Recursive solution for Towers of Hanoi

*Let us Assume **src='A', dest='B'** and **aux='C'***

```
Algorithm TowersOfHanoi(n, src, dest, aux)
// Move n disks from src Pole  to dest Pole
{
          if (n>=1) then
          {
              TowersOfHanoi(n-1,src,aux,dest);
              write(" move top disk from pole",src,"to top of pole",dest);
              TowersOfHanoi(n-1,aux,dest,src);
          }
}
```

Binary Search

- Input list must be sorted order.
- For every iteration, it reduces the list to half.
- For iteration 1 , size of list =n

  For iteration 2, size of list =n/2

  For iteration 3, size of list = n/4

  …………..
- **Time complexity**

  Average Case – O(log n) -> conduct m trails and take the average

  Worst Case – O (log n)

```
Algorithm BinarySearch(a,low,high,ele)
// a is an array, low is lower index of the array
// high is upper index of the array, ele is the element to
//search
{
        if  low > high then
                return -1;
        else {
                mid = (low+high)/2;
                if ele==a[mid] then
                        return mid;
                else if ele < a[mid] then
                        return BinarySearch(a,low,mid-1,ele);
                else
                         return BinarySearch(a,mid+1,high,ele);
}
```

Examples

What is the time complexity of the following code?

Example 1:

```
void function(int n) {
            int i=1,s=1;
            while (s<=n) {
                i++;
                s=s+i;
                printf("*");
            }
}
```

Example 2:

```
void function(int n) {
    int i , count=0;
    for(i=1;i*i<=n; i++)
       count++;
}
```

Example 3:

```
void function(int n){
            for(int i=1;i<=n;i++) {
              for(int j=1;j<=n;j+=i)
              printf("*");
            }
}
```

Example 4:

```
void function(int n) {
        int i, j, k, count=0;
        for(i=n/2;i<=n; i++)
        {
            for(j=1;j+n/2 <=n; j++)
            {
                for(k=1;k<=n; k=k*2) {
                    count++;
                }
            }
        }
}
```

Example 5:

```
void function(int n) {
        int i, j, k, count=0;
        for(i=n/2;i<=n; i++)
        {
            for(j=1;j <=n; j=j*2)
            {
                for(k=1;k<=n; k=k*2) {
                    count++;
                }
            }
        }
}
```

Gate Questions

Q.6 Which one of the following is the tightest upper bound that represents the number of swaps required to sort n numbers using selection sort?

(A) $O(\log n)$ (B) $O(n)$ (C) $O(n \log n)$ (D) $O(n^2)$

Q.7 Which one of the following is the tightest upper bound that represents the time complexity of inserting an object into a binary search tree of n nodes?

(A) $O(1)$ (B) $O(\log n)$ (C) $O(n)$ (D) $O(n \log n)$

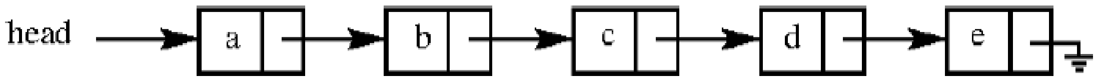
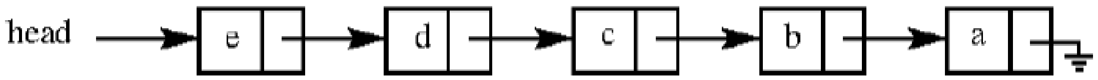
Q.31 Consider the following function:

```
int unknown(int n){
        int i, j, k=0;
        for (i=n/2; i<=n; i++)
            for (j=2; j<=n; j=j*2)
                k = k + n/2;
        return (k);
}
```

The return value of the function is

(A) $\Theta(n^2)$ (B) $\Theta(n^2 \log n)$ (C) $\Theta(n^3)$ (D) $\Theta(n^3 \log n)$

Q.11	Which one of the following statements is TRUE for all positive functions $f(n)$?
(A)	$f(n^2) = \theta(f(n)^2)$, when $f(n)$ is a polynomial
(B)	$f(n^2) = o(f(n)^2)$
(C)	$f(n^2) = O(f(n)^2)$, when $f(n)$ is an exponential function
(D)	$f(n^2) = \Omega(f(n)^2)$

Q.15	Consider the problem of reversing a singly linked list. To take an example, given the linked list below,  the reversed linked list should look like  Which one of the following statements is TRUE about the time complexity of algorithms that solve the above problem in $O(1)$ space?
(A)	The best algorithm for the problem takes $\theta(n)$ time in the worst case.
(B)	The best algorithm for the problem takes $\theta(n \log n)$ time in the worst case.
(C)	The best algorithm for the problem takes $\theta(n^2)$ time in the worst case.
(D)	It is not possible to reverse a singly linked list in $O(1)$ space.

Q.46	Let A be a priority queue for maintaining a set of elements. Suppose A is implemented using a max-heap data structure. The operation $\textsc{Extract-Max}(A)$ extracts and deletes the maximum element from A. The operation $\textsc{Insert}(A, key)$ inserts a new element key in A. The properties of a max-heap are preserved at the end of each of these operations. When A contains n elements, which one of the following statements about the worst case running time of these two operations is TRUE?
(A)	Both $\textsc{Extract-Max}(A)$ and $\textsc{Insert}(A, key)$ run in $O(1)$.
(B)	Both $\textsc{Extract-Max}(A)$ and $\textsc{Insert}(A, key)$ run in $O(\log(n))$.
(C)	$\textsc{Extract-Max}(A)$ runs in $O(1)$ whereas $\textsc{Insert}(A, key)$ runs in $O(n)$.
(D)	$\textsc{Extract-Max}(A)$ runs in $O(1)$ whereas $\textsc{Insert}(A, key)$ runs in $O(\log(n))$.

Q.54	Consider functions **Function_1** and **Function_2** expressed in pseudocode as follows:
	
	Function_1
	while $n > 1$ do
	for $i = 1$ `to` n do
	$x = x + 1$;
	end for
	$n = \lfloor n/2 \rfloor$;
	end while
	
	Function_2
	for $i = 1$ `to` $100 * n$ do
	$x = x + 1$;
	end for
	
	Let $f_1(n)$ and $f_2(n)$ denote the number of times the statement "$x = x + 1$" is executed in **Function_1** and **Function_2**, respectively.
	
	Which of the following statements is/are TRUE?

(A)	$f_1(n) \in \Theta(f_2(n))$
(B)	$f_1(n) \in o(f_2(n))$
(C)	$f_1(n) \in \omega(f_2(n))$
(D)	$f_1(n) \in O(n)$

Consider the following functions

$f(n) = 3\, n^{(\sqrt{n})}$

$g(n) = 2^{(\sqrt{n} \log n)}$

$h(n) = n!$

Which of the following is true?

(A) h(n) is O(f(n))

(B) h(n) is O(g(n))

(C) g(n) is not O(f(n))

(D) f(n) is O(g(n))

Consider the following functions

I. $(n + k)^m = \Theta(n^m)$

II. $2^{(n+1)} = O(2^n)$

III. $2^{(2n+1)} = O(2^n)$

Which of the following is correct?

a) I and II

b) I and III

c) II and III

d) I, II, and III