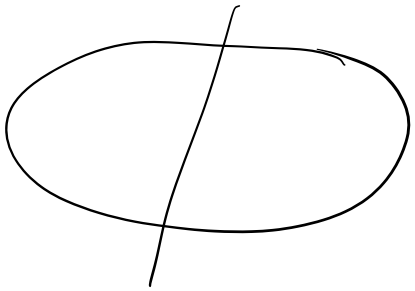
1. divide
2. conquer
3. combine

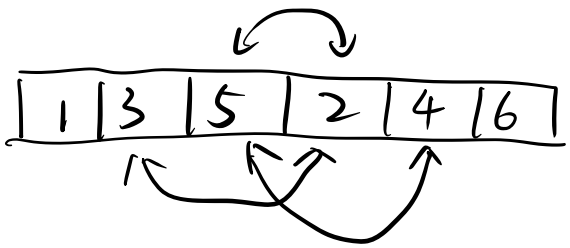


Counting Inversions

Input: an array A of n distinct integer

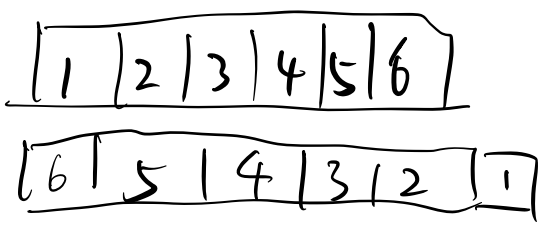
Output: the number of inversions in A.

a pair (i, j) of indices with $i < j$ and $A[i] > A[j]$



Minimum possible # inversions: 0

Maximum " : $\frac{n(n-1)}{2}$



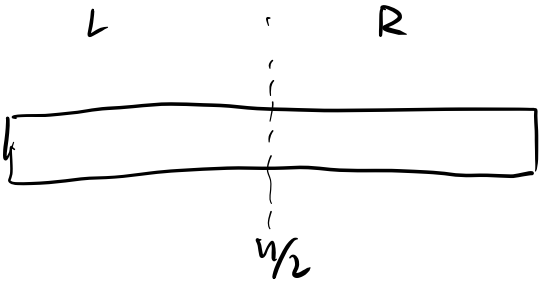
You : apple > banana > orange.

friend : banana > apple > orange

1 2 3

brute-force $O(n^2)$ time

divide & conquer $O(n \log n)$ time



left inversion: $i < j \leq n/2$

right inversion: $n/2 < i < j$

split inversion: $i \leq n/2 < j$

```
CountInv(A)    //A is an array of n distinct integers
  If n≤1:
     return 0
  else:
     leftInv = CountInv(left half of A)
     rightInv = CountInv(right half of A)
     splitInv = CountSplitInv(A)  ⟶ O(n)

  return leftInv + rightInv + splitInv
```

split inversions $O(n^2)$.

(3, 2) (5, 2) (7, 2)

(5, 4) (7, 4)

(7, 6)

C[1 ... n/2] D[1 ... n/2]

i=1 j=1

SplitInv=0

for k=1 to n:

If C[i] < D[j]:

$O(n)$ i = i+1

else // C[i] > D[j]

SplitInv = SplitInv + n/2 − i + 1

j = j+1

$T(n)$ Sort-and-CountInv(A)

$T(1)=a1$ 1. if n≤1:

return (A, 0)

2. else

$T(n/2)$ 3. (C, leftInv) = Sort-and-CountInv(left half of A)

$T(n/2)$ 4. (D, rightInv) = sort-and-CountInv(right half of A)

$O(n)$ 5 (B, splitInv) = merge-and-CountSplitInv(C, D)

6. return (B, leftInv + rightInv + SplitInv)

Merge-and-CountSplitInv(C, D) // C and D has length n/2

```
1. i=1; j=1; SplitInv=0
2. for k=1 to n
3.     if C[i] < D[j]:
4.         B[k] = C[i]
5.         i = i+1
6.     else          // C[i] > D[j]
7.         B[k] = D[j]
8.         SplitInv = SplitInv + (n/2 - i + 1)
9.         j = j+1
10. return (B, splitInv)
```

O(n)

$$\begin{cases} T(1) = O(1) \\ T(n) = 2T(n/2) + O(n) \end{cases} \Rightarrow T(n) = O(n \lg n)$$

Computational Geometry

Closest Pair Problem

Input: a set P of points in a plane.

$P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$

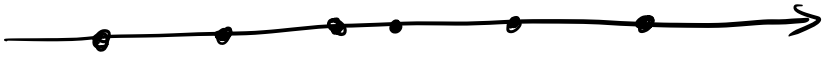
Output: the pair (p_i, p_j) with smallest distance $d(p_i, p_j)$

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Brute-force: $O(n^2)$ time

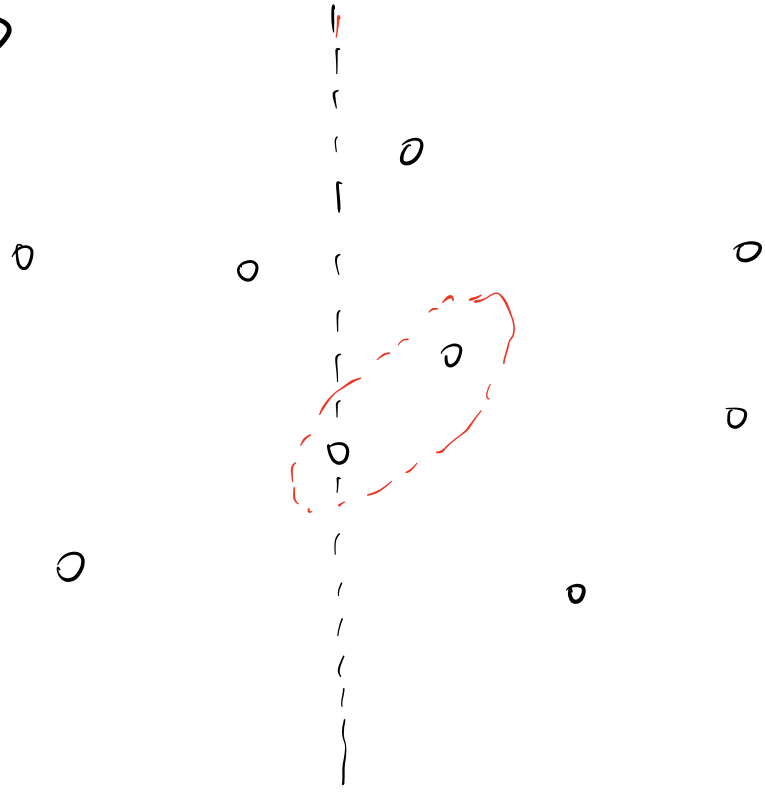
D & C: $O(n \lg n)$ time.

1-D



sorting + scaning $O(n \lg n)$

2-D



left pair:

right pair

split pair

Assumption: $x_i \neq x_j$, $y_i \neq y_j$ for $i \neq j$

P_x : sorted by x-coord
P_y : sorted by y-coord

$T(n)$: ClosestPair(P_x, P_y)

$T(3)=O(1)$ 1. if $n \leq 3$.
2. trivial

obtain from P_x and P_y in $O(n)$
3. L_x = pts on left half plane, sorted by x-coord
4. L_y = pts on left half plane, sorted by y-coord.
5. R_x = " " right " " , sorted by x-coord.
6. R_y = " " right " " , sorted by y-coord.

$T(n/2)$ 7. (l_1, l_2) = ClosestPair(L_x, L_y)

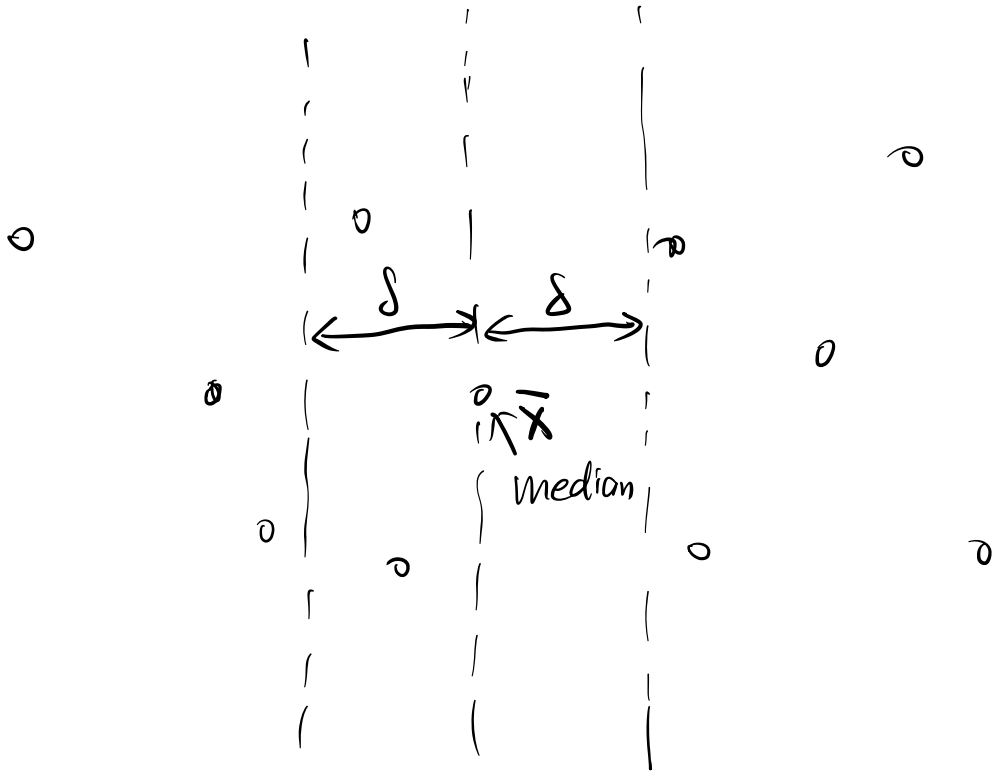
$T(n/2)$ 8. (r_1, r_2) = ClosestPair(R_x, R_y)

$O(1)$ 8.5. $\delta = \min\{d(l_1, l_2), d(r_1, r_2)\}$

$O(n)$ ← 9. (s_1, s_2) = ClosestSplitPair(P_x, P_y, δ)

10. return best of (l_1, l_2), (r_1, r_2), (s_1, s_2)

only consider split pairs whose distance $< \delta$ and return the smallest among them



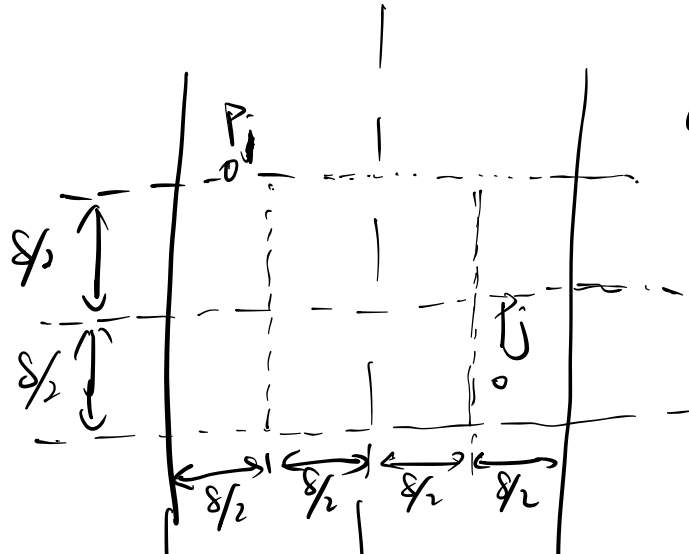
a split pair (p_i, p_j)

$$d(p_i, p_j) < \delta \qquad x_i \le \bar{x} < x_j$$

$$\Rightarrow |x_i - x_j| < \delta$$

$$\Rightarrow x_j - \delta < x_i, x_j < x_i + \delta$$

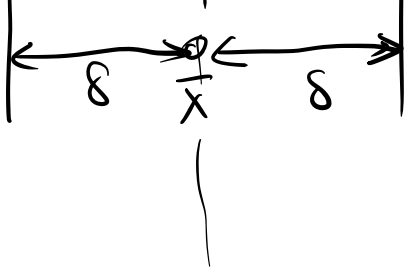
$$\Rightarrow \bar{x} - \delta < x_i, x_j < \bar{x} + \delta$$



$d(p_i, p_j) < \delta$

$\Rightarrow$ in the strip, at most 6 pts between p_i and p_j

$\frac{\delta}{2} \cdot \sqrt{2} = \frac{\delta}{\sqrt{2}} < \delta$



with respect to y-coor

Closest Split Pair (P_x, P_y, δ)

O(n) 1. $\bar{x}$ = largest x-coord of pts in left half plane

obtain from P_y in O(n) 2. S_y = $\leq$ points $q_1, q_2, \ldots, q_t$ with

3. x-coord between $\bar{x}-\delta$ and $\bar{x}+\delta$,

4. sorted by y-coord]

O(1) 5. best = δ

6. bestPair = Null

n × 7 × O(1) = O(n)

7. for i=1 to t-1

8. for j=1 to min{7, t-i}

9. if d(q_i, q_{i+j}) < best

10. best = d(q_i, q_{i+j})

11. bestPair = (q_i, q_{i+j})

12. return bestPair

$$\begin{cases} T(n) = 2\cdot T(n/2) + O(n) \\ T(3) = O(1) \end{cases} \Rightarrow T(n) = O(n \log n)$$

$$T(n) = a \cdot T(n/b) + f(n) \qquad T(1) = O(1)$$

$T(n) = 2 \cdot T(n/2) + cn$

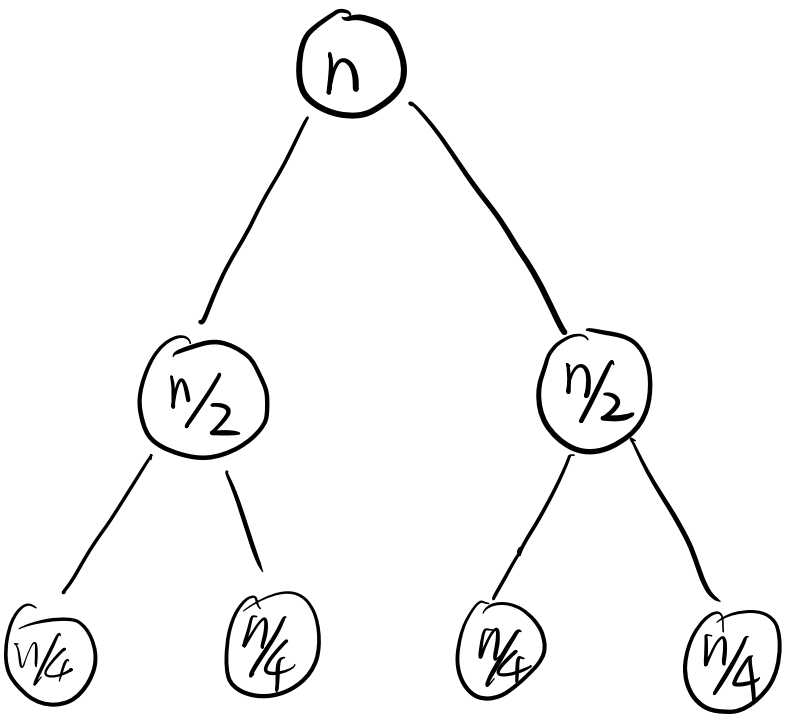
# of recursive calls

input size shrinkage factor

running time of "divide" and "combine" step

Recursion tree method

$$\begin{cases} T(n) = 2 \cdot T(n/2) + cn \\ T(1) = c_1 \end{cases}$$



n

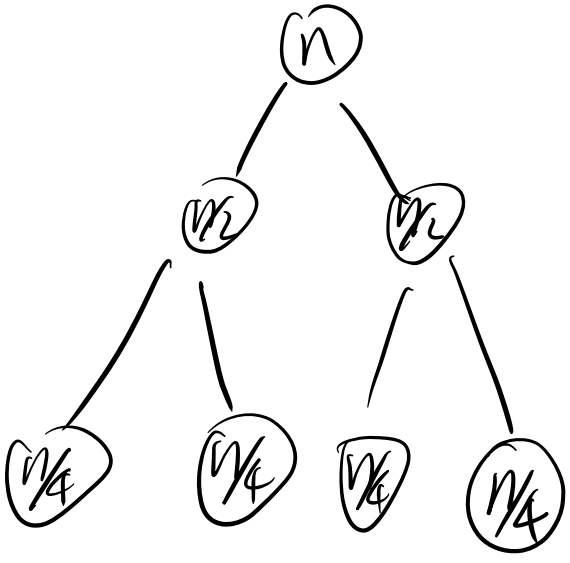
$2 \cdot n/2 = n$

$4 \cdot n/4 = n$

$\cdots\cdots \quad 2^i \cdot \frac{n}{2^i} = n$

