Now, Assume and using substitution method we can write,

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

= 2[ 2 T(n/4) + p] + p = 4 T(n/4) + 2p + p

= 4[ 2 T(n/8) + p] + 2p + p = 8 T(n/8) + 4p + 2p + p

…

= 2k T(n/2k) + (2k – 1) p

Here, n/2k = 1. So, k = .

So, T(n) = n + (n-1) p = n (p + 1) – 1. For large value of n, T(n) = θ(n).

**Proof by induction:**

Guess: The algorithm takes θ(n) time. To prove this we need to find O(n) and Ω(n). and from the recurrence relation we can write, T(n) = 2 T(n/2) + p; where p is a constant.

Base Case: Now, T(n) = O(n) when, T(n) ≤ cn, for some constant c > 0 and n ≥ n0 for some n0 > 0.

If n = 1; T(1) = p1 which is a constant and if n = 2; T(2) = 2 T(1) + p2 = 2p1 + p2.

So, T(2) ≤ 2c, if c ≥ (2p1 + p2).

Also, T(n) = Ω(n) when, T(n) ≥ cn for some constant c > 0 and n ≥ n0 for some n0 > 0.

Similarly if n = 1; T(1) = p3 and n = 2; T(2) = 2 T(1) + p4 = 2p3 + p4.

So, T(2) ≥ 2c’, if c’ ≤ (2p3 + p4).

We find, 2c’ ≤ T(2) ≤ 2c when c ≥ (2p1 + p2) and c’ ≤ (2p3 + p4).

Proof:

Assume T(k) ≤ ck for k<n. -------------------------------------------------------------------------------(i)

Now using the recurrence relation, T(n) = 2 \* T(n/2) + p ---------------------------------------(ii)

Substituting this in (i) we get, T(n) = 2 \* T(n/2) + 1 ≤ 2 \* c \* (n/2) + p = cn + p = c(n+1) –c + p

Now, we get T(n) = O(n) where (c+p) ≥ 0. ------------------------------------------------(iii)

Now, T(n) = Ω(n) when, T(n) ≥ cn for some constant c > 0 and n ≥ n0 for some n0 > 0.

Assume T(k) ≥ ck for k<n. -------------------------------------------------------------------------------(iv)

Now substituting (ii) in (i) we get, T(n) = 2 \* T(n/2) + p ≥ 2 \* c \* (n/2) + p = cn + p = c(n+1) –c + p

Now, we get T(n) = Ω(n) where ) where (c+p) ≤ 0. ------------------------------------------------(v)

From (iii) and (v) it is proved that T(n) = θ(n).