```
Select(A, 1, n, i)

q = Partition(A, 1, n)

// q = n / 2

if (i = q)
   return A[i]
if (i < q)
   Select(A, 1, n / 2 - 1, i)
else
   Select(A, n/2 + 1, …)
```

Select: T(n) = T(n/2) + Theta(n) (“best” case) => T(n) = Theta(n)
Select: T(n) = T(n-1) + Theta(n) (worst case) => T(n) = Theta(n^2)

T(n) = a T(n/b) + f(n)

Master Theorem: n^(log_b (a)) vs. f(n)

n^$\log_b a$ = n^$\lg_2 1$ = n^0 = 1 vs. Theta(n)

==============================

QuickSort: T(n) = 2 T (n/2) + Theta(n) (“best” case)

QuickSort: T(n) = T(n-1) + Theta(n) (worst case) => Theta(n^2)

Imagine we have a way to find a pivot such that we recursively call Select on a fixed percentage of the input array. We “eliminate” a percentage *a*, we look only at *b* = 1 – *a* percentage of elements in the input array.

Example, *b* = 70% of the elements in the recursive call.

T(n) = T(7/10 * n) + Theta(n) => by MT T(n) = Theta(n)

n^0 = 1 vs. f(n) = Theta(n) => MT says T(n) = linear.

Matrix Multiplication:

A, B are n x n matrices of integers. Compute C = A * B, what is T(n) = ?

$A = \begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} \qquad B = \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} \qquad C = \begin{matrix} 3 & 3 \\ 7 & 7 \end{matrix}$

C[1, 1] = 1 * 1 + 2 * 1 = 3
C[1, 2] = 1 * 1 + 2 * 1 = 3

Traditional algorithm:

```
for i=1 to n
   for j = 1 to n
      C[i,j] = 0
      for k=1 to n
          C[i,j] += A[i,k] * B[k,j]
```

T(n) = n * n (elements in C to compute) * Theta(n) = Theta(n^3)
Strassen showed how to do it in T($n^{2.8..}$) = T(n^{lg7})

Can it ever be done in Theta(n lgn)? No, it will be Omega(n^2).

T(n) = 8 T(n/2) + Theta(n^2) => MT Case 1 T(n) = n^$\log_2 8$ = n^3

7 matrix multiplications (n/2 x n/2) => 7 T(n/2)
18 matrix additions (n/2 x n/2) => 18 * n^2 / 4 = Theta(n^2)

T(n) = 7 T(n/2) + Theta(n^2) => by MT Case 1 => T(n) = Theta(n^$\log_2 7$)!