$\text{total} = n \cdot \lg_2 n = O(n \lg n)$

$$\begin{cases} T(n) = 2 \cdot T(n/2) + n^2 \\ T(1) = 1 \end{cases}$$



n^2

$2 \cdot \left(\frac{n}{2}\right)^2 = \frac{n^2}{2}$

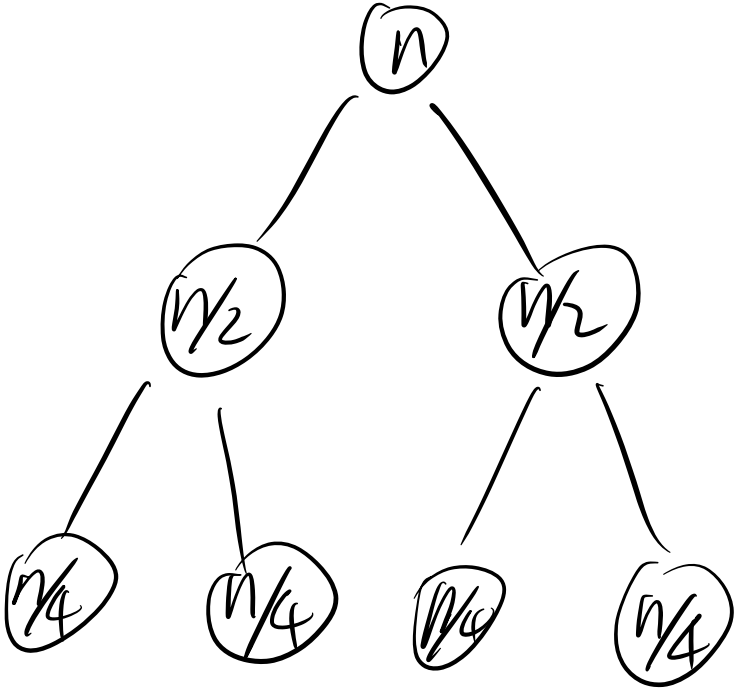
$4 \cdot \left(\frac{n}{4}\right)^2 = \frac{n^2}{4}$

i

$2^i \cdot \left(\frac{n}{2^i}\right)^2 = \frac{n^2}{2^i}$

$$\text{total} = \sum_{i=0}^{\lg_2 n} \frac{n^2}{2^i} \leq 2 \cdot n^2 = O(n^2)$$

$$\begin{cases} T(n) = 2 \cdot T(n/2) + \sqrt{n} \\ T(1) = 1 \end{cases}$$



$\sqrt{n}$

$2 \cdot \sqrt{\frac{n}{2}} = \sqrt{2} \cdot \sqrt{n}$

$4 \cdot \sqrt{\frac{n}{4}} = 2 \cdot \sqrt{n}$

i $\qquad 2^i \cdot \sqrt{\frac{n}{2^i}} = (\sqrt{2})^i \cdot \sqrt{n}$

$i = \lg_2 n \longrightarrow (\sqrt{2})^{\lg_2 n} \sqrt{n}$

$= \sqrt{n} \cdot \sqrt{n} = n$

$$\text{total} = \sum_{i=0}^{\lg_2 n} (\sqrt{2})^i \cdot \sqrt{n}$$

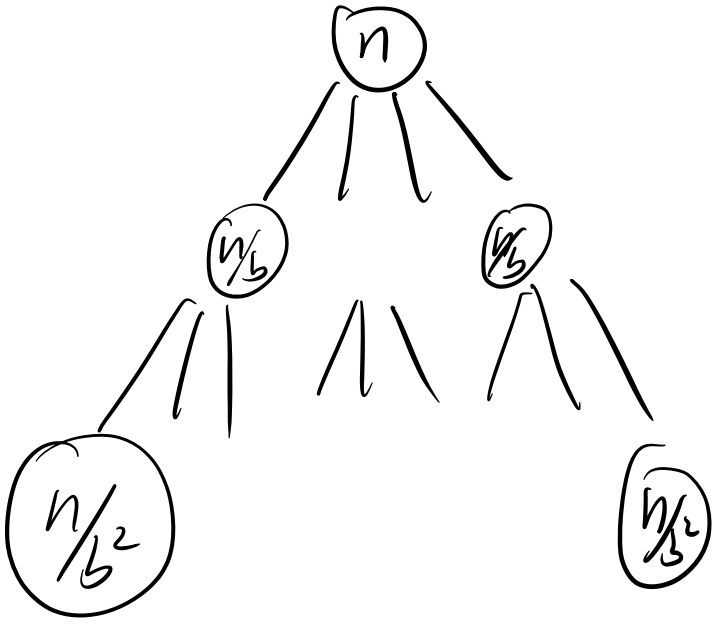
$$= \sqrt{n} \, \frac{1 - (\sqrt{2})^{\lg_2 n + 1}}{1 - \sqrt{2}}$$

