c \log_2 n$$

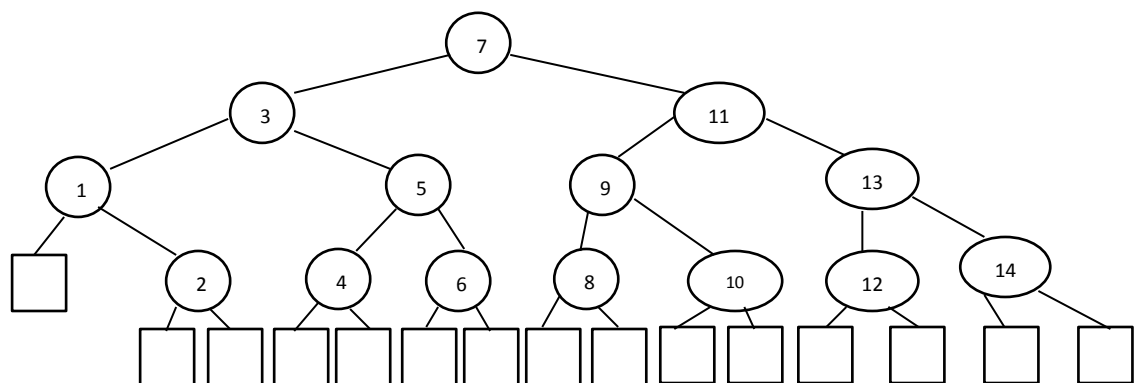
$$T_{\text{BinarySearch}}(n) = \Theta(\log_2 n)$$

This the worst and average cases time complexities for the successful search of the binary search algorithm. While the best case is equal to $\Theta(1)$ (Why?).

For example, the binary search decision tree when $n=14$ is described as following where :

○ Internal nodes (represent successful states)=14=n

□ External node (represent fail states)=15=n+1



Binary search decision tree when $n=14$

Notes:

- The maximum number of comparisons for the successful search = 4 comparisons, Ex: $7 \rightarrow 11 \rightarrow 13 \rightarrow 12$
- The minimum number of comparisons for the successful search = 1 comparisons

H.W:

- Write algorithm to find the maximum number in one dimension array using Divide and Conquer strategy?
- Rewrite Binary search algorithm without recursion(using while)?
- Design the Ternary Search algorithm, then find the time complexities for it? (this algorithm divide the array into five sublists using two divided positions as below:

