Performance Categories of Algorithms

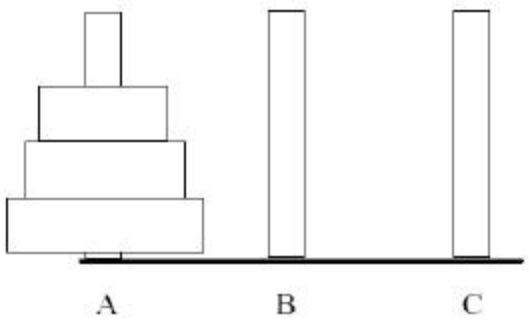
- Sublinear - O(logN)
- Linear - O(N)
- Nearly linear - O(NlogN)
- Quadratic - $O(N^2)$
- Exponential - $O(2^N)$

  $O(N!)$

  $O(N^N)$

An Exponential Algorithm

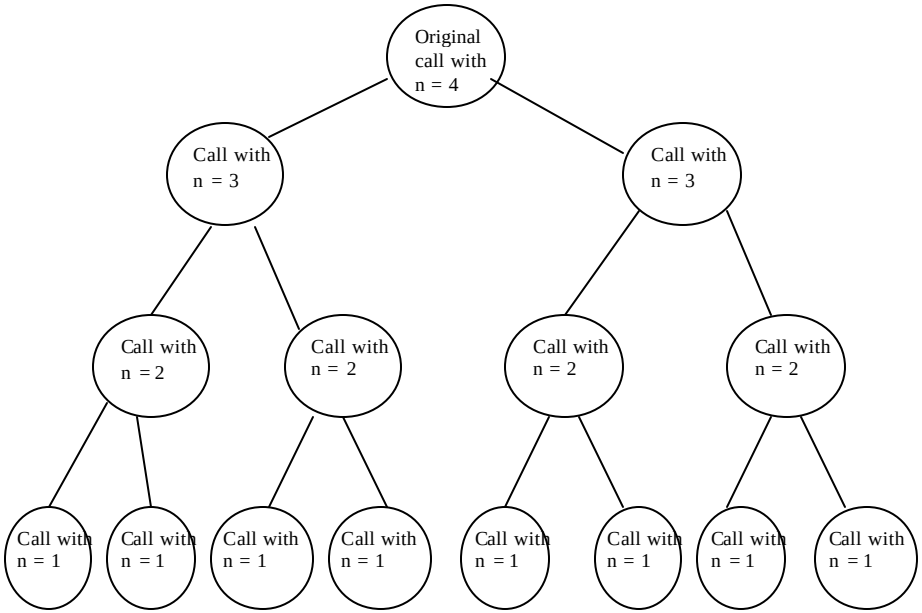
Towers of Hanoi Problem: involves moving a specified number of disks (N) that are all different sizes from one tower to another.

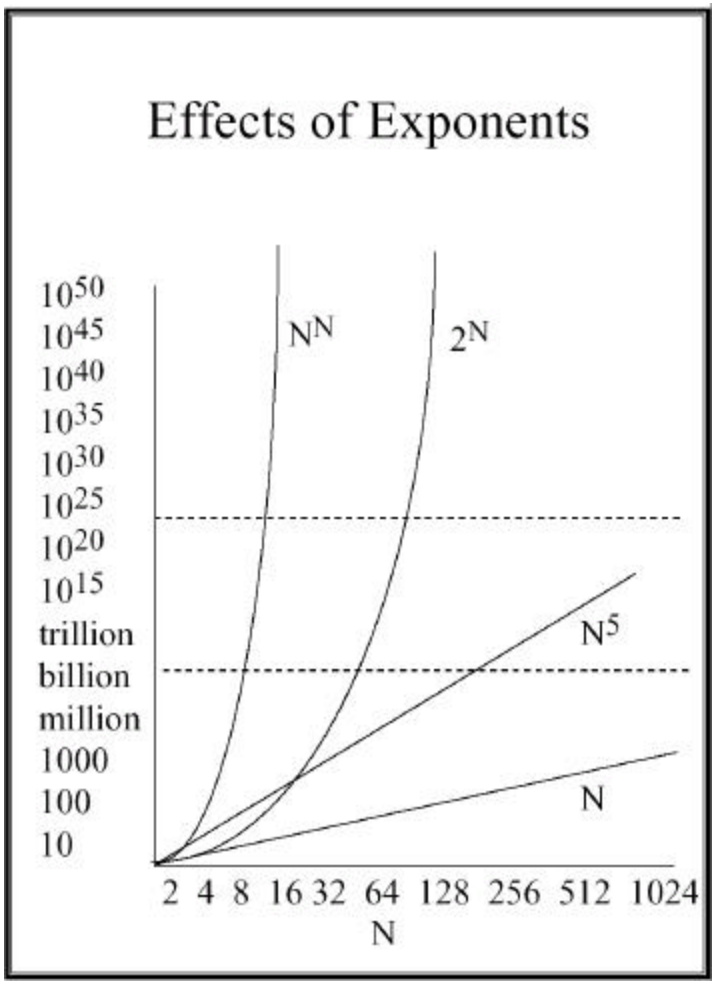



```
void tower(int n, char start, char finish, char temp)
{
  if (n == 1)
    printf("Move from %c to %c\n", start, finish);
  else {
    tower(n-1, start, temp, finish);
    printf("Move from %c to %c \n", start, finish);
    tower(n-1, temp, finish, start);
  }
}
```

Towers of Hanoi – Runtime

- For 3 rings - 7 operations
  - Cost = $(2^N) - 1$
- Each time we increment N by 1, we *double* the amount of work
- Run-time trace of function tower with 4 disks:





Effects of Exponents

Consider Towers of Hanoi (or other 2^N algorithm) for N of only 256:

Time cost is a number with *78 digits* to the left of the decimal.

For comparison:

- Number of microseconds since the Big Bang: a number with 24 digits.
- Number of protons (est'd) in the known universe: a number with 77 digits.

Reasonable vs Unreasonable

Reasonable Algorithms…

- Have N only as a *polynomial* factor
  - O (logN)
  - O (N)
  - O (N^K) where K is a constant

Unreasonable Algorithms…

- Have N as an *exponential* factor
- O (2^N)
- O (N!)
- O (N^N)

Example Problems

1. Algorithm A runs in $O(N^2)$ time, and for an input size of 4, the algorithm runs in 10 milliseconds, how long can you expect it to take to run on an input size of 16?

2. Algorithm A runs in $O(\log_2 N)$ time, and for an input size of 16, the algorithm runs in 28 milliseconds, how long can you expect it to take to run on an input size of 64?

3. Algorithm A runs in $O(N^3)$ time. For an input size of 10, the algorithm runs in 7 milliseconds. For another input size, the algorithm takes 189 milliseconds. What was that input size?

4. For an $O(N^k)$ algorithm, where k is a positive rational number, a friend tells you that instance of size M took 16 seconds to run. You run an instance of size 4M and find that it takes 256 seconds to run. What is the value of k?

5. Algorithm A runs in $O(N^3)$ time and Algorithm B solves the same problem in $O(N^2)$ time. If algorithm A takes 5 milliseconds to complete for an input size of 10, and algorithm B takes 20 milliseconds for an input size of 10, what is the input size that you expect the two algorithms to perform about the same?

6. For an $O(N^3)$ algorithm, an instance with N = 512 takes 56 milliseconds. If you used a different-sized data instance and it took 7 milliseconds how large must that instance be?

Answers

1. Algorithm A runs in $O(N^2)$ time, and for an input size of 4, the algorithm runs in 10 milliseconds, how long can you expect it to take to run on an input size of 16?

$$\frac{4^2}{10ms} = \frac{16^2}{x} \Rightarrow x = 160ms$$

2. Algorithm A runs in $O(\log_2 N)$ time, and for an input size of 16, the algorithm runs in 28 milliseconds, how long can you expect it to take to run on an input size of 64?

$$\frac{\log 16}{28\ ms} = \frac{\log 64}{x} \Rightarrow x = 42\ ms$$

3. Algorithm A runs in $O(N^3)$ time. For an input size of 10, the algorithm runs in 7 milliseconds. For another input size, the algorithm takes 189 milliseconds. What was that input size?

$$\frac{10^3}{7\ ms} = \frac{N^3}{189} \Rightarrow N = 30$$

4. For an $O(N^k)$ algorithm, where k is a positive rational number, a friend tells you that instance of size M took 16 seconds to run. You run an instance of size 4M and find that it takes 256 seconds to run. What is the value of k?

$$\frac{M^k}{16\text{ ms}} = \frac{(4M)^k}{256} \Rightarrow 4^k = \frac{256}{16} \Rightarrow k = 2$$

5. Algorithm A runs in $O(N^3)$ time and Algorithm B solves the same problem in $O(N^2)$ time. If algorithm A takes 5 milliseconds to complete for an input size of 10, and algorithm B takes 20 milliseconds for an input size of 10, what is the input size that you expect the two algorithms to perform about the same?

For algorithm A: $O(N^3)$ means

execution time $\leq c1 * N^3$

for N = 10 execution time is 5ms $= c1 * 10^3$

so , $c1 = 5 / 10^3$

For algorithm B: $O(N^2)$ means

execution time $\leq c2 * N^3$

for N = 10 execution time is 20 ms $= c2 * 10^2$

so , $c2 = 20 / 10^2$

what is N for which

$c1 * N^3 = c2 * N^2$

Substitute for c1 and c2:

$5 / 10^3 * N^3 = 20 / 10^2 * N^2$

$\Rightarrow N = 40$

6. For an $O(N^3)$ algorithm, an instance with N = 512 takes 56 milliseconds. If you used a different-sized data instance and it took 7 milliseconds how large must that instance be?

$$\frac{512^3}{56\text{ms}} = \frac{N^3}{7} \Rightarrow N = 256\text{ms}$$

7. What is the computational complexity of the following algorithm?

```
int N,j,k,sum = 0;

scanf("%d",&N);

for(j=0; j < N; j++){

   for(k=0; k < j; k++) {

      sum = sum + j * k;

   }
}
```

8. What is the computational complexity of the following algorithm? (Assume N is a positive integer)

```
int N,j,k,sum = 0;

scanf("%d",&N);

j = N;

while (j > 1) {

   k = 0;

   while (k > N){

      sum = sum + j * k;
      k ++;

   }
   j = j / 2;
}
```


Binary Trees

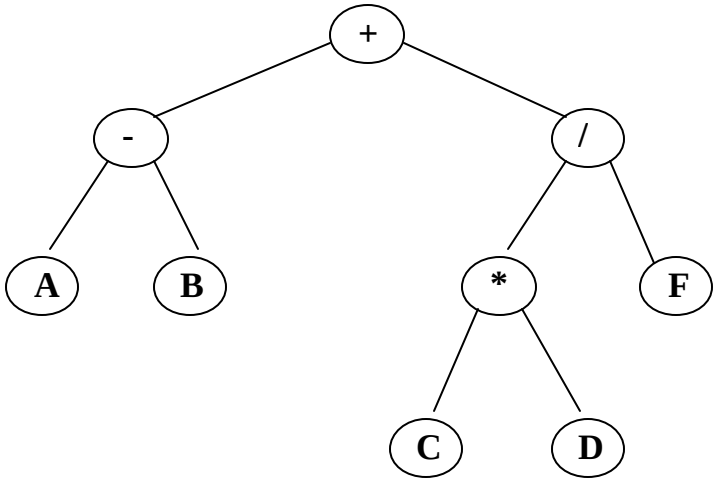
Given the following postorder and inorder traversals of a binary tree, draw the tree.

Postorder: ABCDEFIKJGH
Inorder: CBAEDFHIGKJ

An Application of Binary Trees

Contemporary compilers make use of tree structures in obtaining forms of an arithmetic expression for efficient evaluation:

Example: Binary expression tree for
(A-B) + C * E/F



Inorder traversal gives *infix* form
Preorder traversal gives *prefix* form
Postoredr traversal gives *postfix* form

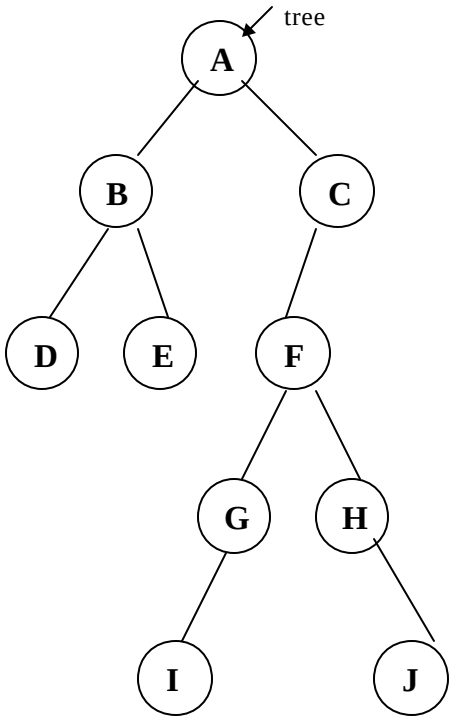
Exercise 1:

Consider the following node structure:

```
struct node {
   dataType data;
   struct node *left;
   struct node *right;
}
```

What is the output produced by the following function for the pictured tree?

```
void treewalk(struct node *tree)
{
   if (tree == NULL)
      printf("OOPS\n");
   else{
      treewalk(tree->right);
      treewalk(tree->left);
      print(tree->data);
  }
}
```



Exercise 2:

a) Insert the following names into a binary search tree and draw the resulting tree.

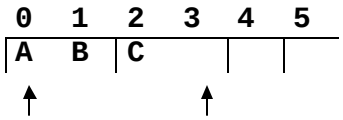
JONES
BILL
DAVE
MARY
LARRY
PAUL
PENNY
KATY
LEO
MIKE
BETTY
DON
ROGER
TOM

b) Delete JONES, PENNY, MARY

QUEUES

Consider the circular array implementation of queues. In this implementation limit the number of items in the queue to one less than the number of elements in the array (i.e. always keep one empty element at the end of the queue). Using this solution, determine the conditions for checking whether the queue is empty or full.

Adopt the following conventions for the front and rear: *front* is to point at the next item to be removed from the queue. *rear* is to point at the next available location, that is the next location to be filled. As an illustration, the following is a queue with three items:



Given the following declarations for the queue :

```
#define SIZE 100
struct queue{
   int items[SIZE];
   int front;
   int rear;
};
```

Write down the functions isEmpty(), isFull() and enqueue().

a) `isEmpty()`:

```
int isEmpty(struct queue q){



}
```

b) `isFull()`:

```
int isFull(struct queue q){



}
```

c) `enqueue()`:

```
void enqueue (struct queue *q, int item){
```