$$\leq \sqrt{n} \, \frac{(\sqrt{2})^{\lg_2 n + 1}}{\sqrt{2} - 1}$$

$$\leq \frac{\sqrt{2}}{\sqrt{2}-1} \cdot \sqrt{n} \cdot \sqrt{n} = O(n)$$

Master Theorem

$$\begin{cases} T(n) = aT(n/b) + f(n) \\ T(1) = 1 \end{cases}$$



$f(n)$ $f(n)$

$a \cdot f(n/b)$ $r \cdot f(n)$

$a^2 \cdot f(n/b^2)$ $r^2 f(n)$

⋮

$a^i \cdot f(\frac{n}{b^i})$ $r^i f(n)$

$i = \log_b n$ $a^i = a^{\log_b n} = n^{\log_b a} \cdot O(1)$

$\sum_{i=0}^{\log_b n - 1} a^i f(n/b^i) = a^{i+1} \cdot f(n)$

$a \cdot f(n/b^{i-1}) = a^{i-1} \cdot a \cdot f\left(\frac{n}{b^{i-1}} \cdot \frac{1}{b}\right)$

$= a^{i-1} \cdot f\left(\frac{n}{b^{i-1}}\right)$

$= f(n)$

1. If $a \cdot f(n/b) = f(n)$, then

$$T(n) = f(n) \cdot \log_b n = \Theta(f(n) \cdot \lg n)$$

2. If $a \cdot f(n/b) = r \cdot f(n)$ for some $r < 1$.

$$T(n) = \Theta(f(n))$$

$$\lesssim \frac{1-r}{r} f(n)$$

3. If $a \cdot f(n/b) = r \cdot f(n)$ for some $r > 1$,

$$T(n) = \Theta(n^{\log_b a})$$

Substitution method.

Guess and prove by induction

$$\begin{cases} T(1) = 1 \\ T(n) = 2 \cdot T(\lfloor n/2 \rfloor) + n \end{cases}$$

$$T(n) = O(n \lg_2 n)$$

$$\Downarrow$$

$$\exists\, C > 0,\ \forall\, n \geq n_0,\ T(n) \leq C \cdot n \lg_2 n$$

1. base case : $n = 2$ $\quad C = 2$

$$T(2) = 2T(1) + 2 = 4 \leq 2 \cdot 2 \cdot \lg_2 2$$

$$4$$

2. Inductive hypothesis

$$\forall\, 2 \leq m \leq n-1)$$

$$T(m) \leq C \cdot m \cdot \lg_2 m$$

3. Inductive step

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

$$\leq 2c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + n$$

$$\leq cn(\lg n - 1) + n$$

$$\leq cn\lg n - (c-1)n$$

$$\leq cn\lg n$$

$$\begin{cases} T(n) = 2T(\lfloor n/2 \rfloor) + n \\ T(1) = 1 \end{cases}$$

$T(n) = O(n)$

$\Downarrow$

$\exists c > 0, \forall n \geq n_0, \quad T(n) \leq cn$

$\exists c > 0, \forall n \geq n_0, T(n) \leq cn$

base case: $n = 1$

$T(1) = 1 \leq c$

① hypothesis

$m \leq n-1, \quad T(m) \leq c \cdot m$

$$T(n) = 2 \cdot T(\lfloor n/2 \rfloor) + n$$

$$\leq 2 \cdot c \cdot \lfloor n/2 \rfloor + n$$

$$\leq cn + n$$

$$\leq (c+1)n \quad \times$$

$T(n) \leq cn \quad = O(n)$